

UNIVERZA V LJUBLJANI  
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miha Strel

**Integracija ogrodja Medius Vaadin  
Common na poslovnih portalih**

DIPLOMSKO DELO  
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Marjan Krisper

Ljubljana, 2012

Rezultati diplomskega dela so intelektualna lastnina Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje Fakultete za računalništvo in informatiko ter mentorja.

*Besedilo je oblikovano z urejevalnikom besedil  $\text{\LaTeX}$ .*



Št. naloge: 01793/2011

Datum: 05.12.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MIHA STREL**

Naslov: **INTEGRACIJA OGRODJA MEDIUS VAADIN COMMON NA POSLOVNIH PORTALIH**

**INTEGRATION OF MEDIUS VAADIN COMMON FRAMEWORK ON BUSINESS PORTALS**

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

V okviru diplomske naloge proučite poslovne portale različnih ponudnikov in raziščite možnosti integracije lastnega ogrodja Medius Vaadin Common na najbolj uporabljanih portalih. Ker podjetja in ustanove za potrebe informacijske podpore svojih poslovnih procesov danes uporabljajo širok nabor različnih informacijskih tehnologij, ki pokrivajo posamezna funkcionalna področja, obstaja potreba, da uporabniki glede na svojo vlogo v podjetjih lahko dostopajo do informacij iz različnih informacijskih sistemov na enem samem mestu. Pri tem upoštevajte da lastno ogrodje predstavlja standardizirano podlago za razvoj kot tudi za enotno uporabniško izkušnjo. Definirajte referenčno poslovno aplikacijo in opišite integracijo ogrodja po posameznih arhitekturnih nivojih. V zaključku ovrednotite opravljeno integracijo.

Mentor:

prof. dr. Marjan Krisper



Dekan:

prof. dr. Nikolaj Zimic

# IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani      Miha Strel,

z vpisno številko      63050110,

sem avtor diplomskega dela z naslovom: Integracija ogrodja Medius Vaadin

Common na poslovnih portalih

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Marjana Krisperja
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 27.1.2012

Podpis avtorja:

# Zahvala

Zahvaljujem se vsem sodelavcem podjetja Medius, ki so s svojimi izkušnjami in nasveti kakorkoli pripomogli pri nastajanju te diplomske naloge, še posebej pa g. Martinu Bolčini in g. Petru Brajaku.

Zahvaljujem se tudi mentorju prof. dr. Marjanu Krisperju in asistentki dr. Ani Šaša Bastinos za njuno pomoč pri izdelavi diplomskega dela.

*Staršem*

# Kazalo

|                                                                   |           |
|-------------------------------------------------------------------|-----------|
| <b>Povzetek</b>                                                   | <b>2</b>  |
| <b>Abstract</b>                                                   | <b>4</b>  |
| <b>1 Uvod</b>                                                     | <b>5</b>  |
| <b>2 Poslovni portali</b>                                         | <b>7</b>  |
| 2.1 Splošen pregled . . . . .                                     | 7         |
| 2.1.1 Definicija poslovnega portala . . . . .                     | 7         |
| 2.1.2 Razvoj portalov v preteklosti . . . . .                     | 10        |
| 2.1.3 Pregled trenutnih ponudnikov . . . . .                      | 12        |
| 2.2 Java EE portali . . . . .                                     | 12        |
| 2.2.1 Java Enterprise Edition in Java Enterprise Edition strežnik | 13        |
| 2.2.2 Portlet . . . . .                                           | 16        |
| 2.2.2.1 Vsebovalnik portleta . . . . .                            | 18        |
| 2.2.2.2 Življenjski cikel portleta . . . . .                      | 18        |
| 2.2.2.3 Standard JSR-168 (Portlet 1.0) . . . . .                  | 21        |
| 2.2.2.4 Standard JSR-286 (Portlet 2.0) . . . . .                  | 26        |
| 2.2.2.5 Standard WSRP . . . . .                                   | 29        |
| 2.2.3 Primeri Java EE portalov . . . . .                          | 30        |
| 2.2.3.1 Liferay Portal® . . . . .                                 | 30        |
| 2.2.3.2 IBM WebSphere Portal® . . . . .                           | 32        |
| <b>3 Medius Vaadin Common</b>                                     | <b>34</b> |
| 3.1 Namen ogrodja . . . . .                                       | 34        |
| 3.2 Komponente ogrodja . . . . .                                  | 35        |
| 3.2.1 Izvajalno okolje . . . . .                                  | 35        |
| 3.2.2 Podatkovni nivo . . . . .                                   | 37        |
| 3.2.3 Poslovni nivo in spletni nivo . . . . .                     | 38        |
| 3.2.3.1 Objektno modeliranje podatkov . . . . .                   | 38        |

|          |                                                                  |            |
|----------|------------------------------------------------------------------|------------|
| 3.2.3.2  | Modeliranje grafičnega uporabniškega vmesnika                    | 44         |
| 3.2.3.3  | Povezava med poslovnim nivojem in nivojem<br>odjemalca . . . . . | 52         |
| 3.2.4    | Nivo odjemalca . . . . .                                         | 53         |
| <b>4</b> | <b>Integracija Medius Vaadin Common na portalih</b>              | <b>59</b>  |
| 4.1      | Priprava primera uporabe . . . . .                               | 59         |
| 4.1.1    | Podatkovni nivo . . . . .                                        | 60         |
| 4.1.2    | Poslovni in spletni nivo . . . . .                               | 62         |
| 4.1.3    | Nivo odjemalca . . . . .                                         | 65         |
| 4.2      | Analiza zahtev za oceno uspeha integracije . . . . .             | 67         |
| 4.3      | Integracija na Liferay Portalu® . . . . .                        | 71         |
| 4.3.1    | Analiza podprtih standardov in priprava okolja . . . . .         | 71         |
| 4.3.2    | Poskus integracije . . . . .                                     | 72         |
| 4.3.2.1  | Podatkovni nivo . . . . .                                        | 72         |
| 4.3.2.2  | Poslovni nivo . . . . .                                          | 72         |
| 4.3.2.3  | Spletni nivo . . . . .                                           | 77         |
| 4.3.3    | Analiza delovanja . . . . .                                      | 92         |
| 4.4      | Integracija na IBM Websphere Portalu® . . . . .                  | 98         |
| 4.4.1    | Analiza podprtih standardov . . . . .                            | 98         |
| 4.4.2    | Priprava okolja . . . . .                                        | 99         |
| 4.4.3    | Poskus integracije . . . . .                                     | 102        |
| 4.4.3.1  | Podatkovni nivo . . . . .                                        | 102        |
| 4.4.3.2  | Poslovni nivo . . . . .                                          | 103        |
| 4.4.3.3  | Spletni nivo . . . . .                                           | 111        |
| 4.4.4    | Analiza delovanja . . . . .                                      | 124        |
| <b>5</b> | <b>Zaključek</b>                                                 | <b>130</b> |
| <b>A</b> | <b>Izvorna koda referenčne implementacije</b>                    | <b>132</b> |
| A.1      | Skripta za izdelavo podatkovne baze MySQL . . . . .              | 132        |
| A.2      | Izvorna koda entitetnih zrn . . . . .                            | 136        |
| A.2.1    | Entitetno zrno Post . . . . .                                    | 136        |
| A.2.2    | Entitetno zrno Address . . . . .                                 | 140        |
| A.2.3    | Entitetno zrno Customer . . . . .                                | 146        |
| A.2.4    | Entitetno zrno Order . . . . .                                   | 152        |
| A.2.5    | Entitetno zrno OrderItem . . . . .                               | 158        |
| A.2.6    | Entitetno zrno Product . . . . .                                 | 162        |

|          |                                                                 |            |
|----------|-----------------------------------------------------------------|------------|
| <b>B</b> | <b>Izvorna koda integracije na portal Liferay®</b>              | <b>167</b> |
| B.1      | Namestitvena datoteka persistence.xml . . . . .                 | 167        |
| <b>C</b> | <b>Izvorna koda integracije na portal IBM Websphere Portal®</b> | <b>169</b> |
| C.1      | Skripta za izdelavo podatkovne baze IBM DB2® . . . . .          | 169        |
|          | <b>Seznam slik</b>                                              | <b>175</b> |
|          | <b>Seznam izvirne kode</b>                                      | <b>178</b> |
|          | <b>Literatura</b>                                               | <b>180</b> |

# Seznam uporabljenih kratic in simbolov

**JSR** (ang. Java Specification Request) - Oznaka za formalne dokumente, ki opisujejo specifikacije in tehnologije javine platforme.

**API** (ang. Application programming interface) - Programski vmesnik, vmesnik, ki zagotavlja, da ima računalniški program na razpolago funkcije drugega računalniškega programa [5]

**JPA** (ang. Java Persistence API) - Programski vmesnik, ki v Javi EE skrbi za preslikovanje podatkov med relacijsko bazo in objektno predstavitvijo teh podatkov v poslovni aplikaciji

**JCP** (ang. Java Community Process) - Organizacija, ki skrbi za sprejemanje standardov Jave.

**WAR** (ang. Web Archive) - Format stisnjene datoteke, ki se uporablja za distribucijo spletnih aplikacij. Ima predpisano strukturo datotek znotraj stisnjene datoteke.

**EAR** (ang. Enterprise Archive) - Format stisnjene datoteke, ki se uporablja za distribucijo spletnih aplikacij. Del EAR datotek so lahko tudi WAR datoteke, prav tako pa imajo tudi predpisano strukturo znotraj stisnjene datoteke.

**WSRP** (ang. Web Services for Remote Portlets) - Standard, ki opisuje način distribucije portletov v obliki spletnih storitev.

**WSDL** (ang. Web Services Description Language) - Jezik za opis spletnih storitev v obliki XML sporočil.

- IDE** (ang. Integrated development environment) - Integrirano razvojno okolje, namenjeno programiranju, ki navadno vsebuje urejevalnik besedila, prevajalnik, povezovalnik in iskalnik napak (npr. Eclipse, RAD) [5]
- ORM** (ang. Object-relational mapping) - Tehnika za preslikavo podatkov med relacijsko podatkovno bazo in objektno predstavitvijo teh podatkov v aplikaciji.
- EJB** (ang. Enterprise Java Beans) - Poslovno Javansko zrno; Vsebuje attribute in metode za implementacijo poslovne logike.
- POJO** (ang. Plain Old Java Object) - Navaden objekt v programskem jeziku Java.
- JAR** (ang. Java Archive) - Format stisnjene datoteke za distribucijo programskih knjižnic.
- RAD** (ang. Rational Application Developer) - Razvojno okolje (IDE) podjetja IBM, namenjeno razvoju javanskih aplikacij za družino aplikacijskih sistemov WebSphere.

# Povzetek

Podjetja in ustanove za potrebe informacijske podpore svojih poslovnih procesov danes uporabljajo širok nabor različnih informacijskih tehnologij, ki pokrivajo posamezna funkcionalna področja, pri tem pa se pojavi potreba po tem, da bi zaposleni glede na svojo vlogo v podjetjih lahko dostopali do informacij iz različnih informacijskih sistemov na enem samem mestu. To lahko dosežemo z uporabo poslovnih portalov, ki navidezno združijo uporabniške vmesnike posameznih aplikacijskih podsistemov na tak način, da dajejo uporabniku vtis, da uporabljajo le eno aplikacijo. To storijo s pomočjo t.i. portletov oz. programskih komponent v obliki uporabniških vmesnikov, pri čemer en portlet predstavlja določeno funkcionalnost podsistema.

Da pa bi lahko učinkovito razvili portlete, potrebujemo razvojno okolje in ogrodje, ki bi nam omogočilo razvoj aplikacije v enem programskem jeziku, tako na strani uporabniškega vmesnika, kot poslovne logike. Osnovano bi moralo biti na splošno sprejetih standardih in omogočiti uporabo različnih podatkovnih baz, s čimer bi zagotovili prenosljivost med različnimi ponudniki portalov. Primer takega ogrodja smo razvili v podjetju Medius<sup>®</sup>, namenjeno pa je bilo razvoju klasičnih poslovnih aplikacij (torej brez portletov), pri čemer nas je kasneje zanimalo tudi, ali je mogoče to ogrodje integrirati na poslovne portale in kakšne spremembe so pri tem potrebne. V diplomski nalogi se spoznamo z osnovno strukturo ogrodja po posameznih arhitekturnih nivojih in zahtevami, ki jih mora izpolnjevati ciljni sistem, da je mogoča uspešna integracija sistema. Nato sledi definicija referenčne poslovne aplikacije zasnovane na tem ogrodju in oblikovanje kriterijev za oceno uspeha integracije. V osrednjem delu diplomske naloge opisuje poskus integracije ogrodja na poslovna portala Liferay Portal<sup>®</sup> in IBM WebSphere Portal<sup>®</sup>. Pri vsakem je predstavljen proces osnovne vzpostavitve sistema, nato pa po korakih sledi opis integracije ogrodja po posameznih arhitekturnih nivojih, pri čemer so podrobno opisane vse spremembe in prilagoditve, ki so potrebne za uspešno izvedbo integracije. Na koncu sledi ocena uspeha integracije.

Izkaže se, da je integracija na oba portala uspešna, pri tem pa so potrebne

spremembe na samem ogrodju minimalne in so v obeh primerih večinoma povezane z različnimi podrobnostmi implementacije določenih standardov na podatkovnem in poslovnem nivoju (predvsem z razlikami pri aplikacijskih strežnikih), kar pomeni, da samega jedra ogrodja ni potrebno spreminjati.

**Ključne besede:**

poslovni portal, portlet, anotacije, okolje Java EE, integracija, standard JSR-286, Liferay Portal<sup>®</sup>, IBM WebSphere Portal<sup>®</sup>, Medius Vaadin Common<sup>®</sup>

# Abstract

To cover all aspects of business processes, enterprises today use an array of information technologies, covering information management across different business functions. For the enterprise to ensure employee access to all required information and maximisation of efficiency, a unified access point should be provided. One way of achieving this is by using enterprise portals, which unite graphical user interfaces of different information subsystems on the presentation level, thus giving employees the feeling, that they are using just one system. Portals achieve this by using so-called portlets or programmatic components in the form of graphical user interfaces in which case each portlet represents one functionality of the information subsystem.

In order to effectively develop the portlets, we need a development environment and framework, which would enable us to develop software by using only one programming language (for development of client and server side). It would have to be based on the existing enterprise software specifications and should enable us to use different databases, making it possible to use the software on different enterprise portals. One example of such framework was developed by Medius<sup>®</sup> (a company specialising in developing Java based middleware). It was designed for development of classic enterprise applications (not to be a part of portal at first), but at a later point a possibility of integrating it on enterprise portals was also considered. The main question considering the integration was the number of changes required to achieve a fully functional portlet, built on top of this platform. The answer to this question is the main focus point of this thesis.

The thesis consists of the introduction to the framework and requirements that have to be fulfilled in order to achieve a successful integration. A simple reference enterprise application built upon the framework is then proposed, as a basis for the integration to enterprise portals. A set of criteria to evaluate the success of integration is also defined. In the central part of the thesis, an attempt of integrating the framework to Liferay Portal<sup>®</sup> and IBM WebSphere Portal<sup>®</sup> is described in detail. Each attempt includes the description of esta-

blishing the basic environment and the steps of integration, divided among the architectural tiers of the application. Each step contains all the details and changes that have to be considered at each tier. Each attempt of integration is followed by an evaluation.

As it turns out, integration can be successfully achieved on both enterprise portals. In each case, the needed changes to achieve this are minimal and in both cases refer to differences between the implementation of certain specifications at the data and business tier (most have to do with differences between application servers). This means that the framework itself does not have to be altered in a way, that would require us to rewrite large portions of code and can be used for the development of software for enterprise portals.

**Key words:**

Enterprise Portal, portlet, annotations, Java EE environment, integration, JSR-286 standard, Liferay Portal<sup>®</sup>, IBM WebSphere Portal<sup>®</sup>, Medius Vaadin Common<sup>®</sup>

# Poglavje 1

## Uvod

Podjetja in ustanove za potrebe informacijske podpore svojih poslovnih procesov danes uporabljajo širok nabor različnih informacijskih tehnologij, ki pokrivajo posamezna funkcionalna področja. Ker pa so le-ta med seboj tipično prepletena, se pojavi potreba po tem, da bi zaposlenci glede na svojo vlogo v podjetjih lahko dostopali do informacij iz različnih informacijskih sistemov na kar najbolj enostaven način. Zaželeno je, da ima zaposleni pregled nad vsemi informacijami, ki jih potrebuje za svoje delo, na enem mestu in da so te predstavljene na način, ki daje vtis, da gre za enoten informacijski sistem (torej da ponuja enotno uporabniško izkušnjo). S tem se za uporabnika poenostavi iskanje in razumevanje informacij ter posledično omogoči večjo učinkovitost.

Kot možen odgovor na te potrebe so ponudniki aplikacijskih sistemov razvili t.i. poslovne portale. To so ogrodja, ki omogočajo integracijo obstoječih sistemov in razvoj novih funkcionalnosti, pri čimer uporabniku nudijo enoten način za dostop do informacij, največkrat v obliki spletnega uporabniškega vmesnika. Integracijo omogočajo z implementacijo splošno sprejetih standardov, med katerimi je ključen standard za portlet. Portlet je zaključena komponenta za prikaz uporabniškega vmesnika, ki jo je mogoče na enostaven način vključiti na portal, prikazuje pa informacije iz določenega aplikacijskega podsistema. Uporabniški vmesnik, ki ga na portalu vidi njegov uporabnik, tako običajno izgleda kot spletna stran, ki je sestavljena iz različnih portletov, ti pa prikazujejo informacije, ki so za uporabnika pomembne. Uporabnik lahko tudi doda ali skrije posamezne portlete.

Za razvoj novih funkcionalnosti vsak ponudnik običajno poleg portala ponudi določen način razvoja aplikacij na osnovi komercialnih ogrodij in razvijalskih okolij, ki jih ima v svoji prodajni ponudbi. Slabost teh orodij pa je v tem, da se morajo razvijalci takih aplikacijskih rešitev naučiti progra-

miranja na ponujeni platformi, kar poleg samega nakupa razvijalskih orodij pomeni še dodaten strošek. Poleg tega so take platforme običajno omejene le na enega ponudnika portalov, kar ima za posledico lahko t.i. „vendor locking“. Lahko se pojavijo še dodatne slabosti, razvoj uporabniškega vmesnika lahko npr. na določenem ogrodju poteka v drugačnem programskem jeziku kot razvoj poslovne logike, torej mora razvijalec dobro poznati širok nabor različnih tehnologij.

Kot odgovor na omenjene slabosti bi si želeli ogrodje, ki bi omogočalo razvoj aplikacije v enem programskem jeziku, tako na strani uporabniškega vmesnika, kot poslovne logike. Osnovano bi moralo biti na splošno sprejetih standardih in omogočiti uporabo različnih podatkovnih baz, s čimer bi zagotovili prenosljivost med različnimi ponudniki portalov. Poleg tega pa bi bila dodatna zahteva, da omogoča hiter in učinkovit razvoj. Tako ogrodje bi razvijalcem zagotavljalo možnost enotnega načina razvoja za različne portale in posledično večjo konkurenčnost na trgu, naročniku sistema osnovanega na tem ogrodju pa bi zagotavljalo hitro implementacijo zahtev, skladnost s standardi in posledično možnost prenosa na drug portal, v primeru da bi se iz kateregakoli razloga odločili za menjavo ponudnika portala (npr. ob prenovi informacijskega sistema v podjetju, opustitvi podpore portalu s strani proizvajalca ipd.).

Kot primer takega ogrodja smo v podjetju Medius razvili programsko rešitev poimenovano Medius Vaadin Common, ki naj bi ustrezala tem zahtevam. Ogrodje je bilo sprva namenjeno le razvoju klasičnih aplikacij, ki bi tekale na aplikacijskem strežniku, kasneje pa se je pojavil tudi interes, da bi se ga lahko uporabilo tudi na poslovnih portalih. Ker je bilo že v začetku zasnovano na odprtokodnih programskih komponentah, ki so osnovane na splošno sprejetih standardih, nas je v podjetju zanimalo, ali je mogoče to ogrodje integrirati v poslovne portale in kakšne so morebitne potrebne prilagoditve, da bi to dosegli. Namen te diplomske naloge je torej preveriti, ali obstoječi portali različnih ponudnikov podpirajo standarde, ki jih za svoje delovanje potrebuje naše ogrodje. Za cilj smo si postavili poiskati presek standardov, ki bi omogočali integracijo ogrodja in definiranje potrebnih prilagoditev, ki bi omogočile delovanje na posameznem portalu.

## Poglavje 2

# Poslovni portali

### 2.1 Splošen pregled

#### 2.1.1 Definicija poslovnega portala

Leta 1998 sta analitka Christopher C. Shilakes in Julie Tylman (oba iz podjetja Merrill Lynch) prva predstavila koncept portala [1]. Njuna definicija se je glasila:

„Poslovni informacijski portali (ang. enterprise information portals) so aplikacije, ki podjetjem omogočijo uporabo notranjih in zunanjih informacij ter uporabnikom omogočijo enoten dostop do personaliziranih informacij, ki jih potrebujejo za sprejemanje poslovnih odločitev, podprtih z informacijami.

So nabor aplikacij, ki združujejo, obvladujejo, analizirajo in distribuirajo informacije v podjetju in zunaj njega (to vključuje poslovno inteligenco, obvladovanje vsebine, podatkovna skladišča ter obvladovanje podatkov).“

Opredelila sta tudi ključne značilnosti poslovnega portala:

- poslovni portali uporabljajo „push“ in „pull“ tehnologije za prenos informacij do uporabnikov preko standardnega spletnega vmesnika (z izrazom „pull“ označimo tehnologije, pri katerih določene informacije eksplicitno zahteva uporabnik sam, medtem ko „push“ označuje tehnologije, ki uporabniku zagotavljajo informacije, ki jih ta lahko ali pa tudi ne potrebuje, pri tem pa jih uporabnik ne zahteva eksplicitno)
- poslovni portali omogočajo interaktivnost

- poslovni portali so vertikalno usmerjeni, t.j. so pogosto nabor aplikacij namenjenim specifičnim področjem v industriji ali pa le specifičnim poslovnim funkcijam v podjetju
- poslovni portali integrirajo raznolike aplikacije, vključno z aplikacijami za obvladovanje vsebin, poslovno inteligenco, obvladovanje podatkov in zunanjih informacij v enoten sistem, ki omogoča deljenje, obvladovanje in vzdrževanje informacij na enem centralnem uporabniškem vmesniku. Poslovni portal je zmožen dostopati do notranjih in zunanjih virov podatkov in informacij ter podpira dvosmerno izmenjavo informacij s temi viri. Poleg tega zna uporabiti pridobljene podatke in informacije za nadaljnjo obdelavo in analizo.

Shilakes in Tylman sta tako poskusila oblikovati celovito definicijo poslovnega portala, tako na strani osnovnih funkcionalnosti kot tudi na strani podsistemov, ki naj bi bili del portala. Čemur njuna definicija ne daje poudarka, pa je podpora delu v skupinah, temveč definira portal le kot vstopno točko do podatkov in informacij, ki nato lahko služijo kot osnova za sprejemanje odločitev v poslovnem okolju.

Definicijo, ki večji poudarek da povezovanju uporabnikov in podpori za delo v skupinah je predstavil Gerry Murray (IDC), ki je poudaril, da portali, ki se osredotočajo le na vsebinski vidik, niso ustrezni za poslovna okolja, saj naj bi nas portali povezovali ne le z vsem kar rabimo, temveč tudi z vsakomer, ki ga potrebujemo. Del portala naj bi tako postali tudi aplikativni sistemi za podporo delu v skupinah. Murray je tudi razdelil poslovne portale na štiri vrste, ki jih je poimenoval z naslednjimi izrazi in definicijami:

- Enterprise information portals (so klasični poslovni portali, ki povezujejo ljudi z informacijami tako, da organizirajo velike zbirke vsebin, glede na temo oz. področje kateremu so namenjeni)
- Collaborative portals (omogočajo skupinam uporabnikov, da vzpostavijo virtualna delovna okolja za projekte in virtualne skupnosti vključno z orodji za podporo sodelovanju)
- Expertise portals (povezujejo med seboj ljudi na podlagi njihovega specifičnega znanja na določenem področju)
- Knowledge portals (počnejo vse to, kar počnejo prejšnji trije in še „nekaj več“ - nekaj česar se ne da natančno definirati)

Murray torej v portalu ne vidi le dostopne točke do različnih vsebin, temveč aplikacijo, ki ponuja celovito podporo določeni vlogi zaposlenca v podjetju. Klasičen poslovni portal vidi le kot osnovo, ki služi za nadaljnji razvoj portala. Portal naj bi omogočal ne le povezovanje aplikativnih sistemov, temveč tudi povezovanje ljudi.

Nekoliko preprostejšo definicijo pa najdemo v knjigi Kevina Roebuck-a „Enterprise Portal: High Impact Strategies“ [2]:

„Poslovni portal (ang. enterprise portal) je ogrodje, ki omogoča povezovanje in integracijo informacij, ljudi in procesov v organizaciji sami in zunaj nje. Zagotavlja varno enotno dostopno točko, pogosto v obliki spletnega uporabniškega vmesnika in je zasnovan za agregacijo in personalizacijo informacij na podlagi portletov specifičnih za posamezno aplikacijo. Posebna lastnost poslovnih portalov je decentralizirano urejanje in obvladovanje vsebine, kar omogoči, da so informacije vedno sveže.“

Ker gre pri poslovnih portalih za področje, kjer ima več podjetij svoj interes, se pogosto zgodi, da določeno podjetje predstavi svojo definicijo poslovnega portala. Pri tem se lahko opira na širše sprejete definicije, ki jim doda določene funkcije, ki jih prvotna definicija ne vsebuje. Pogosto tudi različni avtorji dajo različen poudarek prej omenjenim funkcionalnostim in portal poimenujejo z nekim novim izrazom (v angleščini se za poslovne portale tako uporablja več različnih izrazov npr. business portal, corporate portal, enterprise portal). Če podjetje uspe prepričati dovolj širok krog strokovnjakov, da je njihova definicija pravilna, pridobi določeno prednost, saj lahko reče, da nek drug ponudnik ni pravi ponudnik portala, ker mu manjkajo določene specifične funkcionalnosti.

Pomembno je še razlikovanje med javnimi in poslovnimi portali [2]. Javni portali (ang. public portal) praviloma vsebujejo enosmerno komunikacijo s svojimi uporabniki. Njihov cilj je pritegniti velike množice uporabnikov, ki bi se vračali na portal in ki bi predstavljali reprezentativen vzorec populacije. To je pomembno zato, ker je eden izmed glavnih ciljev prodaja izdelkov, ki jih na javnih portalih oglašujejo njihovi proizvajalci. Javni portali naj bi tako pritegnili čim širši krog uporabnikov, na katere bi potem lahko vplivali z oglasi.

Namen poslovnih portalov pa je omogočiti dostop do pomembnih informacij v določenem poslovnem kontekstu svojim uporabnikom in jim tako omogočiti konkurenčno prednost. Da bi to dosegli, poslovni portali gradijo na dvosmerni komunikaciji s svojimi uporabniki, ki jim omogoča interakcijo in ne le pasivno sprejemanje informacij.

Druga delitev pa razdeli portale na vertikalne in horizontalne, glede na njihov namen. Po definiciji naj bi horizontalni portali pokrivali čim širši nabor funkcionalnih področij, medtem ko se vertikalni portali osredotočijo le na eno funkcionalno področje. V angleščini so vertikalni portali poznani tudi pod imenom „vortals“.

### 2.1.2 Razvoj portalov v preteklosti

Prvi portali so se začeli pojavljati v sredini devetdesetih let, v večini primerov pa je šlo za javne spletne portale, ki so se razvili iz ponudnikov spletnih iskalnikov, tak primer sta npr. Yahoo! in Altavista [2]. Na enem mestu so ponujali nabor določenih funkcij kot so novice, elektronska pošta, vreme, pregled borznih indeksov in iskanje po spletu. Številna podjetja so kmalu izrazila potrebo po podobni enotni vstopni točki za dostop do internih podatkovnih skladišč in aplikacij.

Kot odgovor so ponudniki aplikacijskih sistemov na trgu ponudili prve poslovne portale, ki so ponujali orodja za hitro in enostavno implementacijo portalov v podjetjih. Sprva so ta orodja ponujali ponudniki, ki so se specializirali le za to področje (Epicentric, Plumtree Software...), kmalu pa so jim s konkurenčnimi izdelki sledili ponudniki aplikacijskih strežnikov. Primer takih ponudnikov so bili npr. BEA (kasneje je podjetje prevzel Oracle), Oracle in IBM, ki je prvo različico svojega Websphere Portala ponudil leta 2001. V tem času so se pojavili tudi prvi ponudniki portalov namenjenih neprofitnim organizacijam, primer takega portala je bil v tem času Liferay.

Naslednji pomembni mejnik, vsaj na področju portalov, ki temeljijo na programskem jeziku Java, je bilo sprejetje standarda JSR-168, ki je definiral programski vmesnik za interoperabilnost med portali in portleti, ponudniki aplikacijskih sistemov pa so začeli razvijati portlete, ki jih je bilo mogoče brez prilagajanja namestiti na različne portale. Standard je vključeval definicijo dveh faz v življenjskem ciklusu portletov (procesiranje in risanje), definiral je stanja portletov (ko portal spremeni stanje portleta, se le-ta odzove na spremembo z določeno akcijo, npr. prikazom določene vsebine), stanja okna, na katerem se portlet nahaja (to pomeni količino prostora, ki ga zaseda generirana vsebina portleta), podatkovni model portleta (definira kako lahko portlet shrani parametre za risanje, parametre seje in uporabniške nastavitve) ter format za kreiranje paketa, ki omogoča namestitvev portleta ali skupine portletov vključno s potrebnimi odvisnimi paketi na različne portale.

Druga različica standarda za portlete (JSR-286) je sledila leta 2008 in jo že podpirajo vsi pomembnejši ponudniki portalov. Zasnovana je bila z namenom

odprave določenih pomanjkljivosti standarda JSR-168, ki mu je dodala komunikacijo med portleti, podporo za dinamično generiranje virov ter podporo za AJAX in JSON.

Za razvoj portalov in portletov pa je pomembno še sprejetje standarda WSRP, katerega različica 1.0 je bila sprejeta leta 2003. WSRP prek spletnih storitev omogoča, da portal določen portlet ponudi kot storitev oddaljenim portalom. Oddaljeni portal lahko tako registrira portlet in ga prikazuje brez posebne namestitve na lokalnem strežniku. WSRP ni bil zasnovan kot nadomestilo standardov JSR-168 in JSR-286 temveč kot njuno dopolnilo. Standarda JSR-168 in JSR-286 namreč predpisujeta definicijo portleta, medtem ko WSRP predpisuje definiranje operacij portleta, ki so na voljo oddaljenim portalom.

### 2.1.3 Pregled trenutnih ponudnikov

Danes obstaja široka ponudba različnih poslovnih portalov. Skoraj vsako izmed večjih računalniških podjetij, ki razvijajo aplikativne sisteme, ponuja svojo rešitev. Poleg tega nekateri proizvajalci znotraj enega podjetja ponujajo več različnih portalov, ki so lahko namenjeni različnim uporabam ali pa so jih preprosto pridobili s prevzemi drugih podjetij. V spodnji tabeli je pregled nekaterih najbolj uveljavljenih poslovnih portalov [2].

| Proizvajalec       | Portal                           | Osnovna tehnologija | Pomembnejši podprti standardi                                                      |
|--------------------|----------------------------------|---------------------|------------------------------------------------------------------------------------|
| Liferay, Inc. [8]  | Liferay Portal                   | Java EE             | AJAX, JSR-168, JSR-286, JSR-314 (JSF 2.0), OASIS WSRP 1.0 in 2.0, JSR-170 JCR      |
| Red Hat, Inc. [10] | JBoss Enterprise Portal Platform | Java EE             | OASIS WSRP 1.0, JSR-170 JCR, JSR-286, JSR-301, JSR-329                             |
| Microsoft[12]      | Microsoft SharePoint             | ASP .NET            | Integracija v MS Office, OpenXML, WCAG 2.0                                         |
| Oracle [13]        | Oracle Webcenter Portal 11g      | Java EE             | JSR-168, JSR-286, OASIS WSRP 1.0 in 2.0, JCR 1.0, JSF,                             |
|                    | Oracle WebLogic Portal 10g       | Java EE             | WS-Security, WSRP, AJAX, JSR-168                                                   |
| SAP AG [9]         | SAP NetWeaver Enterprise Portal  | Java EE             | WS-I, WS Security, Microsoft .NET Interoperability, IBM WebSphere Interoperability |
| IBM [11]           | WebSphere Portal                 | Java EE             | JSR-168, JSR-286, WSRP 2.0                                                         |

## 2.2 Java EE portali

Ker so v ospredju zanimanja te diplomske naloge določeni portali osnovani na Java EE platformi, sledi pregled nekaterih pojmov, ki so pomembni za koncept portalov (osnovanih na Javi) in kratek opis portalov, na katerih bo

potekal poskus integracije.

### 2.2.1 Java Enterprise Edition in Java Enterprise Edition strežnik

Tehnologija Jave je sestavljena tako iz programskega jezika kot tudi platforme [7]. Programski jezik Java je visokonivojski objektno usmerjeni jezik z določeno sintakso, javina platforma pa je okolje, v katerem tečejo aplikacije napisane v programskem jeziku Java.

Za Javo obstajajo štiri različne platforme:

- Java Platform, Standard Edition (Java SE)
- Java Platform, Enterprise Edition (Java EE)
- Java Platform, Micro Edition (Java ME)
- JavaFX

Vsaka izmed platform vsebuje izvajalno okolje (ang. Java Virtual Machine) in programski vmesnik za razvoj aplikacij (ang. API - application programming interface). Izvajalno okolje Java je program za določen nabor strojne in programske opreme, ki lahko poganja javanske aplikacije. Ker vsaka platforma vsebuje izvajalno okolje in ker je nabor strojne in programske opreme, na katerega je mogoče določeno platformo namestiti, zelo širok, nam to zagotavlja, da se lahko aplikacija napisana v Javi izvaja na kateremkoli kompatibilnem sistemu z vsemi prednostmi programskega jezika Java (neodvisnost od platforme, stabilnost, enostavnost razvoja, varnost).

Izmed naštetih platform nas bosta zanimali Java SE in Java EE.

Programski vmesnik platforme Java SE zagotavlja ključne funkcionalnosti programskega jezika Java. Definira od osnovnih podatkovnih tipov in objektov do visokonivojskih razredov (ang. classes), ki se uporabljajo za potrebe omrežnega povezovanja, varnosti, dostopa do podatkovnih baz, grafičnih uporabniških vmesnikov in uporabe XML-ov.

Java EE je razvita na osnovi Java SE in zagotavlja dodatne programske vmesnike in izvajalno okolje za razvoj ter izvajanje obsežnih, skalabilnih, večnivojskih, zanesljivih in varnih omrežnih aplikacij in informacijskih sistemov, ki jih poznamo tudi pod imenom poslovne aplikacije (ang. enterprise applications), saj so namenjene reševanju problemov, s katerimi se srečujejo velika podjetja lahko pa tudi različne agencije, vlade ali pa tudi posamezni

razvijalci in majhne organizacije. Zaradi teh lastnosti je tudi primerna za razvoj poslovnih portalov in aplikacij, ki naj bi tekle na teh portalih. Vse te funkcionalnosti pa pogosto naredijo poslovne aplikacije tudi kompleksne, zato je bila Java EE razvita z namenom zmanjšanja kompleksnosti, da bi se lahko razvijalci bolj osredotočili na funkcionalnost razvijane aplikacije, za sistemske podrobnosti pa bi poskrbela Java EE.

Pomemben pojem pri poslovnih aplikacijah je večnivojskost. V večnivojskih aplikacijah je funkcionalnost aplikacije porazdeljena v izolirana področja imenovana nivoji. Danes je najbolj uveljavljena trinivojska arhitektura (prikazana na sliki 2.1), ki sestoji iz naslednjih nivojev:

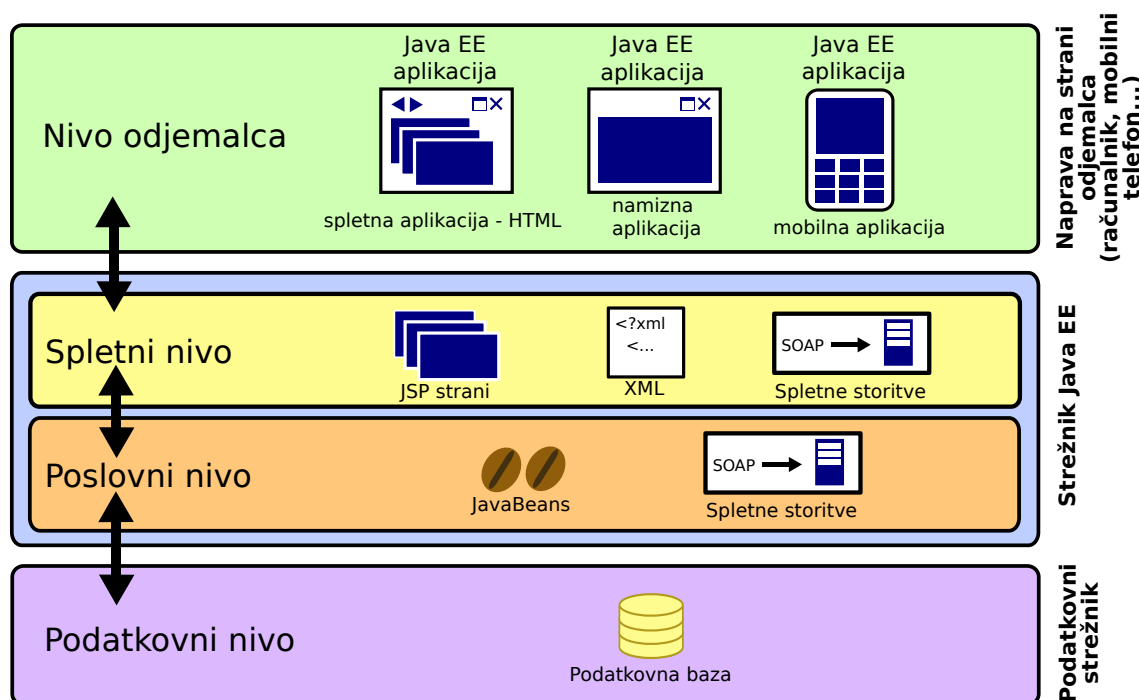
- nivo odjemalca (ang. „client tier“)
- spletni in poslovni nivo (ang. „web tier“ in „business tier“; ta nivo je znan tudi pod izrazom „middle tier“)
- podatkovni nivo (ang. „data tier“, pogosto imenovan tudi z izrazom „enterprise information systems tier“)

Nivo odjemalca predstavlja program, prek katerega odjemalec sproža zahteve (ang. requests) proti poslovnemu nivoju. V primeru poslovnih portalov je ta program največkrat kar spletni brskalnik.

Poslovni nivo sestoji iz komponent, ki skrbijo za poslovno logiko aplikacije. To pomeni, da zagotavljajo funkcionalnosti, ki pokrijejo potrebe določene poslovne domene, npr. finančnih ustanov. V pravilno načrtovani poslovni aplikaciji so ključne funkcionalnosti vsebovane v poslovnem nivoju. Primeri tehnologij, ki se uporabljajo na poslovnem nivoju v primeru Java EE (podroben opis tehnologij, ki so ključne za predmet diplome, je vključen v naslednjih poglavjih) so:

- Enterprise JavaBeans components
- JAX-RS RESTful web services
- JAX-WS web service endpoints
- Java Persistence API entities

Spletni nivo je sestavljen iz komponent, ki procesirajo interakcijo med odjemalcem in poslovnim nivojem. Skrbi za dinamično generiranje vsebine v različnih formatih za odjemalca, zbira vnose uporabnikov odjemalskega vmesnika in jih v ustrezni obliki posreduje poslovnemu nivoju, kontrolira tok oken ali strani



Slika 2.1: Grafični prikaz trinivojske arhitekture z nekaterimi tehnologijami Java EE, ki se uporabljajo na posameznem nivoju

pri odjemalcu, vzdržuje stanje uporabniške seje in izvaja nekaj osnovne logike ter hrani začasne podatke.

Primeri nekaterih Java EE tehnologij, ki se uporabljajo v spletnem nivoju:

- Servlets (Java razredi, ki dinamično procesirajo zahteve in generirajo odgovore, običajno v obliki za spletne strani HTML)
- JavaServer Faces (ogrodje za delo z uporabniškimi vmesniki v obliki komponent, npr. gumbov, poleg tega omogoča validacijo in pretvorbo vnesenih podatkov in vzdrževanje stanja komponent)
- JavaServer Pages (tekstovni dokumenti, ki so preoblikovani v servlete in definirajo, kako je lahko dinamična vsebina dodana statičnim stranem, npr. na straneh HTML)

Podatkovni nivo je sestavljen iz podatkovnih strežnikov, lahko pa tudi iz ERP sistemov in mainframe-ov. Običajno se ti sistemi nahajajo na ločenem strežniku, do njih pa dostopajo komponente iz poslovnega nivoja, ki iz njih

berejo ali pa vanje zapisujejo podatke. V platformi Java EE se za dostop do podatkovnega nivoja uporabljajo naslednje tehnologije (podrobneje so obrazložene v naslednjih poglavjih):

- Java Database Connectivity API (JDBC)
- Java Persistence API (JPA)
- Java EE Connector Architecture
- Java Transaction API (JTA)

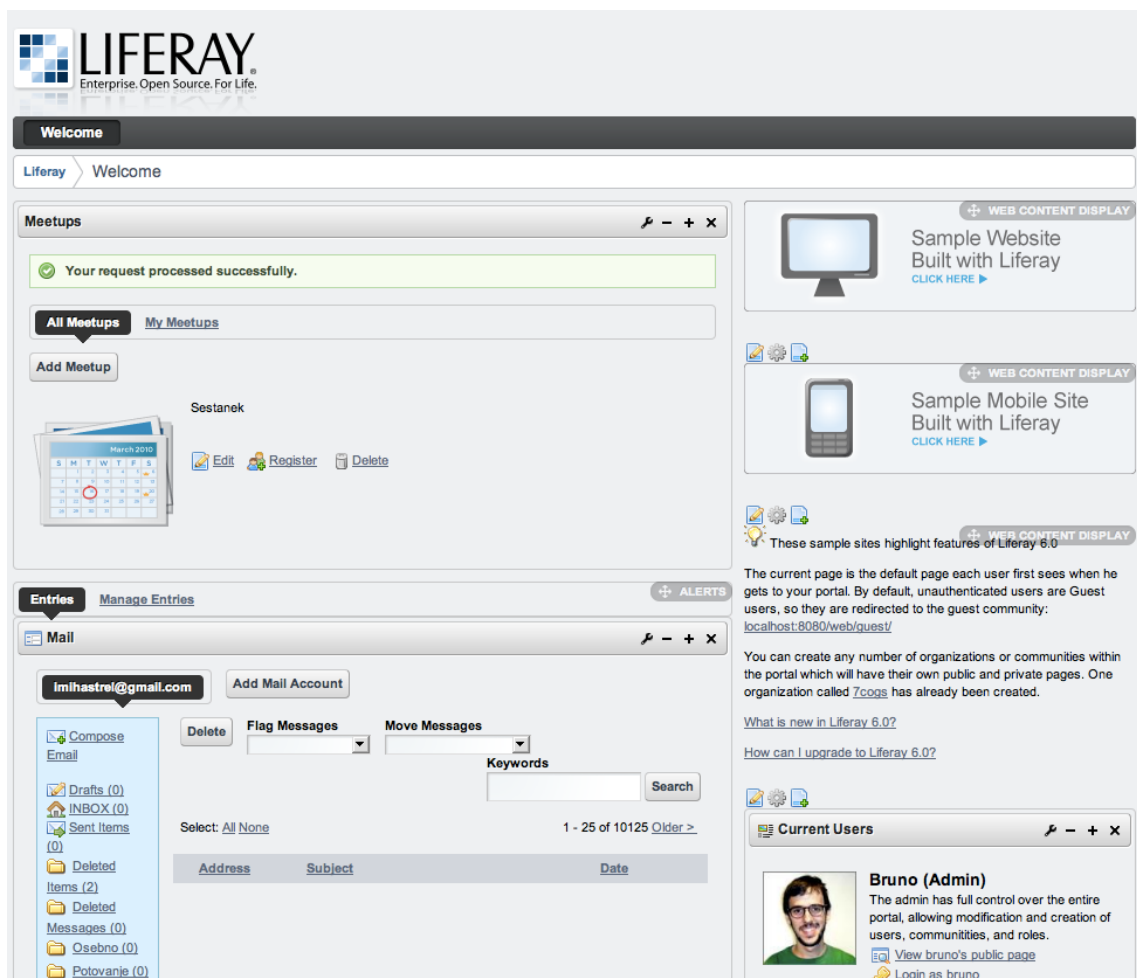
Na osnovi teh definicij lahko predstavimo definicijo strežnika Java EE:

Strežnik Java EE je strežniška aplikacija, ki implementira programske vmesnike Java EE in zagotavlja standardne storitve Java EE platforme. Java EE strežniki so pogosto poimenovani z izrazom „aplikacijski strežnik“, saj omogočajo strežbo podatkov iz aplikacije različnim odjemalcem, podobno kot spletni strežniki strežejo spletne strani spletnim brskalnikom [7].

Portali, ki jih bomo obravnavali v tej diplomski nalogi, pa so pravzaprav aplikacije razvite na osnovi Java EE, ki se izvajajo na Java EE strežnikih in nudijo funkcionalnosti portalov, kot smo jih definirali v prejšnjih poglavjih.

### 2.2.2 Portlet

Portlet je programska komponenta, ki omogoča vključitev in prikaz uporabniškega vmesnika na portalu [2]. Portleti generirajo dele označevalne kode (npr. HTML, XHTML ipd.) imenovane fragmenti, ki jih portal nato združi ali agregira. V primeru, da je uporabniški vmesnik portala zasnovan v obliki spletne aplikacije, je stran na portalu oblikovana kot skupek portletov, ki se med seboj ne prekrivajo. Odjemalci in portleti v tem primeru komunicirajo v obliki zahtevkov-odgovor. To pomeni, da odjemalec z akcijami na vsebini portleta sproža zahteve, ki jih prejme portal in jih nato posreduje portletu, katerega je cilj odjemalec. Portlet nato obdela zahtevo in kot odgovor vrne nov fragment. Vsebina, ki jo portlet generira, je lahko odvisna od uporabnika oz. vloge, ki jo ima uporabnik v sistemu. Vire potrebne za izvajanje portleta zagotavlja t.i. vsebovalnik portleta (ang. portlet container). Primeri portletov so lahko npr. elektronska pošta, borzni indeksi, različne statistike... Slika 2.2 prikazuje primer portletov na portalu Liferay.



Slika 2.2: Primer strani portala Liferay, ki na levi strani prikazuje portleta za pregled sestankov in elektronske pošte

### 2.2.2.1 Vsebovalnik portleta

Za upravljanje portletov skrbi vsebovalnik portleta (ang. portlet container). Ta portletom zagotavlja potrebno izvajalno okolje in skrbi za njihov življenjski cikel, poleg tega pa skrbi tudi za shranjevanje nastavitev portleta. Vsebovalnik portleta od portala prejme zahteve, ki se nanašajo na portlete, ki so v vsebovalniku [3]. Tipično zaporedje dogodkov poteka v naslednjem vrstnem redu:

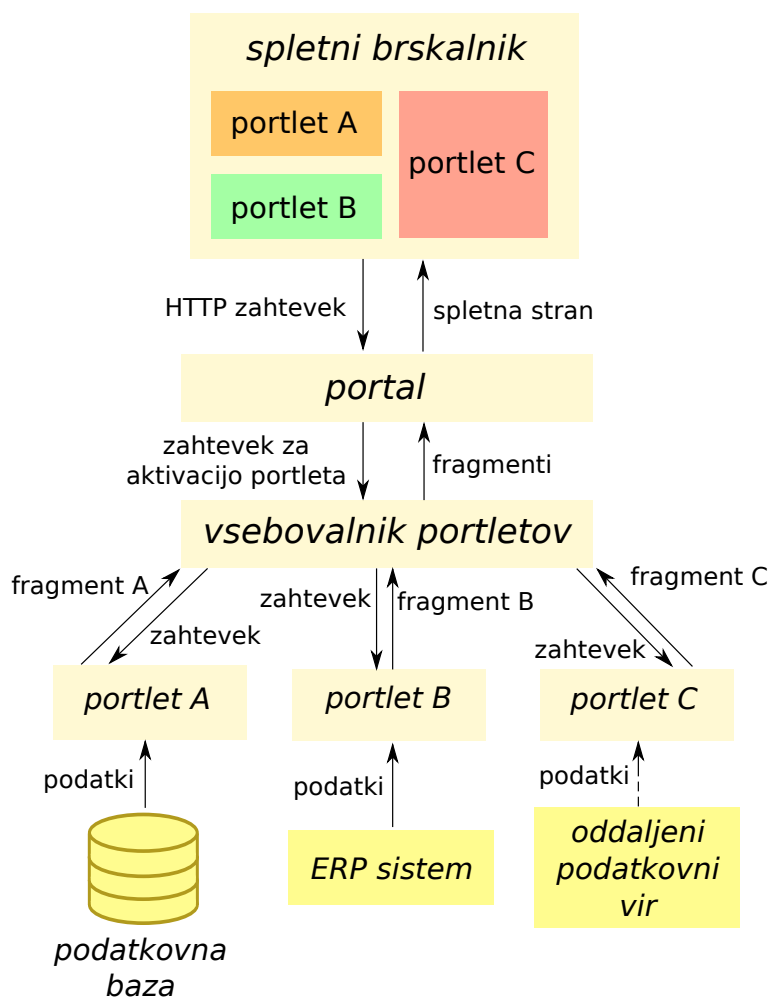
- odjemalec (po avtentikaciji) sproži HTTP zahtevo na portal
- portal prejme zahtevek
- portal preveri ali zahtevek vsebuje določeno akcijo, ki se nanaša na portlete (npr. akcijo za procesiranje ali akcijo za risanje portleta)
- če zahtevek vsebuje akcijo za portlet, portal vsebovalniku portletov pošlje zahtevek za aktiviranje portleta, na katerega se akcija nanaša
- portal aktivira portlet prek vsebovalnika portletov
- portal agregira fragmente, ki jih kot odgovor generirajo portleti, v stran na portalu in jo pošlje odjemalcu

Agregacija je tipično naloga portala, ne pa vsebovalnika portala. Slika 2.3 prikazuje primer agregacije.

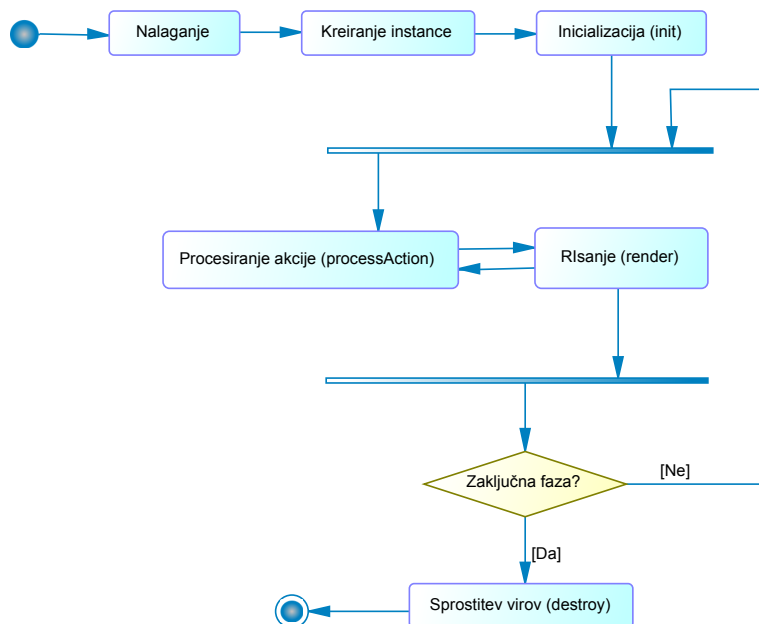
### 2.2.2.2 Življenjski cikel portleta

Vsebovalnik portleta upravlja s portletom v procesu, ki ga imenujemo življenjski cikel portleta [3]. Proces vključuje faze nalaganja portleta, kreiranja njegove instance, inicializacije, procesiranja zahtevkov in njegovega uničenja ob koncu življenjskega cikla. Na platformi Java EE se to kaže v metodah, ki jih definira vmesnik (ang. interface) Portlet:

- `init(PortletConfig);`
- `processAction(ActionRequest, ActionResponse);`
- `render(RenderRequest, RenderResponse);`
- `destroy();`



Slika 2.3: Enostaven grafičen prikaz agregacije fragmentov, ki jih generirajo portleti A, B in C, v stran na portalu



Slika 2.4: Grafični prikaz življenjskega cikla portleta [3]

Slika 2.4 prikazuje življenjski cikel portleta.

V fazi nalaganja in kreiranja instance vsebovalnik portletov kreira instanco portleta. To se lahko zgodi ob zagonu aplikacije portleta ali pa se zgodi z zamikom šele takrat, ko se vsebovalnik portletov prepriča, da je določen portlet dejansko potreben za procesiranje nekega zahtevka.

V fazi inicializacije (klic metode „init()“) mora vsebovalnik portletov inicializirati portlet, preden od njega zahteva procesiranje zahtevkov. V tej fazi portlet poskrbi za inicializacijo časovno ali drugače zahtevnih opravil kot so npr. inicializacije povezav do virov, poleg tega pa izvede akcije, ki jih je potrebno v življenjskem ciklu portleta izvesti le enkrat (t.i. „one-time activities“). Predpogoj za fazo inicializacije je ustrezno instanciran portlet.

V fazi procesiranja zahtevkov lahko vsebovalnik portletov od določenega portleta zahteva procesiranje zahtevkov odjemalcev. V tej fazi se lahko kličeta metodi „processAction(ActionRequest, ActionResponse)“ ali „render(RenderRequest, RenderResponse)“.

Če je odjemalec poslal zahtevek, ki vsebuje akcijo (ang. ActionRequest), portlet izvede zahtevano akcijo, v kateri spremeni stanje portleta glede na parametre, ki so vsebovani v zahtevku. Vsebovalnik portletov nato avtomatsko kliče še metodo „render“, da portlet zgenerira nov fragment, ki odraža spre-

membo vsebine. V primeru da vsebovalnik portleta uporablja predpomnenje portletov, zbere še fragmente preostalih portletov in jih posreduje portalu, ki jih združi v portalno stran. Če pa vsebovalnik ne uporablja predpomnenja, mora poklicati metodo „render“ na vsakem izmed portletov na portalni strani. Tak primer prikazuje slika 2.5.

Odjemalec lahko pošlje tudi zahtevek za risanje (ang. rendering) portleta. Ko vsebovalnik portletov posreduje zahtevek določenemu portletu, ta tipično kreira fragment na podlagi trenutnega stanja. V fazi risanja portleta se samo stanje ne more spreminjati, to se lahko zgodi le v fazi procesiranja zahtevane akcije.

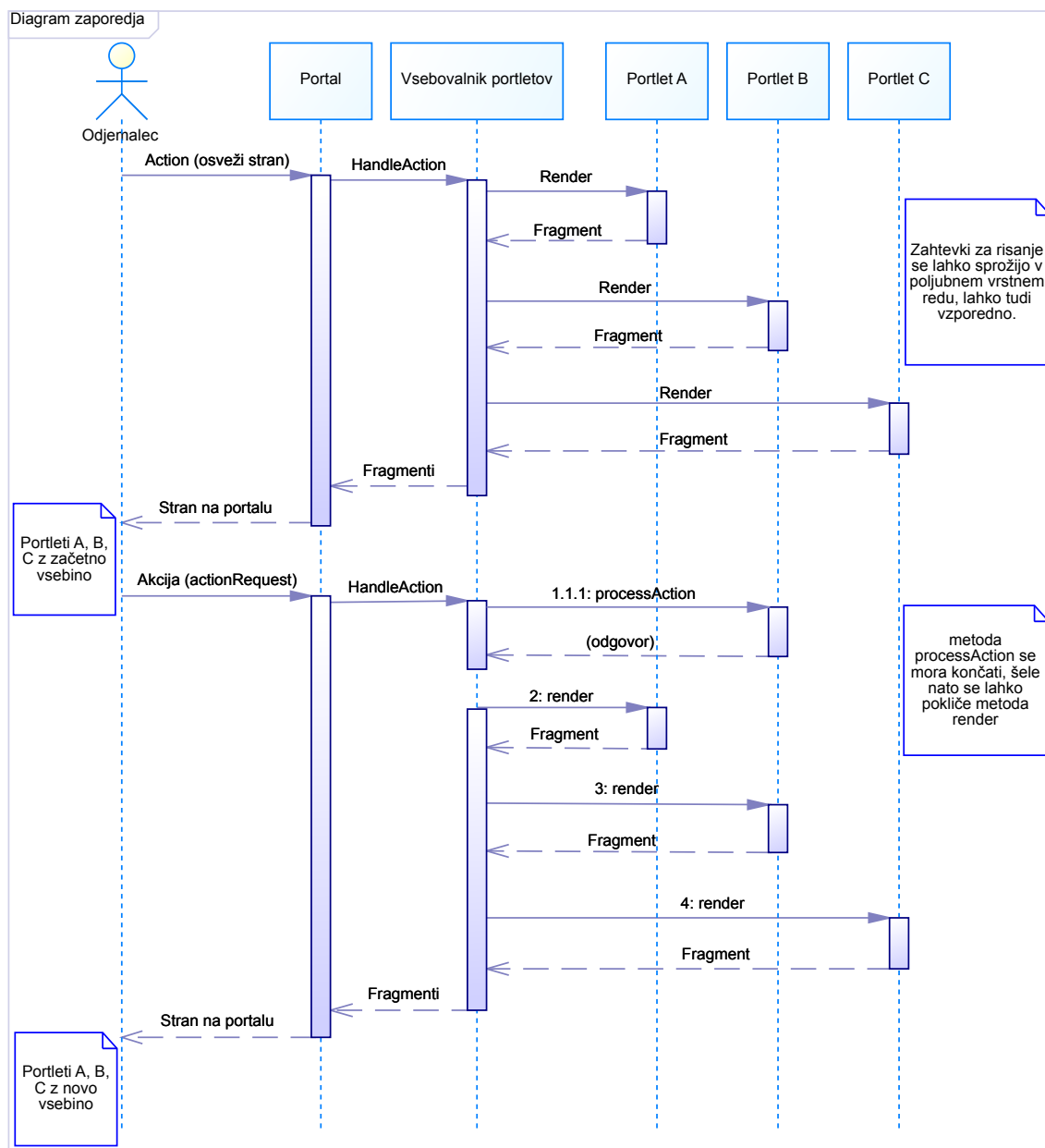
Ker lahko uporabnik naenkrat klikne le na en portlet (torej sproži le eno akcijo), se naenkrat lahko izvaja le ena metoda „processAction“ na uporabnika, kar pa še ne pomeni nujno, da se bo spremenila le vsebina enega portleta. Če imamo npr. na strani portlet, v katerem izberemo določen nogometni klub in dva portleta od katerih eden prikazuje seznam igralcev in drugi statistiko za izbrani klub, bo izbira določenega kluba povzročila spremembo vsebine na vseh portletih. Skrb za to, da se vsi portleti zavedajo spremembe, je v rokah programerja, ki mora poskrbeti, da v fazi „processAction“ ustrezno spremeni potrebne parametre.

V zaključni fazi, ko vsebovalnik portletov presodi, da določen portlet ni več potreben, se kliče metoda „destroy()“, ki poskrbi za sprostitev vseh virov, ki jih je zasedal portlet (to vsebuje sprostitev pomnilnika, referenc na datoteke in čiščenje programskih niti).

### 2.2.2.3 Standard JSR-168 (Portlet 1.0)

Ko so na tržišče poslovnih portalov vstopili prvi ponudniki, je vsak izmed njih ponudil svoj programski vmesnik za izdelavo portletov, kar je imelo za posledico odvisnost od le enega ponudnika (t.i. „vendor lock-in“) [14]. To se je končalo z uvedbo standarda Java Specification Request (JSR) 168, ki je z natančno specifikacijo portletov (vključno z definicijo povezave med portletom in vsebovalnikom portleta) omogočil prenosljivost med posameznimi portali. Standard je poznan tudi pod imenom „Java Portlet Specification 1.0“, sprejet pa je bil v oktobru leta 2003. Portleti, ki so zasnovani na tem standardu so poznani pod imenom „Portlet 1.0“.

Standard JSR-168 je sprejela organizacija Java Community Process (JCP) pod vodstvom podjetij Sun in IBM, pri njegovem nastajanju pa je sodelovala strokovna skupina sestavljena iz širokega nabora ponudnikov portalov in drugih podjetij, med njimi Apache Software Foundation, BEA, Boeing, Borland,



Slika 2.5: Diagram zaporedja za primer osvežitve strani (zgornja polovica) in določeno akcijo (spodnja polovica) [3][6]

Citrix Systems, Fujitsu, Hitachi, IBM, Novell, Oracle, SAP, SAS Institute, Sun, Sybase, Tipco in Vignette [6]. Sama kratica JSR pomeni „Java Specification request“, kar je oznaka za formalne dokumente, ki opisujejo specifikacije in tehnologije javine platforme. Preden je določen JSR sprejet s strani organizacije JCP, je podan v javno formalno strokovno presojo. Ko dokument dobi končno obliko, o njem glasujejo člani vrhovnega odbora JCP. Končna oblika JSR poleg samega dokumenta vsebuje še referenčno implementacijo v obliki prosto dostopne programske kode. V primeru standarda JSR-168 je bila referenčna implementacija narejena v okviru organizacije Apache Software Foundation, poimenovana pa je bila Pluto.

Standard je definiral portlet, vsebovalnik portletov in življenjski cikel portleta, kot so bili predstavljeni v prejšnjih poglavjih. Poleg tega je definiral še nekatere druge funkcionalnosti.

Ena izmed funkcionalnosti so načini delovanja portletov (ang. „portlet modes“). Način delovanja definira funkcijo, ki jo portlet trenutno opravlja. Portleteti namreč opravljajo več različnih opravil in ustrezno generirajo fragmenete glede na funkcijo, ki jo v tistem trenutku izvajajo. Ko vsebovalnik portleta naloži in inicializira portlet, mu poda tudi način delovanja portleta. Portlet nato lahko (ali pa tudi ne) upošteva podani način in kreira ustrezno vsebino. Standard definira 3 različne načine delovanja [15]:

- VIEW je klasičen način delovanja, v katerem se od portleta pričakuje, da generira vsebino glede na trenutno stanje portleta (npr. uporabniški vmesnik, ki lahko služi za interakcijo z uporabnikom ali pa neko statično vsebino). Podpora temu načinu delovanja je obvezna.
- EDIT je način delovanja, v katerem je uporabniku omogočeno, da prilagodi obnašanje portleta, npr. v obliki uporabniškega vmesnika, v katerem lahko uporabnik vnese željene nastavitve. Podpora načinu delovanja EDIT ni obvezna.
- HELP je način delovanja, ki uporabniku ponudi pomoč pri uporabi portleta. Implementiran je lahko kot preprost prikaz pomoči o celotnem portletu na eni strani ali pa prikaže pomoč odvisno od konteksta. Podpora temu načinu delovanja ni obvezna.

Standard proizvajalcem portalov omogoča definicijo dodatnih načinov delovanja, pri čimer velja, da portlet lahko uporablja le načine delovanja, ki jih podpira portal (torej portlet sam ne more definirati novega načina delovanja). Portleteti lahko programsko spremenijo način delovanja med procesiranjem akcije (action request).

Druga funkcionalnost, ki jo definira standard je stanje okna. Stanje okna je indikator, ki portletu pove, kolikšen del okna na strani (npr. v spletnem brskalniku) portala bo dodeljen vsebini, ki jo bo ta portlet zgeneriral. Na podlagi tega podatka lahko portlet prilagodi svoj izpis, da je tako primeren za optimalen prikaz na strani uporabnika. Standard definira naslednja stanja okna [15]:

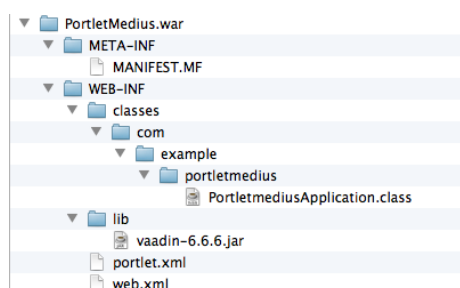
- NORMAL je stanje okna, ki portletu pove, da je možno, da si okno deli z drugimi portleti ali pa ima ciljna naprava omejene zmožnosti prikaza. Od portleta se v tem stanju pričakuje, da omeji velikost kreiranega fragmenta.
- MAXIMIZED je stanje okna, v katerem portlet zaseda celotno stran ali pa mu je v primerjavi z ostalimi portleti dodeljeno več prostora. V tem načinu lahko portlet generira fragment, ki zaseda več prostora.
- MINIMIZED je stanje okna, ki od portleta zahteva, da ta kreira minimalen fragment ali pa celo fragment, ki ne vsebuje nobene vsebine.

Podobno kot pri načinih delovanja, lahko tudi pri stanjih okna proizvajalci definirajo dodatna stanja.

Standard poleg tega določa še strukturo map in datotek, ki so potrebne pri namestitvi in delovanju portleta. Pri tem velja, da je struktura enaka kot pri spletnih aplikacijah, dodatno pa mora imeti še datoteko `/WEB-INF/portlet.xml`, v kateri so zapisane nastavitve portleta. Vsi programski razredi portleta vključno z vsemi razredi, za katere v aplikaciji portleta obstajajo reference, se morajo nahajati v mapi `/WEB-INF/classes` ali pa morajo biti stisnjeni v „Java Archive“ (JAR - to je format za stisnjene datoteke) datoteki, ki se mora nahajati v mapi `/WEB-INF/lib`. Celotno strukturo map in datotek je nato potrebno stisniti v datoteko tipa „WAR“ (Web Archive - format stisnjene datoteke, ki se uporablja za distribucijo spletnih aplikacij), tako pripravljena datoteka pa lahko nato služi za namestitev portleta na portal. Primer strukture map in datotek je na sliki 2.6.

Standard tudi natančno predpisuje strukturo datoteke `portlet.xml`, ki vsebuje nastavitve potrebne za delovanje portleta. Vsaka datoteka `portlet.xml` se začne z začetno deklaracijo xml datotek (značka „XML“), kateri sledi značka za definicijo aplikacije portleta („`portlet-app ...`“). Pomemben element v tej deklaraciji je „`version`“, ki definira različico standarda, za JSR-168 je potrebno ta atribut nastaviti na 1.0.

Začetni deklaraciji sledi deklaracija samega portleta (značka „`portlet`“). V njej portletu najprej določimo osnovne parametre:



Slika 2.6: Primer strukture razširjene WAR datoteke

- opis (značka „description“),
- oznako (značka „portlet-name“)
- oznako za prikaz (značka „display-name“)
- programski razred, ki se kliče za zagon portleta (značka „portlet-class“).
- čas predpomnenja (značka „expiration-cache“)

Oznaka in programski razred sta ključna za kreiranje referenc in inicializacijo portleta na portalu, medtem ko sta oznaka za prikaz in opis namenjena boljšemu pregledu uporabnika. Čas predpomnenja je število sekund, ki določa koliko časa bo portlet v predpomnilniku (število -1 pomeni, da je portlet vedno v predpomnilniku, število 0 pa da ni nikoli) [15].

Osnovnim parametrom sledi deklaracija načinov delovanja, ki jih portlet podpira („portlet-mode“) in podprte načine prikaza (npr. HTML, značka „mime-type“).

Deklaraciji načina delovanja sledi deklaracija podprtih jezikov (značka „supported-locale“) in deklaracija metapodatkov o portletu (značka „portlet-info“):

- značka „title“ (naslov, ki ga bo portal prikazal na portletu)
- značka „short-title“ (krajša oblika naslova, ki se lahko uporabi za reference na portlet)
- značka „keywords“ (ključne besede, ki spadajo v vsebinski kontekst portleta)

Deklaraciji metapodatkov sledi še deklaracija nastavitev portleta, ki so specifični za določenega uporabnika (značka „portlet-preferences“) ter nastavitve za varnost (značka „security-role-ref“).

Vse značke morajo seveda biti ustrezno zaključene. Slika 2.7 prikazuje primer datoteke „portlet.xml“.

Specifikacija JSR-168 definira še več tehničnih podrobnosti, ki pa za razumevanje niso bistvene. Celotna specifikacija je prosto dostopna na spletu [16].

#### 2.2.2.4 Standard JSR-286 (Portlet 2.0)

Standard JSR-286 je nastal pod vodstvom podjetja IBM in bil sprejet s strani organizacije JCP leta 2008. Kot pri standardu JSR-168 je tudi v tem primeru pri nastajanju sodelovala široka skupina zainteresiranih podjetij in organizacij, med njimi Adobe Systems Inc., Oracle, SAP AG, Sun Microsystems, Vignette, BEA Systems, Liferay Inc., Novell Inc., RedHat, TIBCO idr. [18]

Glavna pomanjkljivost standarda JSR-168 je bila, da je definirala portlete kot komponente, ki jih portal agregira v eno stran, ni pa ponudila možnosti, da bi iz teh komponent lahko izdelali integrirano in povezano aplikacijo. To pomanjkljivost naj bi odpravil standard JSR-286, ki je v ta namen definiral dva načina koordinacije med portleti:

- dogodke (ang. „events“), ki omogočijo podporo obvestilom med portleti, ki se lahko nanje ustrezno odzovejo
- deljene parametre (ang. „public render parameters“), ki omogočijo deljenje določenih parametrov med portleti

Slika 2.8 prikazuje koordinacijo prek dogodkov. Da se lahko portleti koordinirajo preko dogodkov, mora imeti portlet A ob namestitvi na portal v svoji konfiguracijski datoteki portlet.xml definirano, da zna generirati dogodek A (izsek datoteke portlet.xml, ki definira dogodek prikazuje slika 2.9). Portleta B in C pa morata imeti v svoji datoteki portlets.xml definirano, da znata sprejeti dogodek A. Administrator nato ob nameščanju ustvari povezavo med portletom A (virom dogodkov) ter portletoma B in C (ponorom dogodkov). Ko uporabnik sproži neko akcijo na portletu A, lahko le-ta generira dogodek, ki ga doda posredovalniku dogodkov (ang. „event broker“). Ta poskrbi, da ob zaključku procesiranja akcije portleta A, ustrezen dogodek A pošlje portletoma B in C. To naredi tako, da na portletih B in C kliče metodo processEvent, kot parameter pa poda dogodek A.

Deljeni parametri risanja omogočajo dostop do istih parametrov iz različnih portletov ali drugih komponent. Podobno kot pri dogodkih se v datoteki portlet.xml definira določen parameter z enolično oznako, do katerega lahko potem

```

<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet"
  version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://java.sun.com/xml/ns/portlet-app_1_0.xsd">

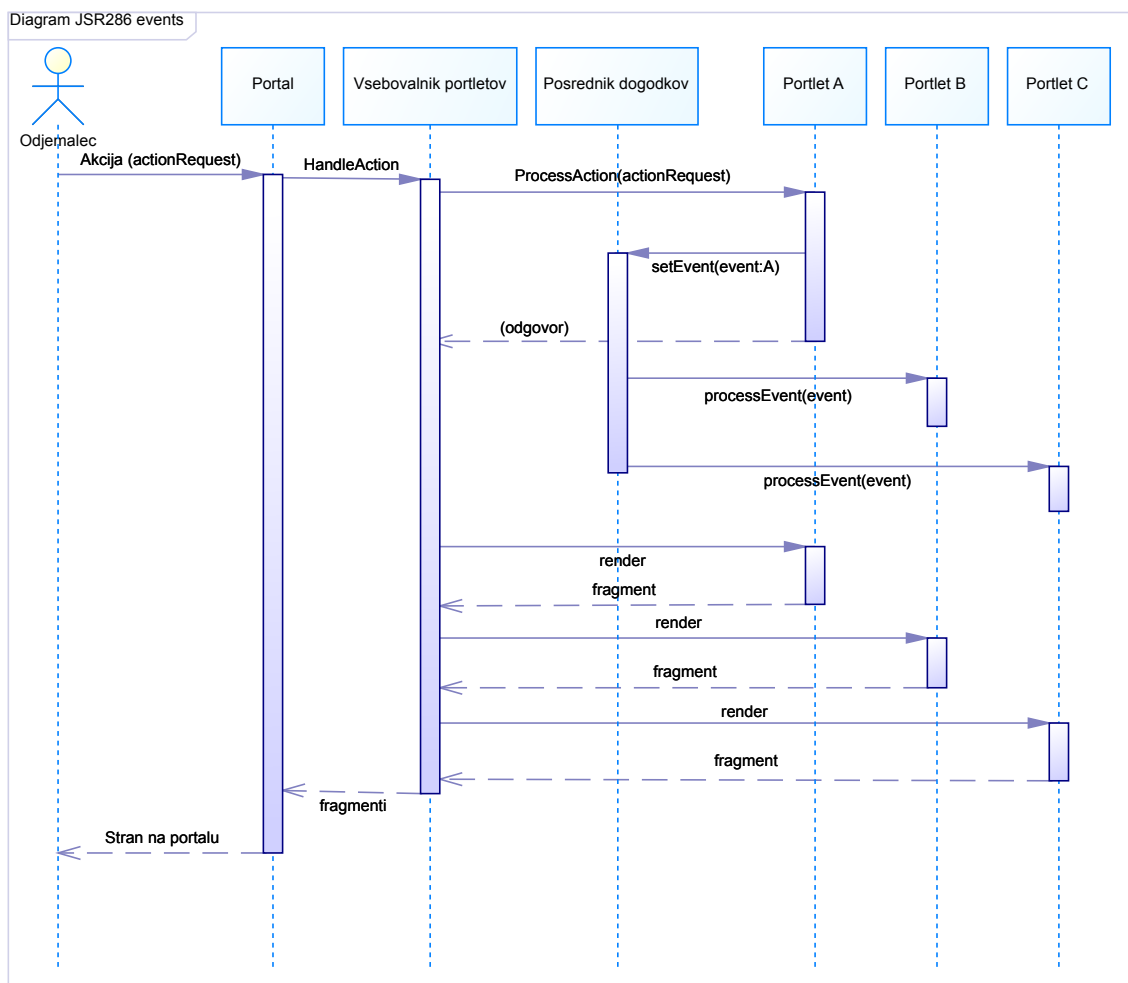
  <portlet>
    <description>Example of creating a portlet</description>
    <portlet-name>ExamplePortlet</portlet-name>
    <display-name>Example Portlet</display-name>
    <portlet-class>org.opensourceportals.samples.ExamplePortlet</portlet-class>
    <expiration-cache>-1</expiration-cache>
    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>view</portlet-mode>
      <portlet-mode>edit</portlet-mode>
      <portlet-mode>help</portlet-mode>
    </supports>
    <supports>
      <mime-type>text/xml</mime-type>
      <portlet-mode>view</portlet-mode>
    </supports>
    <supported-locale>EN</supported-locale>

    <portlet-info>
      <title>Pre-configured title</title>
      <short-title>Example</short-title>
      <keywords>JSR 168, Portlet API, Example, Simple</keywords>
    </portlet-info>

    <portlet-preferences>
      <preference>
        <name>index-location</name>
        <value>/opt/lucene/index</value>
        <read-only>true</read-only>
      </preference>
      <preference>
        <name>sites-to-crawl</name>
        <value>http://jakarta.apache.org</value>
        <value>http://java.sun.com</value>
        <value>http://onjava.com</value>
      </preference>
      <preference>
        <name>crawl-depth</name>
        <value>2</value>
      </preference>
      <preferences-validator>
        com.opensourceportals.validator.CrawlValidator
      </preferences-validator>
    </portlet-preferences>
    <security-role-ref>
      <role-name>ADMIN</role-name>
      <role-link>administrator</role-link>
    </security-role-ref>
  </portlet>
</portlet-app>

```

Slika 2.7: Primer datoteke „portlet.xml“



Slika 2.8: Diagram zaporedja pri pošiljanju dogodka med portleti

```

<event-definition>
  <qname xmlns:x="http://example.com/events">x:foo.bar</qname>
  <value-type>com.example.Address</value-type>
</event-definition>

<portlet>
  <supported-publishing-event>
    <qname xmlns:x="http://example.com/events">x:foo.bar</qname>
  </supported-publishing-event>
</portlet>
  
```

Slika 2.9: Definicija dogodka v datoteki portlet.xml [26]

dostopa portlet. Na te parametre lahko gledamo kot na neko deljeno spremenljivko, katero lahko berejo ali pa vanjo pišejo različni portleti.

JSR-286 je definirala še nekatere druge podrobnosti, ki pa niso bistvene za razumevanje diplomske naloge. Tako kot za JSR-168, so tudi specifikacije za standard JSR-286 dostopne na spletu [26].

#### 2.2.2.5 Standard WSRP

Standard WSRP (ang. „Web Services for Remote Portlets“) je sprejela organizacija OASIS (Organization for the Advancement of Structured Information Standards) in jo podpira večina pomembnejših proizvajalcev poslovnih portalov. Definira natančno določen vmesnik za komunikacijo s spletnimi storitvami, ki so usmerjene h končnim uporabnikom (v nasprotju s tradicionalnimi spletnimi storitvami, ki so namenjene komunikaciji na programskem nivoju). Usmerjenost h končnim uporabnikom se kaže v tem, da te spletne storitve svojim uporabnikom kreirajo uporabniški vmesnik.

Podobno kot tradicionalne spletne storitve omogočajo uporabo storitev določenega back-end sistema neodvisno od tega, kako je implementiran odjemalec, lahko spletne storitve, ki jih definira standard WSRP, definiramo kot storitve, ki omogočajo uporabo ne le uporabo back-end sistema, temveč tudi celotnega uporabniškega vmesnika [27]. Če bi npr. na portal, ki ga v podjetju uporablja nek finančni analitik, radi dodali možnost pregleda vrednosti delnic podjetja, lahko v primeru uporabe portletov, ki so zasnovani po specifikaciji WSRP, preprosto registriramo obstoječ portlet nekega ponudnika in ga dodamo na stran na portalu. Odpade torej programiranje logike za klic tradicionalne spletne storitve in obdelava prejetih podatkov ter njihov prikaz na posebej sprogramiranemu uporabniškemu vmesniku, saj za vse to poskrbi oddaljen ponudnik portleta. Le-ta generira fragment, ki ga pošlje portalu, ki se v tem primeru obnaša kot klient. Portal nato agregira prejete fragmente v stran na portalu.

Da bi to dosegli, je standard WSRP definirala tri akterje [27]:

- ponudnik WSRP (ang. WSRP producer): je spletna storitev, ki ponuja enega ali več portletov in implementira nabor WSRP vmesnikov. Taka spletna storitev je podobno kot klasične spletne storitve dosegljiv s pomočjo standardnega WSDL dokumenta. Ti WSDL dokumenti se lahko npr. nahajajo v registrih spletnih storitev UDDI (Universal Data Discovery Interface).
- WSRP portlet: je komponenta z uporabniškim vmesnikom, ki se nahaja

v ponudniku WSRP in do katere je mogoč oddaljen dostop preko vmesnikov, ki jih definira ponudnik. WSRP portlet sam ni spletna storitev, saj do njega ni mogoče neposredno dostopati, temveč le preko ponudnika WSRP.

- odjemalec WSRP (ang. WSRP consumer): je odjemalec spletne storitve, ki uporablja ponujene spletne storitve ponudnika WSRP in zagotavlja okolje, prek katerega lahko uporabnik dostopa do portletov, ki jih ponujajo ti ponudniki. Najpogosteje se v vlogi WSRP odjemalca nahaja portal.

Ko uporabnik najde določen oddaljen portlet, ki ga želi na svojem portalu, lahko s pomočjo WSDL datoteke na portalu namesti ta portlet (dejansko se ne namesti sam portlet, temveč se ustvari neke vrste referenca na spletno storitev, ki ponuja ta portlet). Ko nato sproži zahtevo po spletni strani na portalu, ki vsebuje ta portlet, portal pokliče oddaljenega ponudnika preko spletnih storitev in mu posreduje zahtevo. Oddaljeni ponudnik nato z ustreznimi parametri sproži metode na ciljanem portletu, ki generira nov fragment. Ta fragment oddaljeni ponudnik kot odgovor pošlje nazaj portalu, ki ga skupaj z drugimi fragmenti agregira v spletno stran, ki jo prikaže v spletnem prskalniku uporabnika.

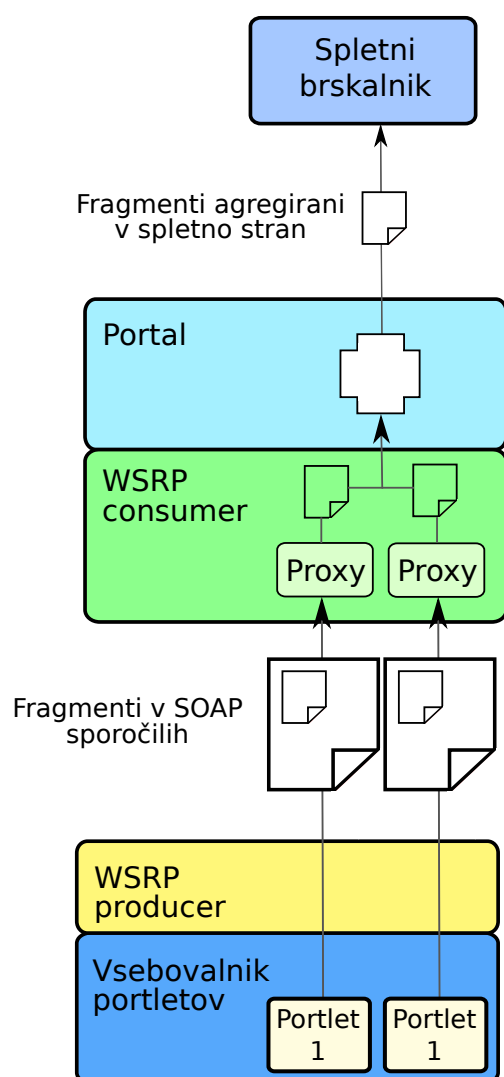
Slika 2.10 prikazuje potek generiranja spletne strani na portalu, ki uporablja oddaljene portlete implementirane po standardu WSRP. Na sliki je prikazano, da portal uporablja le enega WSRP ponudnika, kar pa ni nujno res. Portal namreč lahko naenkrat uporablja več oddaljenih ponudnikov, poleg tega pa lahko na isto spletno stran agregira še vsebino fragmentov, ki jih generirajo lokalno nameščeni portleti.

## 2.2.3 Primeri Java EE portalov

Sledi kratka splošna predstavitev dveh poslovnih Java EE portalov, ki sta bila dva v diplomskem delu uporabljena kot osnova za integracijo. Podrobne tehnične podrobnosti vsakega izmed njih pa so predstavljene v nadaljnjih poglavjih.

### 2.2.3.1 Liferay Portal®

Liferay portal je osnovan na Javi EE in se ga lahko namesti v poljubnem operacijskem sistemu, ki je sposoben poganjati izvajalno okolje Java. Lahko se uporablja v kombinaciji z različnimi aplikacijskimi strežniki (med podprtimi so



Slika 2.10: Agregacija fragmentov iz portletov implementiranih po standardu WSRP [27]

npr. JBoss<sup>®</sup>, Weblogic<sup>®</sup>, Websphere<sup>®</sup>, OracleAS<sup>®</sup> in GlassFish<sup>®</sup>) in podatkovnimi bazami (podprte so IBM DB2<sup>®</sup>, MySQL<sup>®</sup>, Oracle<sup>®</sup>, PostgreSQL<sup>®</sup> in nekatere druge) [8]. Podpira tako Javo EE različico 5.0 kot tudi različico 6.0 in vključuje širok nabor podprtih standardov in orodij. Namestiti ga je mogoče tudi v virtualizirana okolja (npr. VMWare<sup>®</sup>).

Obstajata dve različici:

- Liferay Portal Community Edition<sup>®</sup> - je prosto dostopna različica z nekoliko manj funkcionalnosti in le s podporo skupnosti.
- Liferay Portal Enterprise Edition<sup>®</sup> - vsebuje uradno podporo proizvajalca, poleg tega pa je deležna dodatnega testiranja in je običajno na voljo nekoliko kasneje kot prosto dostopna različica.

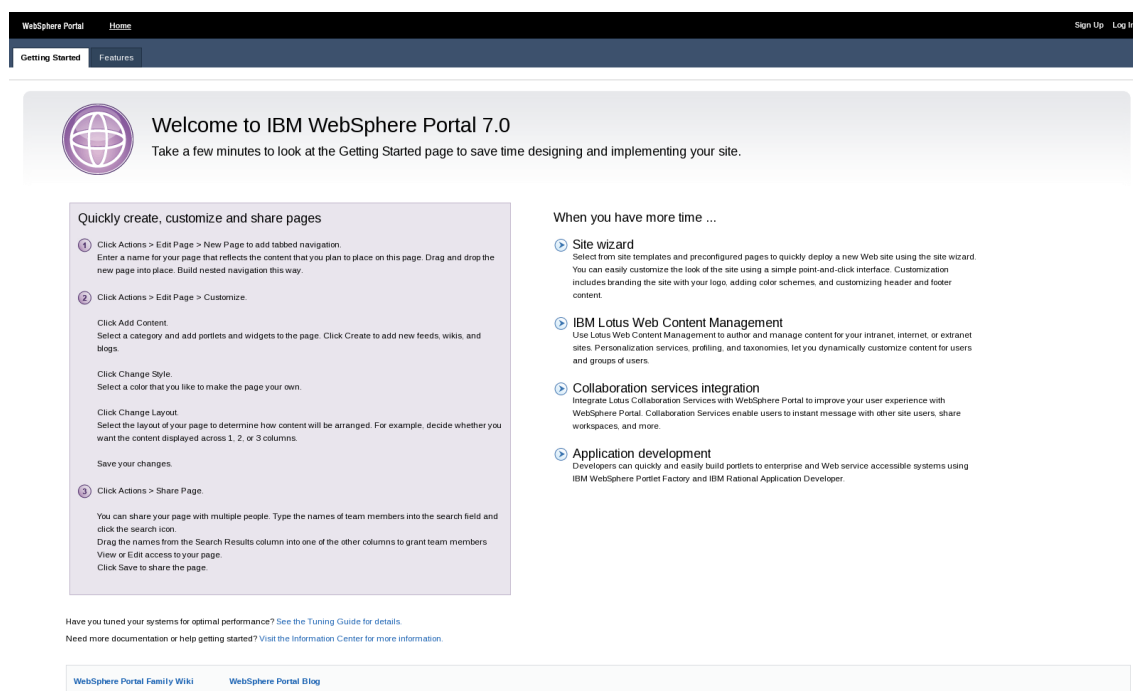
Gartner<sup>®</sup> je septembra leta 2008 Liferay imenoval za enega izmed vodilnih v Gartnerjevem „magičnem kvadrantu“, ki ocenjuje portale („Gartner’s Magic Quadrant for Horizontal Portal Products“) [20]. Med pomembnejšimi uporabniki Liferay portala so podjetja Allianz Insurance, francosko obrambno ministrstvo, Cisco Developer Network, Lufthansa, Vodafone in Volkswagen [19].

Na sliki 2.2 je primer strani na prosto dostopni različici portala Liferay.

### 2.2.3.2 IBM WebSphere Portal<sup>®</sup>

WebSphere Portal je poslovni portal proizvajalca IBM in je del širokega nabora poslovnih orodij, ki so poznana pod imenom WebSphere. Razvit je na osnovi Java EE, vendar s pomembno razliko v primerjavi s preostalimi portali. IBM WebSphere Portal namreč uporablja IBM-ovo različico Jave, ki je kompatibilna z Oraclovo različico. Prvo različico je IBM ponudil na trgu leta 2001 za operacijski sistem AIX (osnovan na Unix-u), kasneje pa je dodal podporo za različne operacijske sisteme. Zadnjo različico 7.0 je tako moč namestiti na Linux, Microsoft Windows, HP-UX, Solaris, i5/OS in z/OS. IBM ga ponuja v več različicah [11]:

- WebSphere Portal Server<sup>®</sup> (osnovna različica, ki ponuja ključne funkcionalnosti)
- WebSphere Portal Enable<sup>®</sup> (nadgrajena različica Portal Server-ja z dodatki za obvladovanje vsebine in naprednim iskanjem prek različnih virov)
- WebSphere Portal Extend<sup>®</sup> (nadgrajena različica Portal Enable z dodatki za podporo delu v skupinah)



Slika 2.11: Vstopna stran IBM WebSphere Portal-a (različice Express)

- WebSphere Portal Express<sup>®</sup> (različica namenjena preizkušanju in demonstracijam, t.i. „proof of concept“; vsebuje večino funkcionalnosti)

WebSphere Portal deluje na osnovi aplikacijskega strežnika WebSphere Application Server<sup>®</sup> in DB2 podatkovne baze, ki sta oba tudi dela paketa ob nakupu. Poleg tega ob nakupu kupec prejme še nabor dodatne programske opreme, ki je namenjena predvsem razvoju aplikacij.

Trenutno WebSphere Portal velja za enega od vodilnih poslovnih portalov na svetu, kar se kaže tudi v Gartnerjevem „magičnem kvadrantu“, kjer je bil ocenjen kot eden izmed vodilnih [20]. Med uporabniki WebSphere portala so Institute For Applied Telemedicine (IFAT), Versicherungskammer Bayern, Sennheiser, Duke University, Reliance Life Insurance Corporation, Icelandair, Michelin Tire Corp, University of North Carolina, German Sport University Cologne, University of London, Finnish Defense Forces, Viessmann, Telenor, US Army, Deutsche Lufthansa AG, NYPD, Dassault Aviation in drugi [21]. Primer strani v IBM WebSphere Portal-u prikazuje slika 2.11.

## Poglavje 3

# Medius Vaadin Common

Medius Vaadin Common je ogrodje za avtomatsko generiranje uporabniških vmesnikov poslovnih aplikacij na osnovi označevanja (anotacij) strežniške kode. Razvit je bil v podjetju Medius, natančno pa ga je opisal Viktor Brajak v svojem diplomskem delu „Avtomatsko generiranje grafičnega uporabniškega vmesnika z označevanjem strežniške javanske kode“ [5]. V sledečem poglavju je povzetek opisa delovanja ogrodja ter tehnologij, ki jih ogrodje uporablja. Ogrodje je bilo zasnovano z namenom izdelave uporabniških vmesnikov na različnih odjemalcih (namizna aplikacija, spletna aplikacija in mobilna aplikacija), vendar se bomo v tem poglavju omejili le na spletne aplikacije, ker so le-te predmet integracije na poslovne portale.

### 3.1 Namen ogrodja

Eden izmed temeljev poslovnih aplikacij so podatkovne baze, kjer se hranijo podatki, ki so pomembni za določeno podjetje ali organizacijo. Da bi lahko te podatke urejali, poslovne aplikacije običajno ponujajo grafični uporabniški vmesnik, ki omogoča dodajanje, urejanje in brisanje teh podatkov, dodatno pa lahko ponuja še vmesnike, ki uporabnikom omogočajo filtriranje, sortiranje in agregacijo podatkov. Običajno je tudi zaželeno, da je možno izvoziti podatke v datoteke, ki se uporabljajo v pisarniškem poslovanju (npr. datoteke za programski paket Microsoft Office, PDF datoteke in XML datoteke). Pri množičnemu vnašanju ali prenosu podatkov iz drugih sistemov pa je zaželeno, da je mogoče narediti uvoz iz teh datotek, brez posebnega ročnega vnašanja.

Ker je število različnih podatkov v poslovnih aplikacijah lahko zelo veliko in ker število komponent v grafičnem vmesniku (gumbi, tabeli, meniji, povezavi) hitro narašča, se pogosto zgodi, da programerji porabijo nesorazmerno

veliko časa za programiranje uporabniškega vmesnika, namesto da bi se bolj posvetili poslovni logiki aplikacije. Posebej težavno je to pri spletnih aplikacijah, ki jih uporabniki uporabljajo s pomočjo različnih spletnih brskalnikov, pri katerih lahko pride do odstopanja v prikazu komponent grafičnega vmesnika, zaradi česar mora programer porabiti še več časa, da aplikacija ponuja enoten videz.

Da bi se izognili temu problemu, smo v podjetju Medius razvili svoje ogrodje, ki avtomatsko zgradi uporabniški grafični vmesnik s pomočjo označene strežniške kode. Programer torej poskrbi za ustrezen opis entitetnih razredov, ki so pravzaprav preslikani podatki iz podatkovne baze, na osnovi teh opisov pa ogrodje zgenerira uporabniški vmesnik. Poleg osnovnih funkcionalnosti (vnos, urejanje, brisanje, izvoz) za urejanje podatkov omogoča tudi dodatno prilagajanje uporabniškega vmesnika, če za to obstajajo zahteve. Ključne prednosti ogrodja so [5]:

- enoten razvoj za različna izvajalna okolja
- hitrejši razvoj aplikacij
- širok nabor grafičnih komponent, ki se avtomatično zgradijo iz opisanih komponent
- enoten strežniški podatkovni model in pripadajoča poslovna pravila v različnih izvajalnih okoljih
- neodvisnost obnašanja in oblike grafičnih komponent ne glede na število razvijalcev
- enostavno spletno oblikovanje
- razvoj izključno v programskem jeziku Java, kar omogoča lažje vzdrževanje

## 3.2 Komponente ogrodja

### 3.2.1 Izvajalno okolje

Medius Vaadin Common je razvit na osnovi Jave EE, ki je bila izbrana zato, ker podpira vse ključne funkcionalnosti, ki so potrebne za razvoj obsežnih in kompleksnih poslovnih aplikacij (npr. zanjo obstajajo kakovostna razvijalna orodja IDE, omogoča sočasnost razvoja z več razvijalci, vsebuje prevajalnike

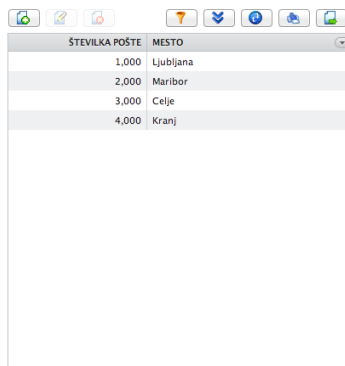
kode, ki generirajo robustno in hitro kodo idr.). Za generiranje grafičnega vmesnika je bilo izbrano ogrodje imenovano Vaadin, ki ponuja naslednje prednosti [5]:

- ponuja enotno razvojno okolje, saj se uporablja le programski jezik Java, za prevedbo na standardne spletne tehnologije (JavaScript, Ajax, JSON, HTML, CSS) pa poskrbi ogrodje samo in je tako transparentno za programerja
- je okolje, ki omogoča prenos dogodkov in delnih sprememb form prek protokola http, torej ni potrebno osveževanje celotne strani za vsako spremembo
- poslovna logika aplikacije je ločena od prikaza in omogoča enostavnejšo vključitev oblikovalcev pri razvoju aplikacije
- ker razvoj poteka le v Javi, je mogoč enostaven hkraten razvoj več programerjev na istem projektu
- ogrodje se lahko enostavno vključi v Java EE

Ogrodje Medius Vaadin Common gradi na trinivojski arhitekturi, pri kateri imamo na podatkovnem nivoju podatkovno bazo, na poslovnem nivoju aplikacijski strežnik, na katerem se izvaja poslovna logika ter na nivoju odjemalca spletni brskalnik.

Razvoj na ogrodju začnemo z načrtovanjem podatkovne baze. Ko imamo dokončen podatkovni model in realizirano podatkovno bazo na podatkovnem strežniku, lahko na poslovnem nivoju kreiramo programske razrede imenovane entitetna zrna (ang. „entity beans“). Entitetno zrno je programski razred, ki predstavlja določeno tabelo v podatkovni bazi in poskrbi za preslikavo podatkov med podatkovno bazo in aplikacijskim strežnikom. To lahko doseže doseže z ORM („Object-relational mapping“) preslikavo, katere implemetacija v Javi se imenuje JPA (Java Persistence API) in sicer tako, da razrede označi (anotira) z označbami „@Entity“. Ob namestitvi aplikacije na aplikacijski strežnik nato ogrodje Java EE prepozna označbo in generira komponento.

Ko je realizirana povezava med podatkovno bazo in aplikacijskim strežnikom, je potrebno realizirati še generiranje uporabniškega vmesnika, ki omogoči urejanje podatkov. To pomeni kreiranje vnosnih mask, na katerih so entitetna zrna predstavljena v obliki tabel, vsaka vrstica v tabeli pa predstavlja eno instanco entitetnega zrna. Za omogočanje dodajanja, urejanja in brisanja so na maski dodani gumbi, ki odprejo dodatno masko z vnosnimi polji za vnašanje



| ŠTEVILKA POŠTE | MESTO     |
|----------------|-----------|
| 1,000          | Ljubljana |
| 2,000          | Maribor   |
| 3,000          | Celje     |
| 4,000          | Kranj     |

Slika 3.1: Primer osnovne maske za prikaz entitetnega zrna „Post“. Zgoraj levo so gumbi za dodajanje, urejanje in brisanje podatkov. Zgoraj desno pa so gumbi za filtriranje, sortiranje, osveževanje, ter izvoz podatkov v različne formate.

podatkov. Pri kompleksnejših vnosih je mogoče realizirati tudi vnašanje s pomočjo čarovnika, ki razbije vnos na več korakov in tako olajša delo uporabniku. Za sortiranje, filtriranje in izvoz podatkov poskrbijo še dodatni gumbi na osnovni maski. Slika 3.1 prikazuje osnovno masko za prikaz entitetnega zrna „Post“, slika 3.2 pa masko za vnos nove instance omenjenega entitetnega zrna.

Označbe za generiranje uporabniškega vmesnika se doda k strežniški kodi in sicer zato, ker je bilo orodje zasnovano tako, da si aplikacije za različne odjemalce lahko delijo strežniško kodo. V ta namen so nastale dodatne anotacije za risanje grafičnega vmesnika. Slika 3.3 prikazuje osnoven koncept ogrodja Medius Vaadin Common razdeljen po posameznih arhitekturnih nivojih. Posamezni nivoji so podrobneje predstavljeni v naslednjih poglavjih.

### 3.2.2 Podatkovni nivo

Ker ogrodje uporablja orodja za objektno relacijsko preslikovanje (ang. „Object Relational Mapping“ - ORM), je izbira podatkovne baze, ki bo služila kot osnova za poslovno aplikacijo, poljubna. Orodje za preslikovanje namreč poskrbi za avtomatsko preslikovanje med relacijsko podatkovno bazo in entitetnimi zrnji ter tako ustvari navidezno objektno bazo, s katero lahko programer dela na objektni način, ne da bi se obremenjeval s podrobnostmi predstavitve podatkov v relacijski bazi. To posledično pomeni, da lahko uporabimo katerokoli podatkovno bazo, ki jo podpira določena implementacija ORM. V splošnem velja, da ponudniki ORM orodij podpirajo večino pomembnejših re-

The screenshot shows a web application interface. At the top, there is a toolbar with several icons. Below the toolbar is a table with two columns: 'ŠTEVILKA POŠTE' and 'MESTO'. The table contains four rows of data:

| ŠTEVILKA POŠTE | MESTO     |
|----------------|-----------|
| 1,000          | Ljubljana |
| 2,000          | Maribor   |
| 3,000          | Celje     |
| 4,000          | Kranj     |

Below the table, a modal window titled 'Nov' is displayed. It contains two input fields: 'Številka pošte: \*' and 'Mesto: \*'. Both fields have red asterisks indicating they are required. Below the input fields are two buttons: 'V redu' and 'Prekliči'.

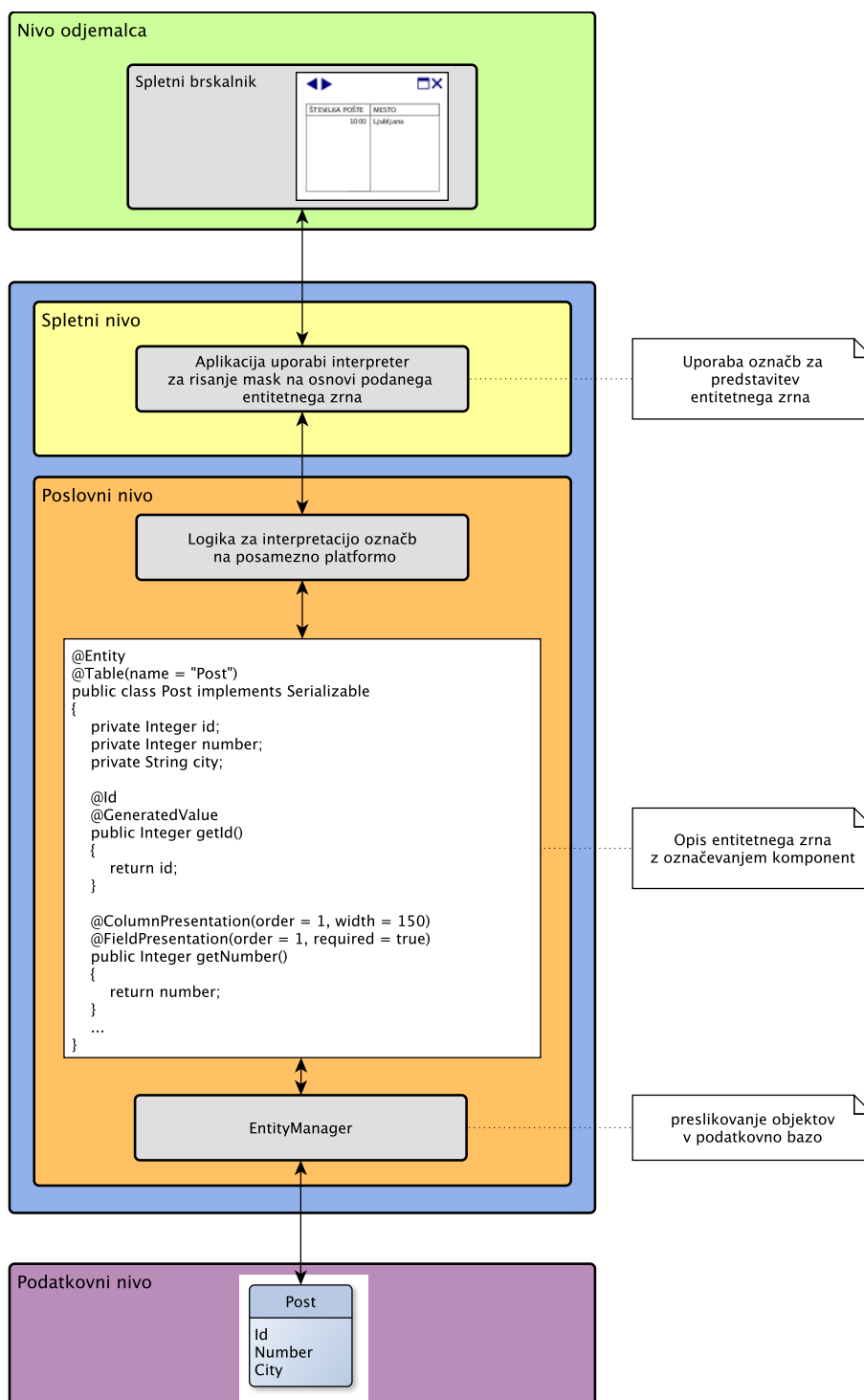
Slika 3.2: Primer vnosne maske, ki vsebuje vnosna polja za vse podatke, ki so potrebni za generiranje instance entitetnega zrna „Post“. Rdeče zvezdice označujejo, da je podatek obvezen.

lacijskih podatkovnih baz (npr. IBM DB2, Oracle, SQL Server, MySQL).

### 3.2.3 Poslovni nivo in spletni nivo

#### 3.2.3.1 Objektno modeliranje podatkov

Na poslovnem nivoju je potrebno najprej poskrbeti za ustrezno preslikavo iz relacijske baze v entitetna zrna, ki jih definira specifikacija EJB 3.0 (ang. „Enterprise Java Beans“). Entitetna zrna v poslovnih aplikacijah predstavljajo podatke, ki jih želimo shraniti v podatkovni bazi, ne vsebujejo pa metod za implementacijo delovnega toka aplikacije. To so navadni objekti (ang. „plain old java objects“ - POJO), ki so dodatno opisani z označbami ter določajo lastnosti objekta in vrsto preslikave v relacijsko podatkovno bazo [5]. Za samo preslikavo pa skrbi vsebnik EJB 3.0 (ang. „EJB3 container“), to je program, ki poskrbi za večino sistemskih podrobnosti in tako olajša delo programerju. Za tabelo Post, ki jo na podatkovnem nivoju definiramo z sql skripto, ki jo pri-



Slika 3.3: Koncept ogrodja Medius Vaddin Common po posameznih arhitekturnih nivojih

kazuje izvorna koda 3.1, bi lahko definicija entitetnega zrna izgledala tako, kot to prikazuje izvorna koda 3.2. Označba „@Entity“ označuje, da gre v primeru tega razreda za entitetno zrno, označba „@Table(name=“Post“)“ pa vsebniku EJB pove ime tabele, v kateri se nahajajo podatki za to entitetno zrno. Označba „@Id“ označuje primarni ključ, pri katerem označba „@GeneratedValue“ dodatno določi, da gre za avtomatsko generirano vrednost. Entitetna zrna lahko vsebujejo še naslednje anotacije:

- @Transient označuje nestatično metodo, ki ni obstojna, torej ne vsebuje ustreznega stolpca v tabeli podatkovne baze. Če neka metoda nima te oznake, zanjo velja nastavitev @Basic, kar pomeni, da bo vrednost parametra, na katerega se nanaša, zapisana v bazo.
- @Column anotacija opisuje stolpce tabele, ki jih uporabljamo pri preslikovanju

Zelo pomembne so tudi anotacije, ki opisujejo povezave med posameznimi entitetnimi zrnji. Opišemo jih z naslednjimi anotacijami:

- @OneToOne označuje povezavo ena proti ena
- @OneToMany označuje povezavo ena proti mnogo
- @ManyToOne označuje povezavo mnogo proti ena
- @ManyToMany označuje povezavo mnogo proti mnogo

Ko imamo definirano entitetno zrno, lahko definiramo sejno zrno. Sejno zrno vsebuje poslovne metode, ki operirajo z entitetami (shranjevanje, spreminjanje, brisanje in ostale specifične metode, odvisne od konteksta aplikacije). Izvorna koda 3.3 prikazuje kako bi lahko izgledalo sejno zrno definirano za entitetno zrno „Post“. V tem primeru sejno zrno vsebuje naslednje metode:

- getAll(); metoda vrne vse instance entitetnega zrna „Post“
- save(Post post); metoda shrani podano instanco entitetnega zrna „Post“ v podatkovno bazo
- delete(Post post); metoda zbriše podano instanco entitetnega zrna „Post“ iz podatkovne baze

Za spremljanje stanja in sinhronizacijo s podatkovno bazo skrbi storitev imenovana entitetni upravljalec (ang. EntityManager), katerega referenca se nahaja tudi v sejnem zrnju. Entitetni upravljalec vsebuje več metod za delo z entitetnimi zrnji, med njimi:

---

**Izvorna koda 3.1** Izsek skripte za generiranje tabele Post v podatkovni bazi Oracle

---

```
/*=====*/
/* Table: "Post"
*/
/*=====*/
create table "Post"
(
    "Id"                INTEGER                not null ,
    "Number"            INTEGER                not null ,
    "City"              VARCHAR2(500)          not null ,
    "Valid"             SMALLINT
default 1 not null ,
    constraint PK_POST primary key ("Id")
)
/
```

---

- `persist()` metoda preslika instanco entitetnega zrna, ki še ni bila ustvarjena, v podatkovno bazo. Uporabimo jo lahko npr. v metodi `save()` znotraj sejnega zrna
- `merge()` metoda preslika spremembe, ki so bile narejene na določeni instanci entitetnega zrna v podatkovno bazo
- `remove()` metoda izbriše instanco entitetnega zrna iz podatkovne baze
- `refresh()` metoda osveži entiteto (vrne trenutno stanje iz podatkovne baze)

Z uporabo teh metod lahko upravljamo z instancami entitetnih zrn brez pisanja SQL poizvedb. Tak primer prikazuje metoda „`save(Post post)`“ v izvorni kodi 3.3. Če pa programer potrebuje neko specifično poizvedbo z dodanimi pogoji lahko definira svojo poizvedbo s pomočjo poizvedovalnega jezika, ki je neodvisen od ciljne podatkovne relacijske baze, za pretvorbo v SQL poizvedbo, ki jo razume ciljna baza, pa poskrbi ogrodje. Tak primer je v metodi „`getAll()`“.

Tudi sejna zrna so objekti (POJO), ki postanejo upravljani objekti, ko jih povežemo s storitvijo Entity Manager. Za preslikovanje v Java EE skrbi knjižnica JPA (Java Persistence API), tako preslikovanje pa imenujemo preslikovanje z označevanjem JPA [5].

---

**Izvorna koda 3.2** Definicija entitetnega zrna na osnovi tabele Post iz relacijske podatkovne baze.

---

```
@Entity
@Table(name = "Post")
public class Post implements Serializable
{
    private Integer id;
    private Integer number;
    private String city;

    public Post(Integer id, Integer number, String city)
    {
        super();
        this.id = id;
        this.number = number;
        this.city = city;
    }

    @Id
    @GeneratedValue
    public Integer getId()
    {
        return id;
    }

    public void setId(Integer id)
    {
        this.id = id;
    }

    public Integer getNumber()
    {
        return number;
    }

    public void setNumber(Integer number)
    {
        this.number = number;
    }

    ...
}
```

---

---

**Izvorna koda 3.3** Definicija sejnega zrna za entiteto Post.

---

```
@Stateless
public class PostManager implements PostManagerLocal
{
    @PersistenceContext
    private EntityManager em;

    public Post delete(Post post)
    {
        String query = "";
        query += "DELETE FROM Post p ";
        query += "WHERE p.id = :param_idPost ";

        Query q = em.createQuery(query);
        q.setParameter("param_idPost", post.getId());
        q.executeUpdate();

        return post;
    }

    @SuppressWarnings("unchecked")
    public List<Post> getAll()
    {
        return em.createQuery
            ("SELECT p
             FROM Post p
             order by p.id asc").getResultList();
    }

    public Post save(Post post)
    {
        return em.merge(post);
    }
}
```

---

### 3.2.3.2 Modeliranje grafičnega uporabniškega vmesnika

Ko smo definirali sejna zrna, smo zaključili z objektnim modeliranjem podatkov v aplikaciji. Sedaj potrebujemo še označbe, s katerimi ogrodju Medius Vaadin Common povemo, kakšen prikaz želimo na uporabniškem vmesniku. Kot smo že prej omenili, tudi te označbe dodamo strežniški kodi. V osnovi jih razdelimo na tri skupine:

- Anotacije Editor so namenjene opisovanju mask za urejanje (dodajanje novih in urejanje obstoječih) instanc entitetnega zrna. Opisujejo razporeditev in način prikaza polj na maski.
- Anotacije Table opisujejo predstavitev podatkov v tabelah na osnovnih uporabniških maskah.
- Anotacije Lookup so anotacije, ki opisujejo kreiranje izbirnih mask.

V nadaljevanju sledi opis tistih anotacij, ki se bodo uporabljale na primeru uporabe, ki bo služil za potrebe integracije na različne poslovne portale. Samo ogrodje ponuja bistveno večji nabor anotacij, ki pa za razumevanje tega diplomskega dela niso bistvene, zato bomo njihov opis namenoma izpustili (podroben opis vseh anotacij se nahajav diplomskem delu „Avtomatsko generiranje grafičnega uporabniškega vmesnika z označevanjem strežniške javanske kode“ [5]).

Izmed anotacij Editor se na primeru uporablja anotacijo `@FieldPresentation`. To je anotacija, ki opiše, kakšen naj bo prikaz in obnašanje posameznega atributa entitetnega zrna na maski za urejanje. S to anotacijo anotiramo metode `„get()“` v entitetnih zrnih, ki se nanašajo na določen atribut entitetnega zrna (t.i. `„getter metode“`). Vsebuje lahko več parametrov, med njimi:

- `order (int)`  
Določi vrstni red prikaza atributa. Posamezni atributi se na maski prikažejo v naraščajočem vrstnem redu vrednosti parametra `order`.
- `type (Fieldtype)`  
Določi tip polja. Možne vrednosti so `NORMAL` (prikaže se navadno tekstovno polje), `MULTILINE` (prikaže se tekstovno polje z več vrsticami), `LOOKUP` (prikaže se meni za izbiro enega elementa; uporablja se pri relacijah mnogo proti ena), `LOOKUP_SIMPLE` (enako kot `LOOKUP`, le da se uporabi preprost spustni meni), `LOOKUP_MANY` (prikaže meni za relacije ena proti mnogo), `LOOKUP_MANYTOMANY` (prikaže iskalnik za relacije mnogo proti mnogo).

- `required` (boolean)  
Določi ali je vnos vrednosti v polje obvezen.
- `lookupClass` (Class)  
Določi razred entitetnega zrna, ki bo prikazan v meniju za izbiro.
- `manyToManyClass` (Class)  
Določi vmesno entiteto pri kreiranju iskalnika `LOOKUP_MANYTOMANY`.
- `manyToManyBackProperty` (String)  
Parameter določa, kateri atribut vmesne entitete se uporabi kot prvi del razmerja. Drugi del se določi s parametrom `manyToManyProperty` (String).

Izmed anotacij `Table` primer uporabe vsebuje anotacijo `@ColumnPresentation`, ki določi, kako naj bo nek atribut prikazan v tabeli. Tudi to anotacijo uporabimo na metodah „`get()`“. Izmed parametrov se v primeru uporabe uporabljajo naslednji:

- `order` (int)  
Določi vrstni red prikaza stolpca v tabeli. Stolpci so prikazani v naraščajočem vrstnem redu vrednosti parametra `order`.
- `width` (int)  
Določa širino stolpca v tabeli (število slikovnih točk po širini). Privzeta vrednost je -1 in v tem načinu se širine stolpcev določijo avtomatsko.
- `align` (ColumnAlignType)  
Določa poravnavo besedila v stolpcu tabele.

Pomembna je še anotacija `@LookupPresentation`, ki določi, kako je predstavljen izbirni meni pri izbiri referenčiranega entitetnega zrna. Če imamo npr. entitetno zrno „`Post`“, ki vsebuje številke in imena pošt ter drugo entitetno zrno „`Address`“, ki poleg ulice in hične številke vsebuje tudi objekt tipa „`Post`“, lahko z anotacijo `@LookupPresentation` povemo, kako naj se na maski za urejanje entitete „`Address`“ pokaže možnost za izbiro določene pošte. Anotacijo se uporabi na nivoju razreda. Med drugim vsebuje naslednje parametre:

- `name` (String)  
Določi ime iskalnika.
- `manager class` (Class)  
Sejno zrno, ki se uporablja za dostop do podatkov obravnavanega entitetnega zrna.

---

**Izvorna koda 3.4** Primer anotacij „getter“ metod za entitetno zrno „Post“.

---

```
@ColumnPresentation(order = 1, width = 150)
@FieldPresentation(order = 1, required = true)
public Integer getNumber()
{
    return number;
}

@ColumnPresentation(order = 2)
@FieldPresentation(order = 2, required = true)
public String getCity()
{
    return city;
}
```

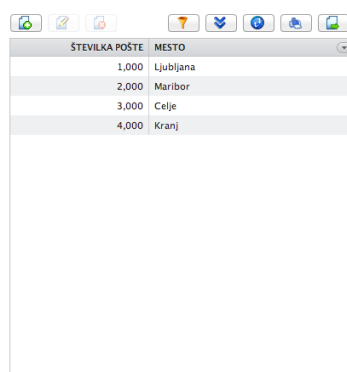
---

- format (String)

Določa, kako naj bodo izbrani podatki predstavljeni v iskalnem polju.

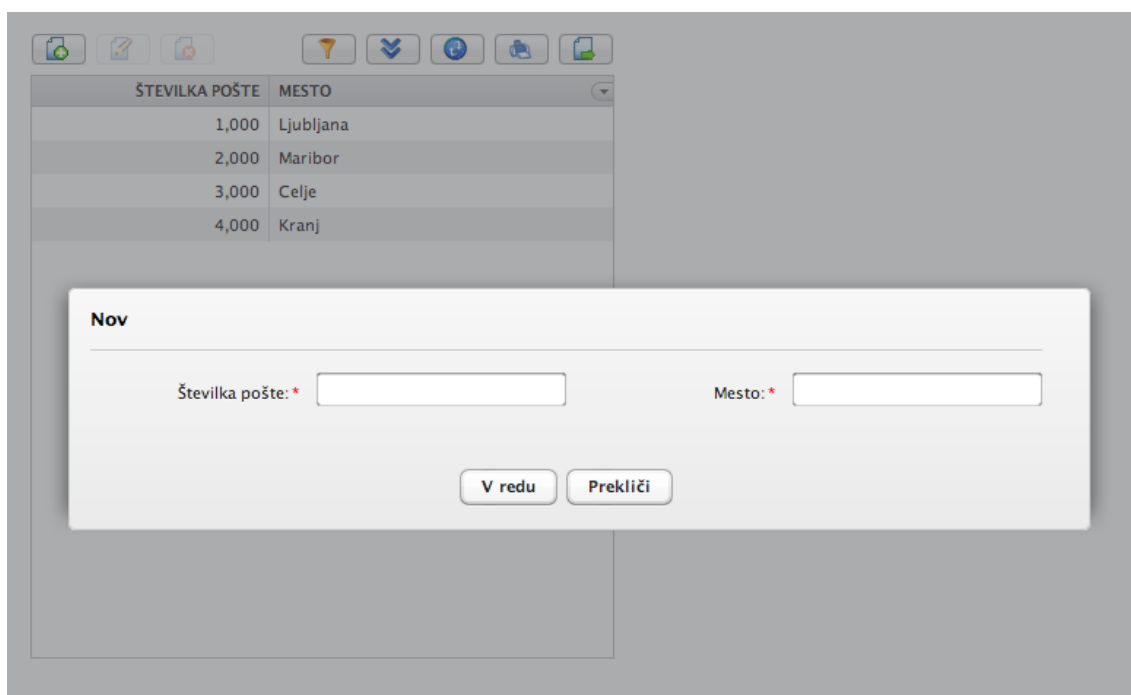
Izvorna koda 3.4 prikazuje primer preprostih anotacij za attribute entitetnega zrna „Post“, ki vsebuje tri attribute. Anotacija `@ColumnPresentation` pri metodi „`getNumber()`“ določa, da bo atribut „number“ v tabeli predstavljen v prvem stolpcu, katerega širina bo 150 slikovnih točk. Anotacija `@FieldPresentation` pri istem atributu pa določa, da bo na maski za kreiranje in urejanje instance tega entitetnega zrna vnosno polje za atribut „number“ na prvem mestu, vnos pa je obvezen. Podoben je pomen anotacij pri metodi „`getCity()`“ s to razliko, da pri anotaciji `@ColumnPresentation` ni parametra „width“, kar pomeni, da bo širino stolpca določena avtomatsko. Ker tabela vsebuje le stolpca za atributa „number“ in „city“ (atribut „id“ nima anotacije `@ColumnPresentation`, torej v tabeli ne bo prikazan) in ker je širina stolpca za „number“ fiksno določena, bo stolpec za atribut „city“ zavzel vso preostalo širino tabele. Slika 3.4 prikazuje generirano masko na osnovi anotacij `@ColumnPresentation`, slika 3.5 pa primer vnosne maske generirane na osnovi opisanih `@FieldPresentation`.

Naslednji primer prikazuje relacijo med entitetnima zrnoma „Address“ in „Post“, ki sta oblike mnogo proti ena. Ko dodajamo torej novo instanco entitetnega zrna „Address“, želimo v maski za dodajanje imeti možnost izbire natančno ene instance entitetnega zrna „Post“. To dosežemo z anotacijo `@FieldPresentation`, kateri določimo parameter „type“ in mu dodelimo vrednost „`FieldType.LOOKUP_SIMPLE`“, kar se v generirani maski pokaže kot spustni



| ŠTEVILKA POŠTE | MESTO     |
|----------------|-----------|
| 1,000          | Ljubljana |
| 2,000          | Maribor   |
| 3,000          | Celje     |
| 4,000          | Kranj     |

Slika 3.4: Primer generirane maske za prikaz entitetnega zrna „Post“ na osnovi izvorne kode 3.4



| ŠTEVILKA POŠTE | MESTO     |
|----------------|-----------|
| 1,000          | Ljubljana |
| 2,000          | Maribor   |
| 3,000          | Celje     |
| 4,000          | Kranj     |

**Nov**  
  

Številka pošte: \*

Mesto: \*

Slika 3.5: Primer generirane vnosne maske za kreiranje nove instance entitetnega zrna „Post“

---

**Izvorna koda 3.5** Primer anotacij „getter“ metod za entitetno zrno „Address“.

---

```
@FieldPresentation(order = 4, required = true,
                    type = FieldType.LOOKUP_SIMPLE)
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "fk_idPost",
            referencedColumnName = "Id",
            nullable = false)
public Post getPost()
{
    return post;
}
```

---

meni, iz katerega lahko izberemo natančno eno vrednost. Definirati moramo še povezavo med entitetnima zrnoma, kar dosežemo z anotacijo `@ManyToOne` one na metodi „getPost()“ v entitetnem zrnju „Address“ (torej več naslovov ima lahko enako pošto) in anotacijo `@JoinColumn(...)`, ki ji določimo parametra „name“ (vsebuje naslov atributa relacije Address v relacijski bazi, ki vsebuje tuji ključ relacije Post) in „referencedColumnName“ (določa ime atributa v relaciji Post, ki predstavlja osnovo za tuji ključ v tabeli Address). Primer teh anotacij prikazuje izvorna koda 3.5, generirano vnosno masko pa slika 3.6.

Izvorna koda 3.6 prikazuje obraten primer, torej relacijo ena proti mnogo na entitetnih zrnih „Customer“ in „Address“ (torej stranka ima lahko več naslovov). V tem primeru želimo na vnosni maski prikaz seznama vseh obstoječih naslovov stranke in možnost izbire in vnosa novih. To dosežemo tako, da v entitetnem zrnju „Customer“ anotiramo metodo „getAddresses()“ (ki vrača niz instanc tipa „Address“) z anotacijo `@FieldPresentation`, ki ji določimo parameter „type“ z vrednostjo „FieldType.LOOKUP\_MANY“, dodatno pa moramo še določiti parameter „lookupClass“, v katerem določimo razred referenciiranega entitetnega zrna. Tako anotiran razred kreira vnosno masko, kjer je eno od polj pravzaprav tabela vseh instanc entitetnega zrna „Address“, ki so ji dodani gumbi za dodajanje, urejanje ter brisanje posameznih instanc.

Tudi v tem primeru moramo definirati povezavo med obema entitetama. To na strani entitetnega zrna naredimo z anotacijo `@OneToMany`, ki ji določimo parameter „mappedBy“, ki pove ime atributa na referenciiranem entitetnem zrnju, ki vsebuje referenco trenutnega razreda (v tem primeru ima entitetno zrno „Address“ atribut „customer“, ki je referenca na instanco entitetnega

| ULICA      | HIŠNA ŠT. | POŠTA            | KRAJ      | DRŽAVA    |
|------------|-----------|------------------|-----------|-----------|
| Slovenska  | 12        | 1000 - Ljubljana | Ljubljana | Slovenija |
| Prešernova | 2         | 1000 - Ljubljana | Ljubljana | Slovenija |
| Slovenska  | 3         | 1000 - Ljubljana | Ljubljana | Slovenija |

**Nov**

Ulica: \*

Pošta: \*

Država: \* 

1000 - Ljubljana
2000 - Maribor
3000 - Celje
4000 - Kranj

Hišna št.: \*

Kraj: \*

Prekliči

Slika 3.6: Primer vnosne maske zgrajeni na izvorni kodi z anotacijo `@FieldPresentation`, ki vsebuje parameter „type“ z vrednostjo „FieldType.LOOKUP\_SIMPLE“

zrna „Customer“). Na nasprotni strani relacije pa ustvarimo atribut, katerega tip je ustrezen razred izvornega entitetnega zrna (prej omenjeni „customer“), metodo „getCustomer()“ pa anotiramo z anotacijo „ManyToOne“ in `@JoinColumn`, v kateri določimo ustrezne parametre (atribut „name“ določi ime atributa, ki je tuji ključ v relaciji; atribut „referencedColumnName“ pa določi ime atributa v nasprotni relaciji, ki je osnova za referenco). Slika 3.7 prikazuje primer vnosne maske zgrajene na osnovi teh anotacij.

Zadnji primer je prikaz možne rešitve za implementacijo relacije mnogo proti mnogo. Predpostavimo, da imamo entitetno zrno „Order“, ki predstavlja naročila in entitetno zrno „Product“, ki predstavlja izdelke. Radi bi implementirali relacijo, kjer imamo lahko na enem naročilu več postavk izdelkov, določen izdelek pa je lahko na več naročilih. V ta namen definiramo še entitetno zrno „OrderItem“, ki predstavlja postavke posameznega naročila. Relacijo mnogo proti mnogo nato implementiramo tako, da imamo na maski klasično tabelo naročil, kjer ob gumbu za dodajanje prikažemo masko za dodajanje novega naročila. S tem je definiran levi del relacije. Desni del relacije (torej postavke izdelkov) pa prikažemo kot tabelo postavk na vnosni maski, ki ima

---

**Izvorna koda 3.6** Anotacije entitetnih zrn „Customer“ in „Address“, ki so potrebne za generiranje polja tipa „FieldType.LOOKUP\_MANY“

---

```
...
public class Customer implements Serializable
{
    private Set<Address> addresses = new HashSet<Address>();
    ...
    @FieldPresentation(order = 4, required = true,
                        type = FieldType.LOOKUP_MANY,
                        lookupClass = Address.class)
    @OneToMany(fetch = FetchType.LAZY, mappedBy = "customer")
    public Set<Address> getAddresses()
    {
        return addresses;
    }
}

...
public class Address implements Serializable
{
    private Customer customer;
    ...
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "fk_idCustomer",
                referencedColumnName = "Id",
                nullable = true)
    public Customer getCustomer()
    {
        return customer;
    }
}
```

---

The screenshot shows a web application interface. At the top, there is a table with the following data:

| IME   | PRIIMEK | DATUM ROJSTVA    | OMOGOČEN                            | NASLOVI                                           |
|-------|---------|------------------|-------------------------------------|---------------------------------------------------|
| Janez | Novak   | 07.09.2011 21:38 | <input checked="" type="checkbox"/> | Slovenska 3, Ljubljana<br>Prešernova 2, Ljubljana |

Below the table is a modal window titled "Uredi". It contains the following fields and controls:

- Ime:** \* Janez
- Priimek:** \* Novak
- Datum rojstva:** \* 07.09.2011 21:38
- Naslovi:** \*
 

| ULICA      | HIŠNA ŠT. | POŠTA            | KRAJ      | DR        |
|------------|-----------|------------------|-----------|-----------|
| Slovenska  | 3         | 1000 - Ljubljana | Ljubljana | Slovenija |
| Prešernova | 2         | 1000 - Ljubljana | Ljubljana | Slovenija |
- ☒ Omogočen
- Buttons: V redu, Prekliči

Slika 3.7: Primer vnosne maske zgrajeni na izvorni kodi z anotacijo `@FieldPresentation`, ki vsebuje parameter „type“ z vrednostjo „FieldType.LOOKUP\_MANY“

dodane gumbе za dodajanje, brisanje in urejanje postavk. Ko pritisnemo na gumb za dodajanje postavk za razliko od relacije ena proti mnogo ne dobimo vnosne maske za entitetno zrno, temveč seznam obstoječih instanc entitetnega zrna „Product“. Na seznamu izberemo željene izdelke in sistem doda izdelke kot postavke na naročilo. S pritiskom na gumb za brisanje se zbrše določeno postavko, s pritiskom na gumb za urejanje postavke pa se nastavlja dodatne attribute, ki jih ima vmesni entitetni tip (v našem primeru količino izdelkov).

Za doseg take maske je potrebno uporabiti več anotacij.

Na entitetnem zrnju „Order“ je potrebno definirati seznam instanc entitetnega zrna „OrderItem“ (torej postavk naročila), metodo „getOrderItems()“ pa anotiramo z dvema anotacijama. Prva je anotacija „@OneToMany“, ki ji določimo parameter „mappedBy“ z vrednostjo, katera predstavlja ime atributa v vmesnem entitetnem tipu, ki je referenca na entitetno zrno „Order“. S tem povežemo entitetni zrna „Order“ in „OrderItem“. Druga anotacija je

@FieldPresentation, ki ji določimo naslednje parametre:

- „manyToManyBackProperty“ je parameter, ki določa atribut vmesnega entitetnega tipa, ki je referenca na levi del relacije, v našem primeru je to „order“ oz. naročilo
- „type“ določi način predstavitve entitetnega tipa „OrderItem“, vrednost nastavimo na FieldType.LOOKUP\_MANYTOMANY
- „manyToManyProperty“ definira ime atributa v vmesnem entitetnem tipu, ki predstavlja referenco na desni del relacije, torej v našem primeru „product“,
- „manyToManyClass“ določi ime entitetnega zrna, ki predstavlja vmesno entiteto
- „lookupClass“ določi katero entitetno zrno se uporabi v izbirnem meniju, v našem primeru je to „Product“

Izvorna koda 3.7 prikazuje primer anotacij na entitetnem zrnu „Order“.

Sledijo anotacije na vmesnem entitetnem tipu „OrderItem“, ki mora vsebovati dva atributa in sicer „order“ in „product“, katerih tipa sta ustrezni entitetni zrna („Order“ in „Product“). Metodo „getOrder()“ anotiramo z anotacijama „@ManyToOne“ in „@JoinColumn“ z ustreznimi parametri (s tem povežemo entitetni zrna „Order“ in „OrderItem“), podobno pa anotiramo še metodo „getProduct()“ (s tem povežemo entitetni zrna „OrderItem“ in „Product“). Izvorna koda 3.8 prikazuje primer anotacij na entitetnem zrnu „OrderItem“.

Tako anotirana entitetna zrna služijo za generiranje vnosne maske naročila, kot ga prikazuje slika 3.8. Ko pritisnemo na gumb za dodajanje postavk se odpre seznam obstoječih izdelkov (slika 3.9), v katerem lahko označimo več izdelkov, ki jih želimo dodati na naročilo.

### 3.2.3.3 Povezava med poslovnim nivojem in nivojem odjemalca

Za dejansko interpretacijo anotacij in generiranje uporabniških mask v ogrodju Medius Common Vaddin poskrbi interpreter (samega delovanja interpreterja ne bomo natančno opisovali, ker ni bistven za razumevanje osnovnega delovanja ogrodja), ki poleg lastnih metod uporablja še komponente razvojnega ogrodja Vaadin. Od tu naprej strežniški del ogrodja Vaadin poskrbi za vse podrobnosti komunikacije z odjemalcem.

---

**Izvorna koda 3.7** Anotacije entitetnega zrna „Order“, ki so potrebne za generiranje polja tipa „FieldType.LOOKUP\_MANYTOMANY“

---

```

...
public class Order implements Serializable
{
    private List<OrderItem> orderItems = new LinkedList<OrderItem>();
    ...
    @OneToMany(mappedBy = "order",
                fetch = FetchType.LAZY)
    @FieldPresentation(order = 4,
                       manyToManyBackProperty = "order",
                       type = FieldType.LOOKUP_MANYTOMANY,
                       manyToManyProperty = "product",
                       manyToManyClass = OrderItem.class,
                       lookupClass = Product.class)

    public List<OrderItem> getOrderItems()
    {
        return orderItems;
    }
    ...
}

```

---

### 3.2.4 Nivo odjemalca

Na strani odjemalca imamo lahko poljuben spletni brskalnik (pri razvoju Medius Vaadin Common ogrodja se je testiralo v brskalnikih Microsoft Internet Explorer, Mozilla Firefox in Google Chrome). V njem ob zagonu aplikacije teče odjemalski del ogrodja Vaadin, ki prejete podatke s spletnega dela ogrodja preoblikje v ustrezno obliko za ogrodje Google Web Toolkit (GWT - to je ogrodje, na katerem gradi Vaadin, saj omogoča enostaven razvoj kompleksnih spletnih aplikacij, ki temeljijo na AJAX-u). GWT nato generira ustrezno kodo HTML in JavaScript ter prikaže grafični uporabniški vmesnik v brskalniku. Slika 3.10 prikazuje dele ogrodja Medius Vaadin Common, potrebnih za risanje po posameznih arhitekturnih nivojih. Slika 3.11 prikazuje tipičen izgled poslovne aplikacije razvite z ogrodjem Medius Vaadin Common, kot ga vidi končni uporabnik v brskalniku.

---

**Izvorna koda 3.8** Anotacije entitetnega zrna „OrderItem“, ki so potrebne za generiranje polja tipa „FieldType.LOOKUP\_MANYTOMANY“

---

```
...
public class OrderItem implements Serializable
{
    private Order order = new Order();
    private Product product = new Product();

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "fk_idOrder",
                referencedColumnName = "Id",
                nullable = false)
    public Order getOrder()
    {
        return order;
    }

    @FieldPresentation(order = 4,
                        required = true,
                        type = FieldType.LOOKUP_SIMPLE)
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "fk_idProduct",
                referencedColumnName = "Id",
                nullable = false)
    public Product getProduct()
    {
        return product;
    }
    ...
}
```

---

The screenshot displays a software application window. At the top, there is a toolbar with icons for file operations and data management. Below the toolbar is a table with the following columns: ID RAČUNA, DATUM NASTANKA, VELJAVNOST, STRANKA, and POSTAVKE. The table contains one row of data.

| ID RAČUNA | DATUM NASTANKA   | VELJAVNOST                          | STRANKA     | POSTAVKE                 |
|-----------|------------------|-------------------------------------|-------------|--------------------------|
| 2         | 11.11.2011 09:11 | <input checked="" type="checkbox"/> | Janez Novak | Miška (1x)<br>Miška (1x) |

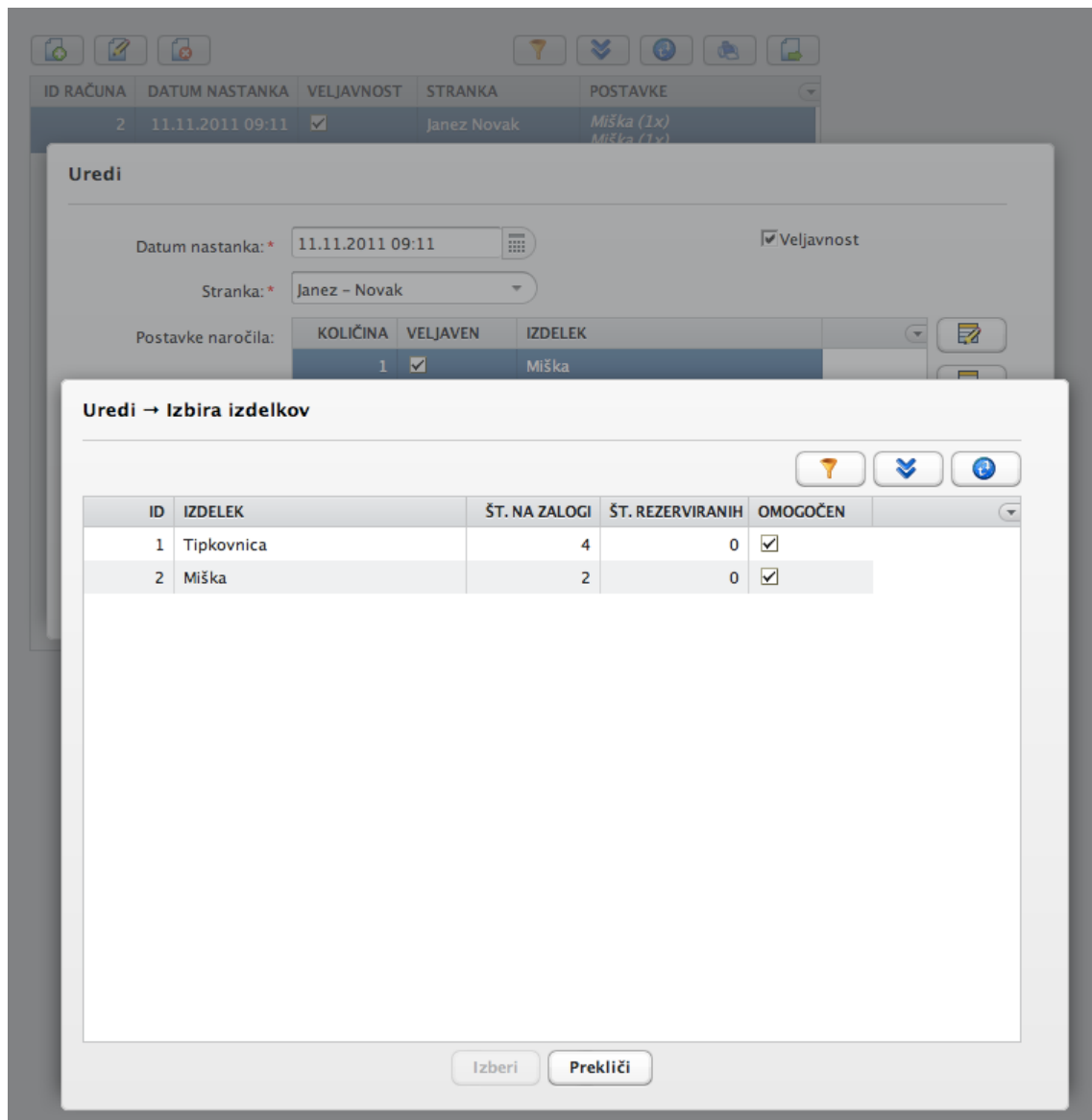
Below the table, a modal window titled "Uredi" is open. It contains the following fields and controls:

- Datum nastanka:** A text field containing "11.11.2011 09:11" and a calendar icon.
- Veljavnost:** A checkbox labeled "Veljavnost" which is checked.
- Stranka:** A dropdown menu showing "Janez - Novak".
- Postavke naročila:** A table with columns KOLIČINA, VELJAVEN, and IZDELEK.

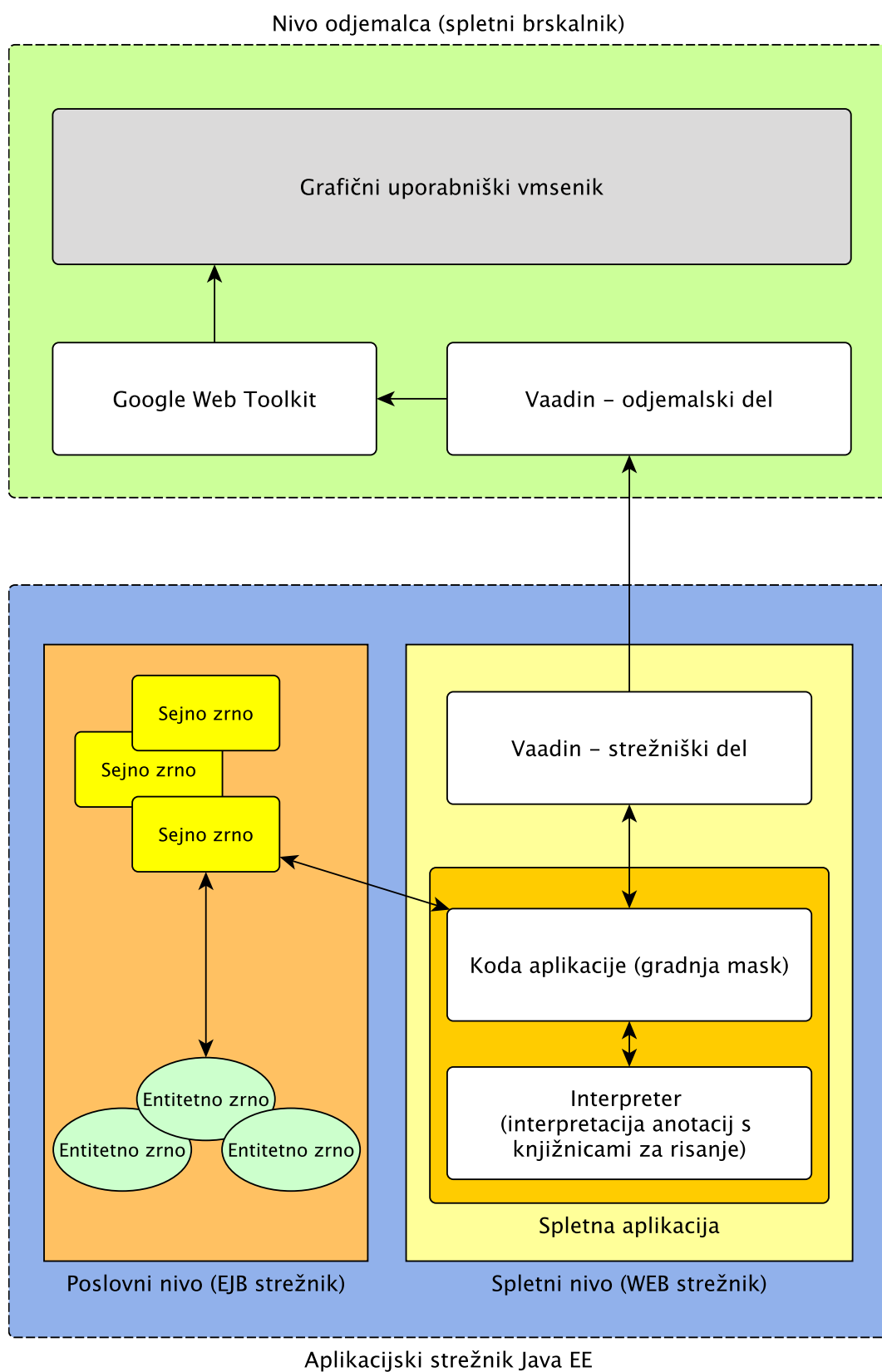
| KOLIČINA | VELJAVEN                            | IZDELEK |
|----------|-------------------------------------|---------|
| 1        | <input checked="" type="checkbox"/> | Miška   |

At the bottom of the modal window, there are two buttons: "V redu" and "Prekliči".

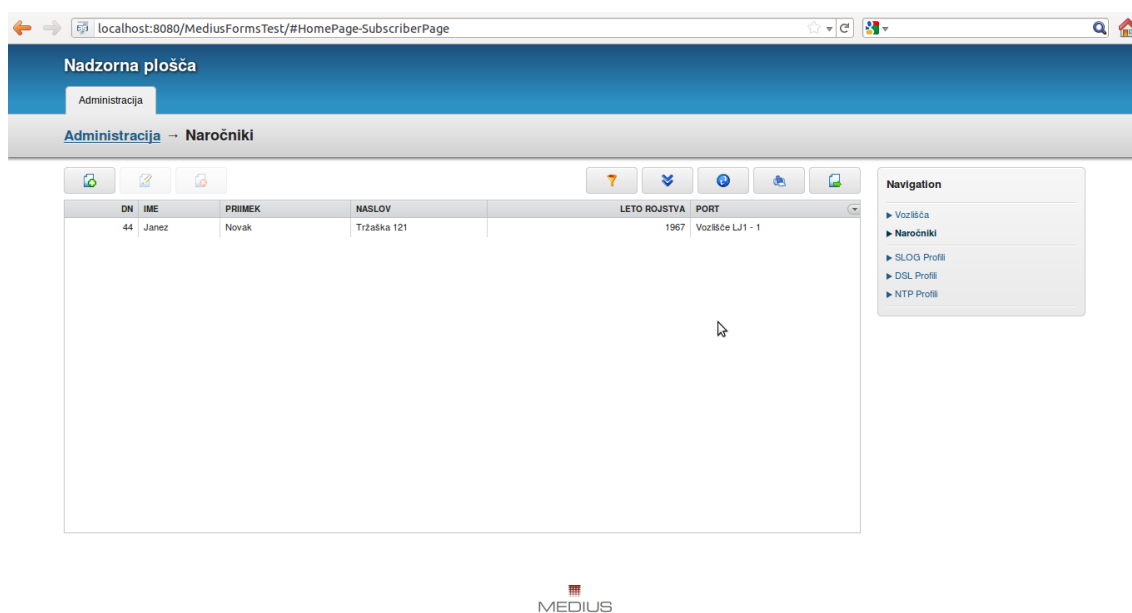
Slika 3.8: Primer vnosne maske za naročila zgrajeni na izvorni kodi z anotacijo `@FieldPresentation`, ki vsebuje parameter „type“ z vrednostjo „FieldType.LOOKUP\_MANYTOMANY“



Slika 3.9: Primer maske za izbiro izdelkov (v vlogi definiranja desnega dela relacije mnogo proti mnogo)



Slika 3.10: Arhitektura gradnje uporabniškega vmesnika



Slika 3.11: Izgled tipične aplikacije razvite z ogrodjem Medius Vaadin Common

## Poglavje 4

# Integracija Medius Vaadin Common na portalih

V tem poglavju sledi opis poskusa integracije ogrodja Medius Vaadin Common na različne poslovne portale. V ta namen je v poglavju najprej definiran primer uporabe, na katerem temelji podatkovna baza in ki bo osnova za kreiranje poslovne aplikacije. To aplikacijo bomo poskusili integrirati. Za oceno uspeha integracije je definiran tudi nabor zahtev, ki morajo biti izpolnjene, da lahko označimo določeno integracijo kot zadovoljivo in seznam kriterijev, ki služijo kot osnova za oceno in primerjavo uspeha integracije.

### 4.1 Priprava primera uporabe

Primer uporabe bo preprosta poslovna aplikacija, ki bo pokrila osnovne funkcionalnosti ogrodja Medius Vaadin Common:

- Osnoven prikaz entitetnega zrna v tabeli
- Dodajanje instance preprostega entitetnega zrna (šifranti) preko vnosne maske
- Dodajanje instance kompleksnega entitetnega zrna (torej zrna, ki vsebuje vnosna polja OneToMany, ManyToOne ali ManyToMany)
- Relacijo entitetnih zrn ena proti mnogo in ustrezen prikaz OneToMany vnosnega polja
- Relacijo entitetnih zrn mnogo proti ena in ustrezen prikaz ManyToOne vnosnega polja

- Relacijo entitetnih zrn mnogo proti mnogo in ustrezno implementacijo tega razmerja

Domena bo podjetje, ki prodaja določene izdelke in želi aplikacijo za administracijo podatkov.

Želijo možnost dodajanja in urejanja strank. Zahtevani podatki so identifikacijska številka, ime, priimek, datum rojstva ter možnost vnosa več naslovov pri isti stranki. Podjetje ponuja različne izdelke, zato potrebujejo tudi urejanje izdelkov, pri čemer so zahtevani podatki identifikacijska številka, ime izdelka, količina na zalogi in število trenutno rezerviranih artiklov. Sistem vsebuje tudi podatke o naročilih (identifikacijska številka naročila, kupec, datum nastanka), vsako naročilo pa lahko vsebuje več postavk izdelkov. Vsaka postavka naročila ima tudi atribut količina, ki pove koliko izdelkov določene vrste je kupec kupil. Določen izdelek se lahko nahaja na različnih naročilih. Vsaka entiteta mora imeti tudi atribut veljavnost.

Na podlagi opisa je bil zasnovan načrt podatkovne baze. Konceptualni nivo prikazuje slika 4.1.

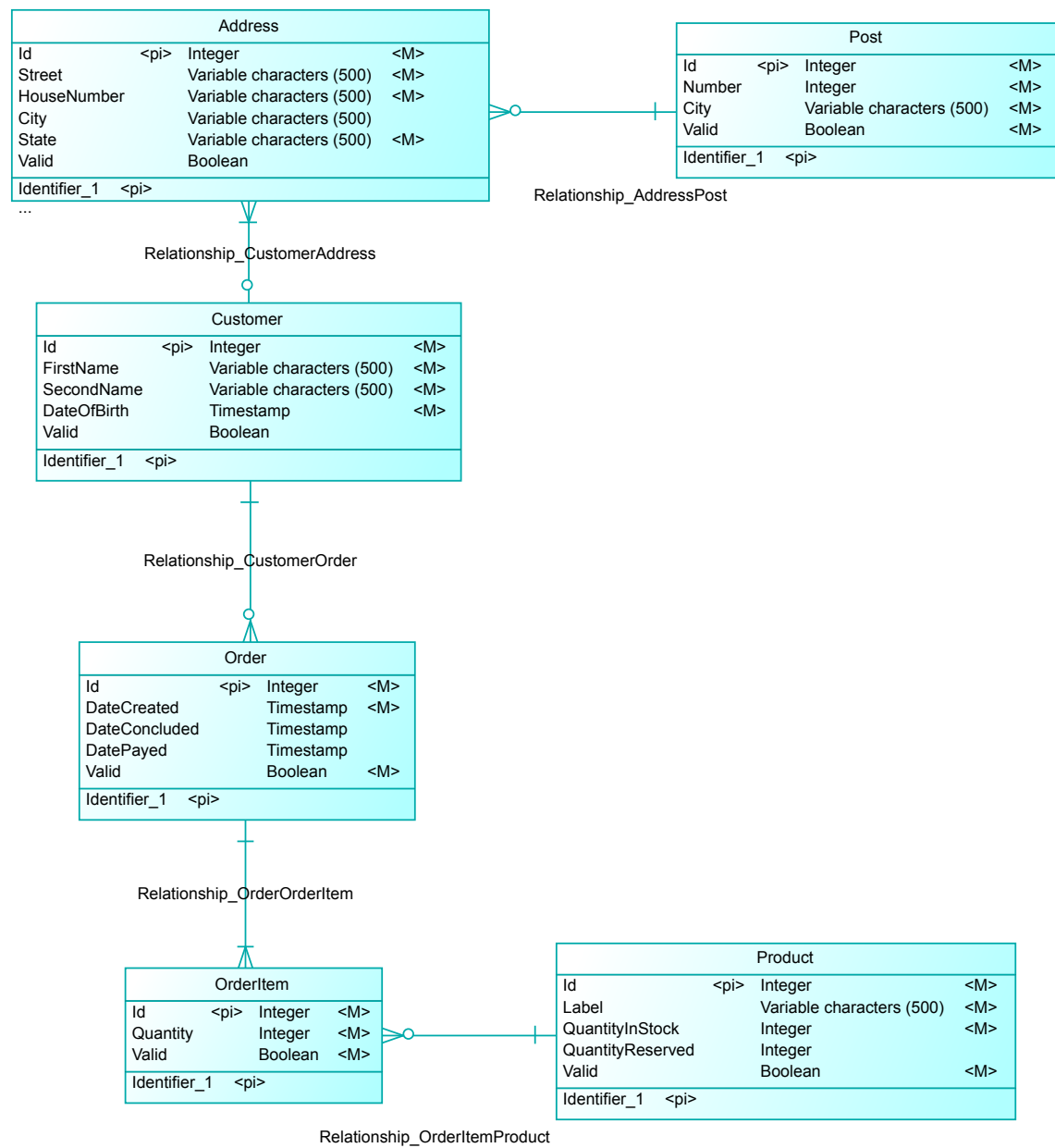
Na osnovi načrta podatkovne baze je bila realizirana referenčna implementacija, ki je nato služila kot osnova za integracijo aplikacije na portale.

Za referenčno implementacijo so bile izbrane tehnologije, ki so se uporabljale pri razvoju ogrodja Medius Vaadin Common:

- podatkovna baza MySQL (različica 5.1.44)
- aplikacijski strežnik JBoss (različica 5.0.1)
- Hibernate (implementacija JPA za ORM preslikavo)
- brskalnik Mozilla Firefox na strani odjemalca
- Eclipse 4.1 Indigo, Java EE (IDE)

#### 4.1.1 Podatkovni nivo

Na podatkovnem nivoju imamo podatkovni strežnik MySQL 5.1.44. S pomočjo orodja Sybase Power Designer je bil iz konceptualnega diagrama izdelan diagram na logičnem nivoju ter skripta za izdelavo podatkovne baze, kot jo prikazuje izvorna koda v prilogi A.1. Skripta je bila nato izvršena na podatkovnem strežniku ki je zgeneriral podatkovno bazo. Le-ta je bil poimenovana „Medius“.



Slika 4.1: Konceptualni nivo podatkovne baze

### 4.1.2 Poslovni in spletni nivo

Na poslovnem nivoju so bila definirana entitetna zrna, kot jih prikazujejo priloge A.2.1-A.2.6 v sklopu dodatka A.2. S stališča uporabnika so pomembni štrije pregledi entitetnih zrn v aplikaciji:

- Pregled šifrantna pošta (entitetno zrno „Post“)
- Pregled izdelkov (entitetno zrno „Product“)
- Pregled strank (entitetno zrno „Customer“)
- Pregled naročil (entitetno zrno „Order“)

Pregled poštnih števil in izdelkov je realiziran kot preprosta tabela s preprostimi vnosnimi maskami. Pregleda in urejanje strank in naročil je bolj kompleksno in je realizirano z anotacijami, kot so bile predstavljene v poglavju 3.2.3:

- entitetni zrno Customer in Address sta anotirani z anotacijami za razmerje ena-proti-mnogo (torej na maski za urejanje stranke, lahko vnesemo več naslovov za eno stranko)
- entitetni zrna Order in Customer sta anotirani za razmerje mnogo-proti-ena (na maski za vnos novega naročila, lahko izberemo le eno določeno stranko)
- entitetni zrna Order in Product sta anotirani za razmerje mnogo-proti-mnogo (pri vnosu postavk na maski za naročila lahko dodamo več izdelkov, pri čemer lahko nek izdelek nastopa na različnih naročilih), realizirano s pomočjo vmesne entitete OrderItem

Definirana so bila tudi vsa sejna zrna na podoben način kot jih opisuje poglavje 3.2.3.

Za povezavo med entitetnimi zrna in podatkovno bazo skrbi Hibernate, to je ogrodje, ki implementira specifikacijo JPA. Za delovanje tega ogrodja je potrebno projektu, ki definira entitetna in sejna zrna, dodati nastavitveno datoteko, kot jo prikazuje izvorna koda 4.1. V njej definiramo potrebne parametre za pravilno delovanje z MySQL podatkovno bazo in ustrezno ime podatkovnega vira na aplikacijskem strežniku, ki skrbi za povezavo do podatkovne baze.

Za nastavitev podatkovnega vira pa moramo v mapi „server/default/deploy“ dodati XML dokument z opisom vira. V njem definiramo naslov podatkovnega strežnika, uporabniško ime, geslo ter gonilnik, ki se uporablja

---

**Izvorna koda 4.1** Nastavitvena datoteka za persistence.xml za Hibernate

---

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/
xml/ns/persistence" xmlns:xsi="http://www.w3.org
/2001/XMLSchema-instance" xsi:schemaLocation="http://
java.sun.com/xml/ns/persistence http://java.sun.com/
xml/ns/persistence/persistence_2_0.xsd">
    <persistence-unit name="MediusTestEJB">

        <provider>org.hibernate.ejb.
            HibernatePersistence</provider>
        <jta-data-source>mediusDatasource</jta-
            data-source>
        <jar-file>/mysql-connector-java-5.0.8-
            bin.jar</jar-file>

        <properties>
            <property name="hibernate.
                dialect" value="org.hibernate
                .dialect.MySQLDialect" />
            <property name="hibernate.
                hbm2ddl.auto" value="update" /
            >
            <property name="hibernate.
                show_sql" value="true" />
            <property name="hibernate.
                format_sql" value="true" />
            <property name="hibernate.
                default_catalog" value="
                medius" />
        </properties>

    </persistence-unit>
</persistence>
```

---

---

**Izvorna koda 4.2** Nastavitvena datoteka za podatkovni vir do MySQL podatkovne baze

---

```
<!DOCTYPE datasources PUBLIC "-//JBoss//DTD_JBOSS_JCA_
    Config_1.5//EN" "http://www.jboss.org/j2ee/dtd/jboss-
    ds_1_5.dtd">
<datasources>
    <local-tx-datasource>
        <jndi-name>
            mediusDatasource
        </jndi-name>
        <use-java-context>
            false
        </use-java-context>
        <connection-url>
            jdbc:mysql://localhost:3306/
            medius
        </connection-url>
        <driver-class>
            com.mysql.jdbc.Driver
        </driver-class>
        <user-name>
            medius
        </user-name>
        <password>
            medius
        </password>
    </local-tx-datasource>
</datasources>
```

---

za dostop do podatkovne baze. Posebej pomemben je še naziv povezave („jndi-name“), ki določi pod katerim imenom lahko Hibernate najde podatkovni vir na aplikacijskem strežniku (ta naziv je nato kot paramater „jta-data-source“ podan v persistence.xml datoteki). Primer takega dokumenta prikazuje izvorna koda 4.2. S tem je zaključena implementacija strežniškega dela aplikacije.

Spletni del aplikacije definira poglede (ang. „View“), na katere so dodane posamezne maske za pregled entitetnih zrn. Primer takega pogleda vsebuje

---

**Izvorna koda 4.3** Primer definicije pogleda na spletnem delu aplikacije.

---

```
public class PostView extends XAPage{

    public static final String ID = "PostView";
    public static final String TITLE = "PostView";

    public PostView() {
        super(ID, TITLE, false);
    }

    @Override
    protected void createContents() {
        setWidth("400px");
        setHeight("100%");
        XATableView<Post> tableView = new
            XATableView<Post>(Post.class,
                PostManagerLocal.class);
        tableView.setWidth("400px");
        addComponent(tableView);
        System.out.println("Test2");
    }
}
```

---

izvorna koda 4.3. Vsi ostali pogledi so definirani na enak način, le da vsebujejo ustrezno referenco na drugo entitetno zrno.

Poslovni in spletni nivo tečeta na aplikacijskem strežniku JBoss.

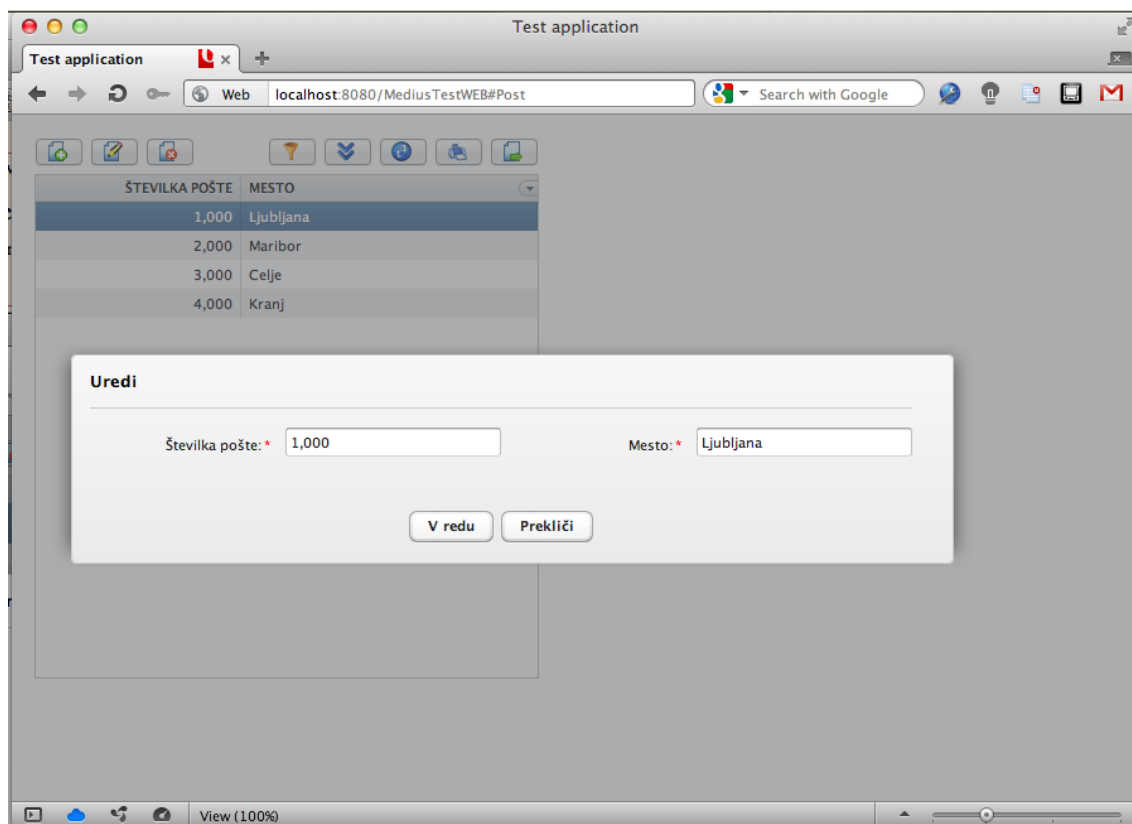
### 4.1.3 Nivo odjemalca

Na strani odjemalca imamo spletni brskalnik (podprti so Mozilla Firefox, Google Chrome in Microsoft Internet Explorer), preko katerega dostopa uporabnik do aplikacije.

Slika 4.2 prikazuje primer pogleda za entitetno zrno Post (v ospredju je vnosna maska, v ozadju pa osnovna tabela entitet).

Slika 4.3 prikazuje primer pogleda za entitetno zrno Product (v ospredju je vnosna maska, v ozadju pa osnovna tabela entitet).

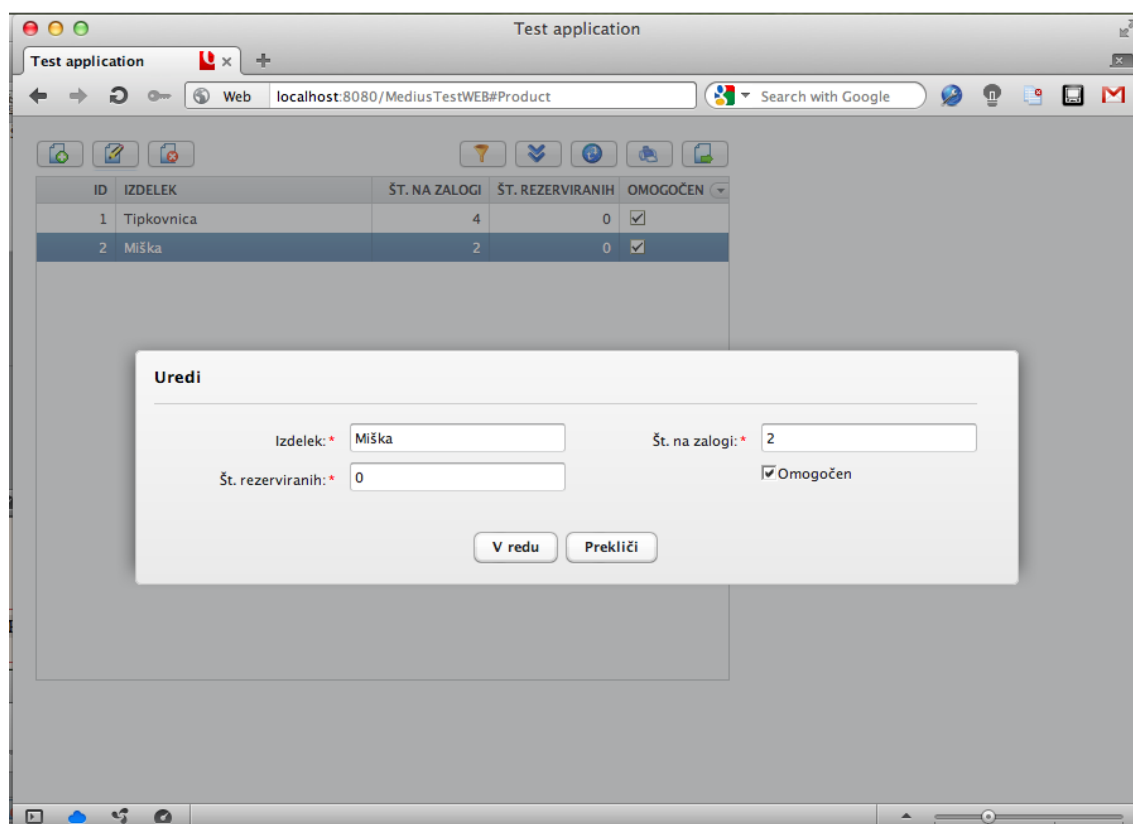
Slika 4.4 prikazuje primer pogleda za entitetno zrno Customer (stranka). V



Slika 4.2: Primer pogleda za entitetno zrno Post.

ozadju je osnovna tabela entitet, nad njo je vnosna maska za urejanje stranke (do nje pridemo s klikom na gumb „Uredi“), povsem v ospredju pa je maska za dodajanje naslova v seznam naslovov na maski za urejanje stranke. Na maski za dodajanje naslova je viden tudi „LookupSimple“ za izbiro ene same pošte.

Slika 4.5 prikazuje primer pogleda za entitetno zrno Order (naročilo). V ozadju je osnovna tabela entitet, nad njo je vnosna maska za urejanje naročila (do nje pridemo s klikom na gumb „Uredi“), povsem v ospredju pa je maska za izbiro izdelkov, ki jih želimo kot postavke v naročilu. Za razliko od vnosne maske Customer, kjer smo lahko kar dodajali nove naslove, lahko pri določanju postavk le izbiramo med obstoječimi izdelki. Na maski za urejanje naročila je viden tudi „LookupSimple“ za izbiro ene same stranke, kateri pripada naročilo.



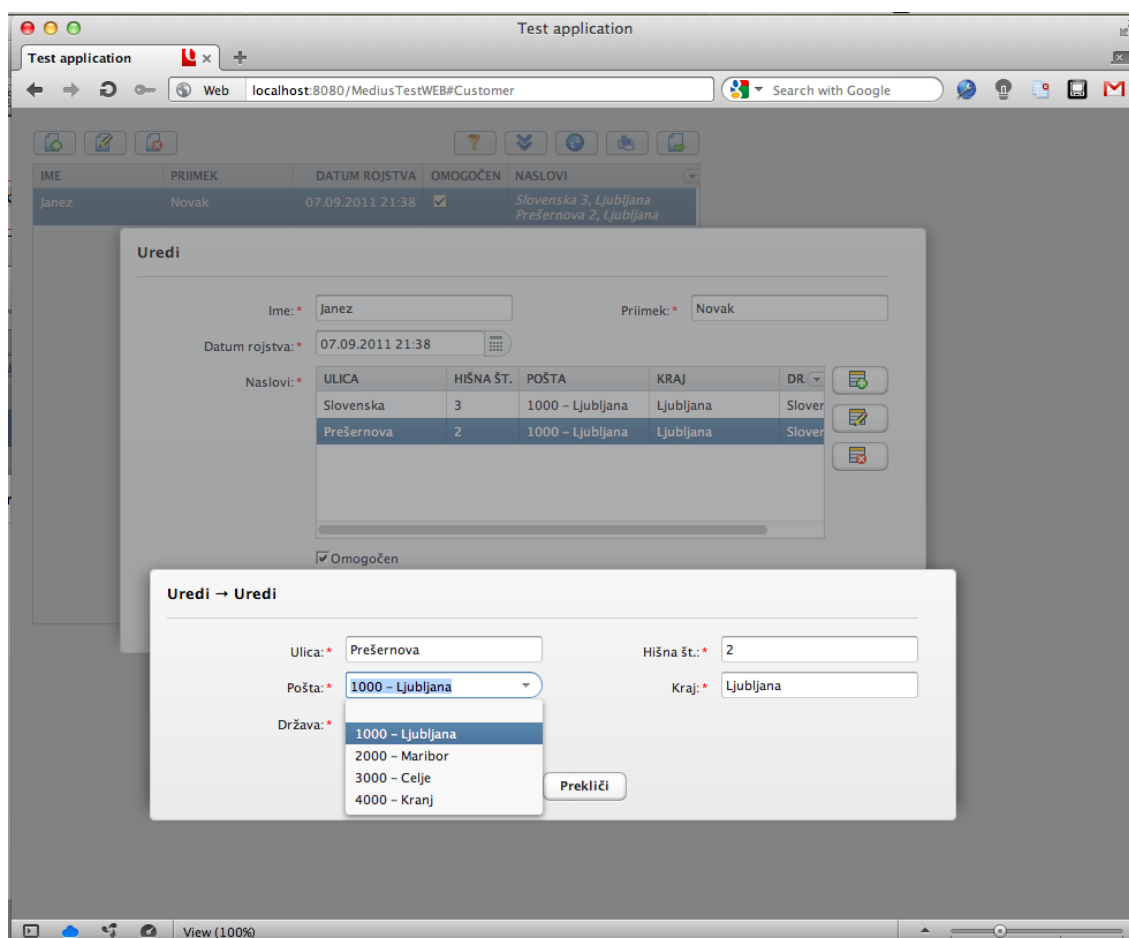
Slika 4.3: Primer pogleda za entitetno zrno Product.

## 4.2 Analiza zahtev za oceno uspeha integracije

Ponudniki poslovnih portalov običajno ob nakupu portala ne ponujajo le portala samega, temveč celoten ekosistem, katerega del je tudi portal. To pomeni, da nabor aplikacijskih sistemov, ki jih dobimo ob nakupu, vsebuje:

- podatkovno bazo
- aplikacijski strežnik
- portal
- dodatne aplikacije za razvoj (pri nekaterih ponudnikih jih je potrebno posebej dokupiti)

Ker je temu tako, se zdi smiselno, da poskus integracije razdelimo po posameznih arhitekturnih nivojih:

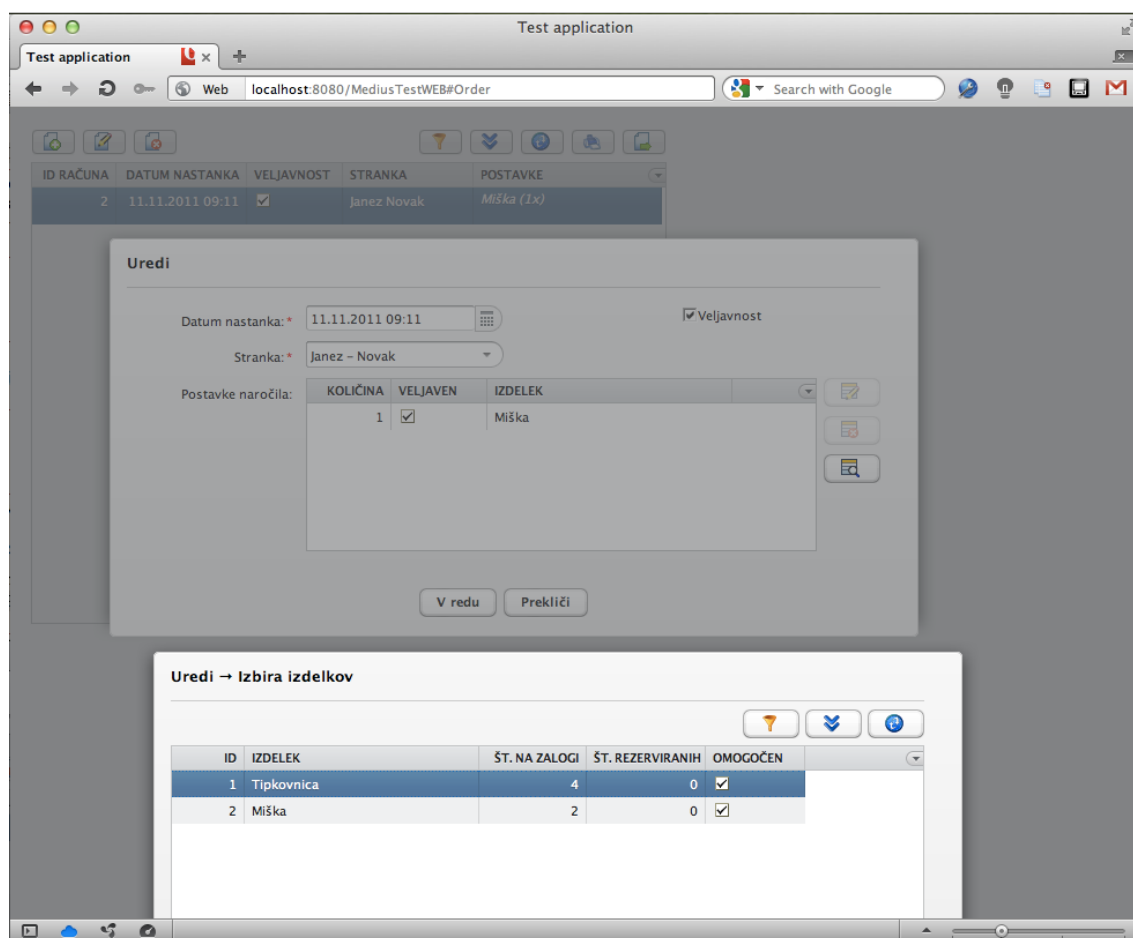


Slika 4.4: Primer pogleda za entitetno zrno Customer.

- Podatkovni nivo

Na podatkovnem nivoju v orodju Power Designer iz konceptualne sheme podatkovne baze generiramo shemo na logičnem nivoju glede na ciljno podatkovno bazo. Nato preverimo potrebne prilagoditve, ki jih zahteva uporabljena podatkovna baza ter generiramo skripto za izdelavo podatkovne baze. Skripto izvedemo na podatkovnem strežniku in tako generiramo podatkovno bazo.

Kriterij za uspešno oceno integracije je torej, da je mogoče realizirati podatkovno bazo, kot smo jo definirali v prejšnjih poglavjih. Pozornost na tem nivoju posvetimo količini dela, ki je potrebno, da s privzete namestitve podatkovnega strežnika pridemo do delujoče podatkovne baze.



Slika 4.5: Primer pogleda za entitetno zrno Order.

- Poslovni in spletni nivo  
Na poslovnem nivoju moramo poskrbeti za izbiro ustrezne implementacije JPA, s pomočjo katere lahko povežemo podatkovno bazo z našo aplikacijo. Definirati moramo ustrezne podakovne vire na aplikacijskem strežniku, ki služijo za osnovno povezavo, ki jo uporablja ponudnik JPA. Na koncu preverimo, ali so potrebne prilagoditve v strežniški kodi aplikacije (pri definiciji entitetnih ter sejnih zrn) in kodi v pogledih na spletnem delu aplikacije. Posebno pozornost namenimo potrebnim spremembam, da poglede na spletnem delu aplikacije prikažemo v ustreznih portletih na portalu.  
Kriteriji za uspešno oceno integracije na tem nivoju so:

- delujoči podatkovni viri na aplikacijskem strežniku
  - delujoča implementacija JPA (dodajanje, urejanje, brisanje podatkov v samo podatkovno bazo ter pridobivanje podatkov iz baze)
  - količina potrebnih sprememb na strežniški kodi aplikacije
  - količina potrebnih sprememb na spletnem delu aplikacije
  - ustrezne nastavitve za prikaz pogledov v portletih na portalu
  - kompleksnost vzpostavitve osnovnega portala
- Nivo odjemalca  
Na nivoju odjemalca preverimo delovanje aplikacije v naslednjih točkah:
    - ustrezen prikaz osnovne aplikacije, vključno z vsemi grafičnimi elementi (slike, barve, preliv, komponente grafičnega vmesnika...)
    - delovanje dodajanja, urejanja, brisanja entitet na uporabniških maskah (vključno z vsemi razmerji do ostalih entitetnih zrn)
    - delovanje sortiranja, filtriranja v tabelah
    - delovanje izvoza v datoteke PDF in Microsoft Excel
    - pravilen prikaz in delovanje portleta pri različnih stanjih okna (minimizacija, maksimizacija in klasičen prikaz)

Da lahko neko integracijo v celoti označimo kot zadovoljivo, mora veljati, da je zadovoljiva pri vseh navedenih kriterijih. Dodatno je pri vsaki implementaciji kratko predstavljena tudi postavitev osnovnega okolja v kateri poteka poskus integracije.

## 4.3 Integracija na Liferay Portalu®

Prvi poskus integracije bo potekal na poslovnemu portalu Liferay. Izbran je bil, ker ga je mogoče uporabljati na aplikacijskem strežniku JBoss, ki je bil osnova za razvoj ogrodja Medius Vaadin Common, poleg tega pa je mogoče dobiti tudi prosto dostopno različico.

### 4.3.1 Analiza podprtih standardov in priprava okolja

Izbrali bomo različico Liferay portala, ki teče na aplikacijskem strežniku JBoss, posledično ne pričakujemo nobenih težav vezanih na aplikacijski strežnik, saj je le-ta enak, kot je bil pri referenčni implementaciji. Prav tako bomo izbrali enako podatkovno bazo MySQL in enakega ponudnika JPA, t.j. Hibernate. Izkaže se tudi, da Liferay privzeto podpira standarde JSR-168, JSR-268 in Java EE okolje [8]. V primeru našega ogrodja Medius Vaadin Common pa je še dodatna prednost, da so razvijalci ogrodja Vaadin, na katerem gradi naše ogrodje, že poskrbeli za primer integracije Vaadin aplikacije kot portleta na portal in celo izdelali dodatek za IDE Eclipse, ki omogoča enostaven razvoj portletov za portal Liferay.

Konfiguracija okolja za integracijo je naslednja:

- operacijski sistem Mac OS X 10.7.2 (Lion; uspešno preizkušeno je bilo tudi delovanje na operacijskem sistemu Linux)
- aplikacijski strežnik JBoss 5.1.0 [28]
- podatkovna baza MySQL 5.1.44
- Liferay 6.0.4 [28]
- Eclipse 4.1 Indigo IDE (različica Java EE)

Vse kar je potrebno narediti za osnovno delovanje je, da s spletne strani prenesemo stisnjeno datoteko, v kateri je kompletan aplikacijski strežnik z že nameščenim portalom. Datoteko razširimo v mapo in v IDE okolju Eclipse ustvarimo nov strežnik JBoss ter ga zaženemo. Ko se strežnik uspešno zažene, avtomatsko odpre portal v privzetem brskalniku.

Za namestitev podatkovne baze je bil uporabljen XAMPP, to je distribucija Apache aplikacij (MySQL, PHP in Pearl) od katerih je bil uporabljen samo strežnik MySQL.

---

**Izvorna koda 4.4** Napaka ob prvem poskusu definiranja podatkovnega vira na JBoss strežniku

---

```
15:08:11,934 ERROR [ProfileServiceBootstrap] Failed to
load profile: Summary of incomplete deployments (SEE
PREVIOUS ERRORS FOR DETAILS):
```

DEPLOYMENTS MISSING DEPENDENCIES:

```
Deployment "jboss.jca:name=mediusDatasource,service=
DataSourceBinding" is missing the following
dependencies:
```

```
Dependency "jboss:service=invoker,type=jrmp" (should
be in state "Create", but is actually in state
"** NOT FOUND Depends on 'jboss:service=invoker,
type=jrmp' **")
```

DEPLOYMENTS IN ERROR:

```
Deployment "jboss:service=invoker,type=jrmp" is in
error due to the following reason(s): ** NOT FOUND
Depends on 'jboss:service=invoker,type=jrmp' **
```

---

## 4.3.2 Poskus integracije

### 4.3.2.1 Podatkovni nivo

Na podatkovnem nivoju je bila v orodju Power Designer najprej iz konceptualne sheme podatkovne baze kreirana shema na logični ravni. Izkazalo se je, da posebne prilagoditve za MySQL podatkovno bazo niso potrebne. Na podatkovnem strežniku je bila ustvarjena prazna podatkovna baza z imenom „Medius“, nato pa je bila s pomočjo skripte ustvarjena še sama struktura podatkovne baze. Ustvariti je bilo potrebno še ustreznega uporabnika „medius“ in mu dodeliti ustrezne pravice za dostop.

### 4.3.2.2 Poslovni nivo

Prvi korak na poslovnem nivoju je kreiranje podatkovnega vira na aplikacijskem strežniku, kar dosežemo na enak način kot pri referenčni implementaciji z ustrezno XML datoteko, ki jo prikazuje izvorna koda 4.2. Ko je bila dodana ta datoteka in ponovno zagnan strežnik je takoj prišlo do napake, ki jo kaže izvorna koda 4.4.

---

**Izvorna koda 4.5** Sporočilo na standardnem izhodu strežnika JBoss ki pove, da je bil ustvarjen podatkovni vir do podatkovne baze z imenom „mediusDatasource“.

---

```
15:47:47,751 INFO [WrapperDataSourceService] Bound
ConnectionManager 'jboss.jca:service=
DataSourceBinding,name=mediusDatasource' to JNDI name
'mediusDatasource'
```

---

Izkaže se, da je distribucija strežnika JBoss, na katerem je nameščen Liferay portal, nekoliko okrnjena in ne podpira specifikacije EJB 3.0. Če primerjamo to distribucijo s privzeto distribucijo, ki jo dobimo na spletni strani JBoss-a, ugotovimo, da nekatere datoteke na Liferay-evi distribuciji manjkajo. S primerjavo obeh distribucij lahko določimo manjkajoče datoteke in jih dodamo distribuciji Liferay (paziti moramo, da primerjamo vedno enaki različici strežnika JBoss). Gre za več datotek znotraj podmape „default“, kot jih prikazuje slika 4.6.

Manjkajoče datoteke dodamo distribuciji Liferay-a in ponovno zaženemo strežnik. Tokrat na standardnem izhodu strežnika v konzoli dobimo sporočilo o ustvarjenem podatkovnem viru z imenom „mediusDatasource“ (sporočilo prikazuje izvorna koda 4.5).

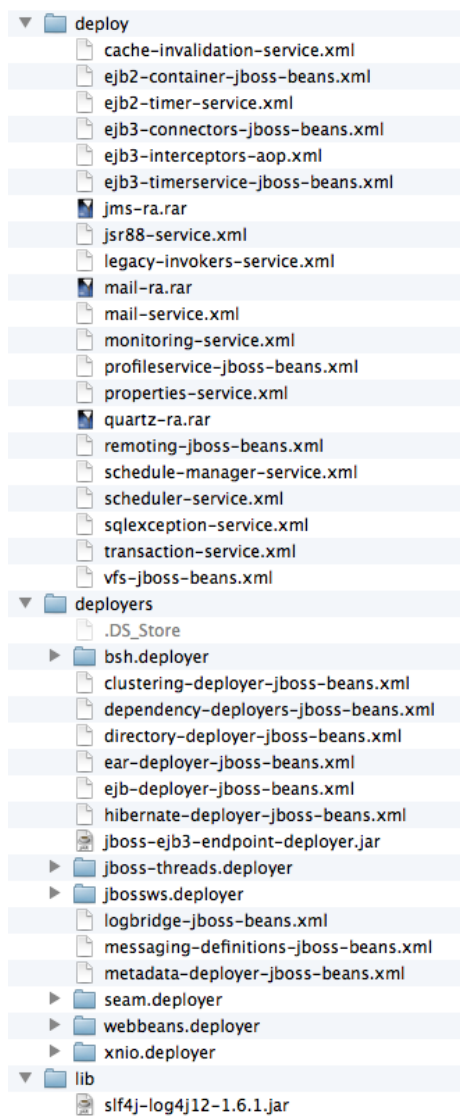
Po uspešni definiciji podatkovnega vira na aplikacijskem strežniku je potrebno realizirati strežniški del kode (entitetna in sejna zrna).

V ta namen v IDE okolju Eclipse ustvarimo nov projekt tipa „JPA Project“, ki ga poimenujemo npr. „MediusTestEJB“, pri nastavitvah pa določimo, da je ciljno izvajalno okolje (ang. „Target runtime“) JBoss različice 5.0. Dodatno še določimo različico JPA, ki jo nastavimo na 2.0 in obkljukamo možnost „Add project to an EAR“, pod ime pa nastavimo npr. „MediusTestEAR“ (nastavitve so prikazane na sliki 4.7). V zadnjem koraku čarovnika za izdelavo JPA projekta še izberemo možnost „Disable Library Configuration“ in v nastavitvi „Annotated classes must be listed in persistence.xml“. Ostale nastavitve pustimo na privzetih vrednostih. Po potrditvi bo Eclipse ustvaril prazen skelet projekta.

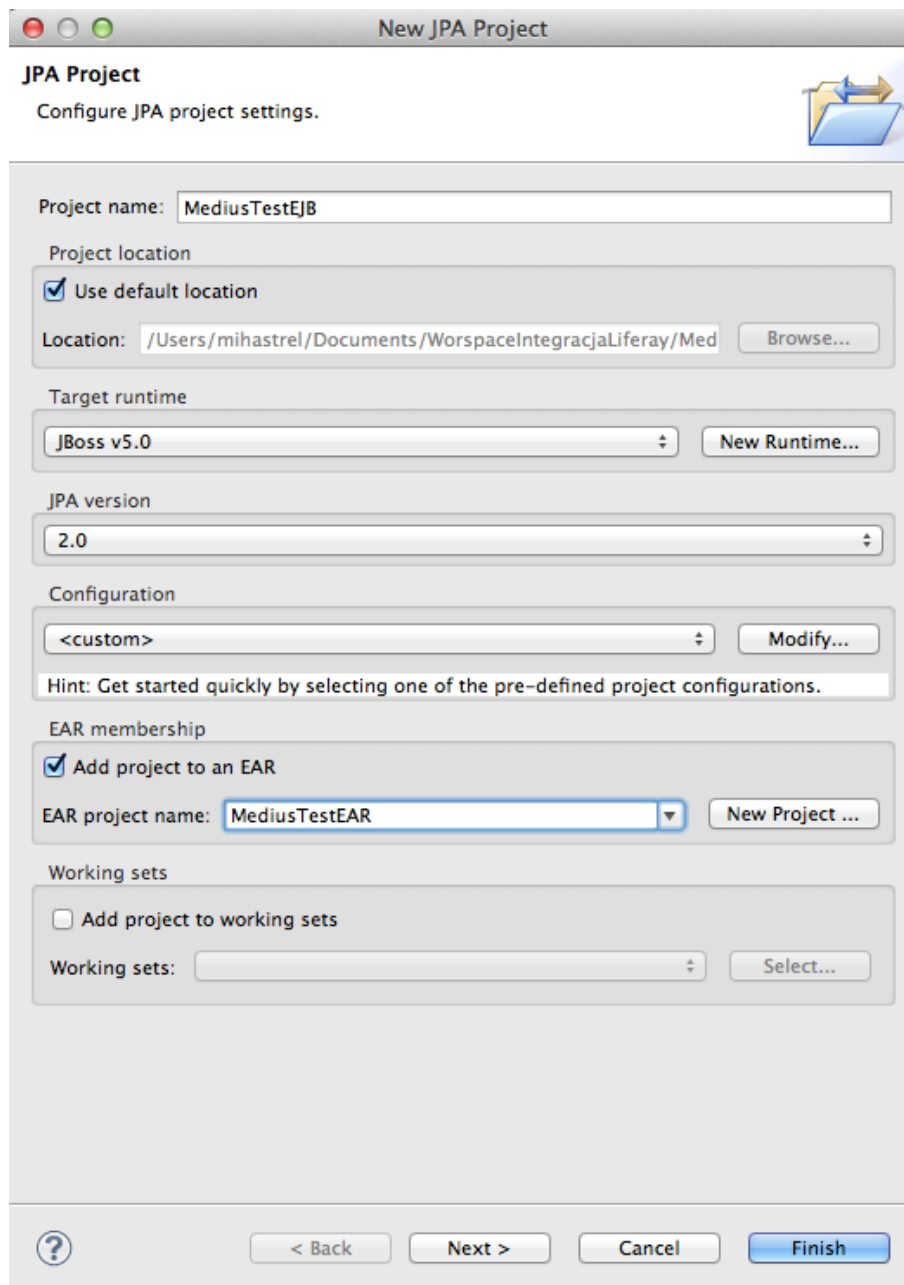
V projektu nato ustvarimo novo mapo za izvorno kodo poimenovano „src-ejb“, v katero dodamo entitetna in sejna zrna (za boljšo strukturiranost projekta uporabimo pakete) ter programske vmesnike strežniške kode.

Slika 4.8 prikazuje strukturo projekta MediusTestEJB.

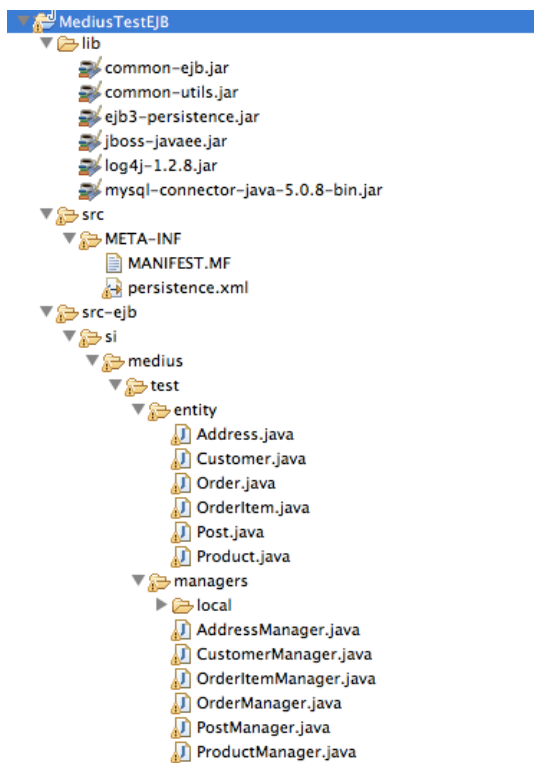
V mapi „src/META-INF“ se nahaja namestitvena datoteka persistence.xml, ki definira parametre za ponudnika JPA Hibernate. Ker smo v čarovniku na-



Slika 4.6: Manjkajoče datoteke znotraj „default“ mape na distribuciji JBoss, ki je osnova namestitve portala Liferay



Slika 4.7: Prikaz nastavitve za JPA projekt v razvojnem okolju Eclipse



Slika 4.8: Struktura projekta MediusTestEJB.

stavili eksplicitno navajanje anotiranih entitetnih zrn, mora namestitvena datoteka vsebovati tudi te reference. Eden izmed parametrov je tudi podatkovni vir, ki ga bo Hibernate uporabil za dostop do podatkovne baze. Kot vrednost tega parametra navedemo ime podatkovnega vira, ki smo ga predhodno namestili na aplikacijski strežnik. Dodatek B.1 prikazuje uporabljeno namestitveno datoteko.

V projekt dodamo še mapo „lib“, v katero dodamo knjižnice, ki so potrebne za delovanje ogrodja Medius Vaadin Common, poleg njih pa še knjižnico za povezavo do MySQL podatkovne baze (t.i. JDBC gonilnik), katere referenca je v namestitveni datoteki persistence.xml (brez gonilnika v mapi „lib“ bo Eclipse javljal napako na projektu). Eclipse vidi referenco v persistence.xml lokalno, torej zahteva prisotnost „jar“ datoteke gonilnika JDBC v mapi „lib“ projekta „MediusTestEJB“, vendar pa se bo ta projekt namestil preko EAR projekta, torej na strežniku korenski imenik ne bo v projektu „MediusTestEJB“, temveč v „MediusTestEAR“. V tem imeniku bo aplikacijski strežnik iskal referenco na gonilnik JDBC, zato mora imenik ob namestitvi vsebovati mapo „lib“ z gonil-

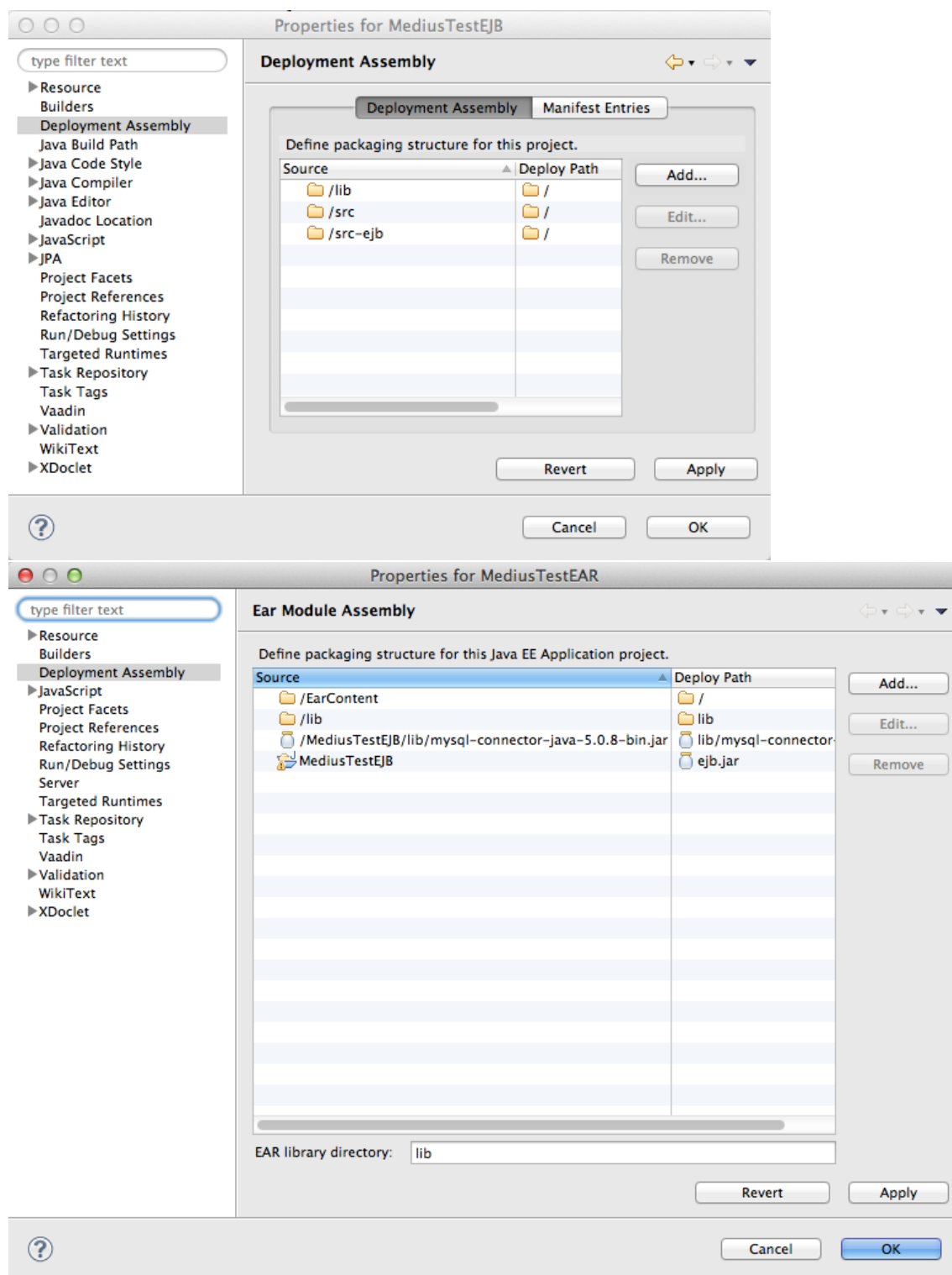
nikom JDBC. To bi lahko dosegli tako, da bi v projektu dodali mapo „lib“ in vanjo še enkrat skopirali JDBC gonilnik, vendar bi to pomenilo, da bi imeli isto datoteko na dveh različnih mestih v projektu (tega si ne želimo, saj bi npr. ob uvedbi nove različice gonilnika morali spreminjati datoteke na več mestih). Da bi se temu izognili, nastavimo namestitveni deskriptor projekta (najdemo ga v nastavitvah projekta pod „Deployment Assembly“) „MediusTestEAR“ tako, da ob nameščanju avtomatsko iz projekta „MediusTestEJB“ prekopira trenutno aktualno datoteko „jar“ JDBC gonilnika. To storimo tako, da v pogledu deskriptorja izberemo možnost „Add“, nato „Archives from Workspace“, ponovno „Add“ ter v strukturi poiščemo ustrezne „jar“ datoteke. Paziti moramo še, da deskriptor nastavimo tako, da te datoteke kopira v ustrezno mapo „lib“.

Nastaviti je potrebno še preostale namestitvene deskriptorje, ki se uporabijo ob nameščanju na aplikacijski strežnik. Najprej v projektu „MediusTestEJB“ privzetim nastavitvam namestitvenega deskriptorja dodamo vnos, ki določi, da mora projekt vsebovati tudi mapo „lib“, ki smo jo naknadno dodali k projektu (mapo „src-ejb“ bo Eclipse dodal avtomatsko). Nato moramo dokončno nastaviti še namestitveni deskriptor projekta „MediusTestEAR“, ki bo vseboval projekt MediusTestEJB kot „jar“ datoteko s strežniško kodo (namestitev na aplikacijski strežnik poteka s pomočjo EAR stisnjene datoteke). V namestitvenem deskriptorju projekta „MediusTestEAR“ poiščemo zapis projekta „MediusTestEJB“ in na desni strani deskriptorja popravimo namestitveno pot iz vrednosti „lib/MediusTestEJB.jar“ v „ejb.jar“. S tem smo zaključili nastavitve strežniške kode.

Nato lahko poskusimo z namestitvijo strežniške kode na aplikacijski strežnik JBoss. To lahko storimo v Eclipsu tako, da obstoječemu strežniku (v pogledu „Java EE“) dodamo projekt „MediusTestEAR“ in izberemo možnost „Publish“ (če strežnik ne bo že sam začel namestitve). Eclipse bo nato generiral ustrezno EAR datoteko ter jo namestil na aplikacijskem strežniku JBoss. Na standardnem izhodu aplikacijskega strežnika nato lahko vidimo sporočila o nameščanju sejnih zrn, kot jih prikazuje izsek izvirne kode 4.6 (vidno je obvestilo o namestitvi sejnega zrna Address Manager).

#### 4.3.2.3 Spletni nivo

Po uspešni namestitvi strežniškega dela preostane še implementacija spletnega dela aplikacije. V referenčni implementaciji je bil spletni del realiziran tako, da je bil za vsako entitetno zrno definiran pogled (ang. „View“), na katerem je bila maska z osnovno predstavitvijo entitetnega zrna ter gumbi za dostop do vnosnih mask.



Slika 4.9: Namestitvena deskriptorja projekta MediusTestEJB (zgoraj) in MediusTestEAR (spodaj).

---

**Izvorna koda 4.6** Prikaz sporočil strežnika o uspešnem nameščanju sejnih zrn.

---

```

10:47:19,837 INFO [JBossASKernel] Created
      KernelDeployment for: ejb.jar
10:47:19,837 INFO [JBossASKernel] installing bean:
      jboss.j2ee:ear=MediusTestEAR.ear,jar=ejb.jar,name=
      AddressManager,service=EJB3
10:47:19,837 INFO [JBossASKernel] with dependencies:
10:47:19,837 INFO [JBossASKernel] and demands:
10:47:19,837 INFO [JBossASKernel] jboss.
      ejb:service=EJBTimerService
10:47:19,837 INFO [JBossASKernel] persistence.
      unit:unitName=MediusTestEAR.ear/ejb.jar#MediusTestEJB
10:47:19,837 INFO [JBossASKernel] and supplies:
10:47:19,837 INFO [JBossASKernel] Class:si.medius.
      test.managers.local.AddressManagerLocal
10:47:19,837 INFO [JBossASKernel]
      jndi:MediusTestEAR/AddressManager/remote
10:47:19,838 INFO [JBossASKernel]
      jndi:MediusTestEAR/AddressManager/local
10:47:19,838 INFO [JBossASKernel]
      jndi:MediusTestEAR/AddressManager/local-si.medius.
      test.managers.local.AddressManagerLocal
10:47:19,838 INFO [JBossASKernel] Added bean(jboss.
      j2ee:ear=MediusTestEAR.ear,jar=ejb.jar,name=
      AddressManager,service=EJB3) to KernelDeployment of:
      ejb.jar

```

. . . (namestitvena obvestila za vsako sejno zrno)

```

10:47:20,675 INFO [EJBContainer] STARTED EJB: si.medius
      .test.managers.AddressManager ejbName: AddressManager
10:47:20,681 INFO [JndiSessionRegistrarBase] Binding
      the following Entries in Global JNDI:

```

```

      MediusTestEAR/AddressManager/local - EJB3.x
      Default Local Business Interface
      MediusTestEAR/AddressManager/local-si.medius.
      test.managers.local.AddressManagerLocal -
      EJB3.x Local Business Interface

```

---

Na portalu bo predstavitev nekoliko drugačna. Za vsako entitetno zrno bomo definirali portlet, na katerem bo tabela z osnovno predstavitevijo entiteta zrna. Ta portlet bomo namestili na portal, ki ga bo prikazal v seznamu portletov, uporabnik pa si ga bo lahko namestil na svojo stran na portalu.

Ker so razvijalci Vaadina izdelali poseben vtičnik za generiranje portletov namenjenih Liferay portalu, bo izdelava osnovnega skeleta spletnega dela aplikacije enostavna. V Eclipsu namestimo vtičnik po navodilih razvijalcev [29] in zaženemo čarovnik za nov projekt tipa „Vaadin project“, ki ga poimenujemo npr. MediusTestWEB. Izvajalno okolje nastavimo na „JBoss 5.0“, parameter „configuration“ pa nastavimo na „Vaadin, Java 6, Servlet 2.4“. Ker hočemo aplikacijo na portal namestiti kot portlete, v možnostih za parameter „Deployment configuration“ izberemo „Generic portlet (Portlet 2.0)“, t. j. implementacijo po specifikaciji JSR-286. Različica Medius Vaadin Common ogrodja, ki jo uporabljamo za integracijo uporablja Vaadin različice 6.6.6 zato pri nastavitvah pazimo, da izberemo ustrezno različico Vaadina (sicer pride do konfliktov med nameščenimi knjižnicami). Čarovnika za kreiranje spletnega dela prikazuje slika 4.10.

Nato v naslednjih korakih sprejmemo privzete možnosti, dokler ne pridemo do zadnjega koraka. Tu pazimo, da je izbrana vrednost „Portlet 2.0“ pri parametru „Portlet version“. Ostale parametre lahko pustimo na privzetih vrednostih. Eclipse bo nato zgradil osnovni skelet aplikacije.

Eden izmed prvih korakov pri sami implementaciji je, da v spletni projekt dodamo vse potrebne knjižnice, ki jih aplikacija potrebuje za delovanje. V ta namen v korenskem imeniku definiramo mapo „lib“, kamor dodamo osnovne knjižnice (torej knjižnice, ki niso specifične le za spletni del). Knjižnice, ki so specifične za spletni del, pa damo v mapo „WebContent/WEB-INF/lib“. V nastavitvah projekta moramo nato še dodati reference na vse dodane knjižnice. Poleg tega moramo tudi definirati, da projekt „MediusTestWEB“ referencira projekt „MediusTestEJB“, da ima Eclipse dostop do definicij entitetnih zrn.

Naslednji korak je kopiranje tem, ki določijo videz naše aplikacije (teme vsebujejo CSS datoteke, slike za uporabniški vmesnik...). Teme skopiramo v mapo „WebContent/VAADIN/themes“. Dodatno moramo teme skopirati tudi na strežnik JBoss v mapo jboss-5.1.0/server/default/deploy/ROOT.war/html/VAADIN/themes. S tem smo pripravili osnovno okolje za spletno aplikacijo.

Sledi implementacija aplikacije. V mapi „src“ ustavarimo ustrezen spletni pogled za določeno entitetno zrno (primer za entitetno zrno „Post“ je v izvorni kodi 4.7), in osnovno aplikacijo, ki vsebuje ta pogled (primer uporabe pogleda „PostView“ prikazuje izvorna koda 4.8). V osnovni aplikaciji med drugim določimo tudi temo, ki jo aplikacija uporablja, v našem primeru je

**New Vaadin Project**

Create a Vaadin Dynamic Web project.

Project name:

Project location

☒ Use default location

Location:

Target runtime

Configuration

Vaadin project running on Java 6.0, Servlet 2.4 .

Vaadin

Deployment configuration:

Vaadin version:

☐ Use latest nightly build (in same branch)

Slika 4.10: Nastavitve za spletni del aplikacije.

---

**Izvorna koda 4.7** Primer spletnega pogleda za entitetno zrno „Post“.

---

```
public class PostView extends XAPage{

    public PostView() {
        super(ID, TITLE, false);
    }

    @Override
    protected void createContents() {
        setWidth("400px");
        setHeight("100%");
        XATableView<Post> tableView = new
            XATableView<Post>(Post.class,
                PostManagerLocal.class);
        tableView.setWidth("400px");
        addComponent(tableView);
    }
}
```

---

to tema z imenom „test“. Pomembna podrobnost pri aplikaciji je metoda `IManagerProvider`, ki definira dostop do entitetnega upravljalca. Ker za Liferay uporabljamo enak aplikacijski strežnik kot pri referenčni implementaciji, posebne prilagoditve za dostop do entitetnega upravljalca niso potrebne.

Ko smo definirali aplikacijo, moramo še poskrbeti, da Vaadin zgradi ustrezne komponente za uporabniški vmesnik (t.i. „widgetset“). Za to poskrbi vtičnik v orodju Eclipse, ki v mapi „WebContent/VAADIN/widgetsets“ zgenerira ustrezne komponente. Zgenerirane komponente moramo nato skopirati še v mapo „jboss-5.1.0/server/default/deploy/ROOT.war/html/VAADIN/widgetsets“ na aplikacijskem strežniku. Strežniku moramo tudi povedati, naj uporabi komponente, ki smo jih mi dodali. To storimo tako, da v mapi „jboss-5.1.0/server/default/deploy/ROOT.war/WEB-INF/classes“ ustvarimo datoteko „portal-ext.properties“, v katero zapišemo naslednji vrstici (velja za naš primer):

- `vaadin.resources.path=/html`
- `vaadin.widgetset=si.medius.test.portletApplications.widgetset.MediustestwebWidgetset`

Da bi lahko aplikacijo prikazali kot portlet na portalu Liferay, moramo še ustrezno definirati deskriptor „portlet.xml“. Izsek datoteke „portlet.xml“ za

---

**Izvorna koda 4.8** Primer osnovne aplikacije za entitetno zrno „Post“.
 

---

```

public class PostApplication extends
    MediusCommonApplication
{

    private TPTMultiView controller;

    @Override
    public void init ()
    {
        super.init ();
        TM.setDefaultLanguage ("sl");

        Window mainWindow = new Window ("Application_Post
            ");
        mainWindow.addStyleName ("medius");
        setTheme ("test");
        setMainWindow (mainWindow);

        controller = new TPTMultiView (true);
        controller.setSizeUndefined ();
        controller.setWidth ("100%");
        controller.setHeight ("440px");

        controller.addView ("Post", new PostView ());

        mainWindow.addComponent (controller);
        mainWindow.setHeight ("100%");
    }

    ...

    @Override
    protected IManagerProvider initManagerProvider ()
    {
        return new EJBManagerProvider ("
            MediusTestEAR");
    }
}

```

---

naš primer prikazuje izvorna koda 4.9. V njej definiramo potrebne parametre, kot jih definira standard JSR-286. V našem primeru so to naslednji parametri:

- opis in oznaka portleta
- programski razred, ki se kliče ob zagonu portleta (`com.vaadin.terminal.gwt.server.ApplicationPortlet2`)
- programski razred, ki predstavlja našo aplikacijo (`si.medius.test.portletApplications.PostApplication`)
- definiramo katere komponente naj se uporabijo (`si.medius.test.portletApplications.widgetset. MediustestwebWidgetset`)
- definiramo podprte načine delovanja portleta (naš portlet podpira le klasičen način delovanja „view“)
- dodaten opis portleta, ki je specifičen za Liferay

Ustrezno je potrebno nastaviti tudi datoteki „web.xml“ (primer prikazuje izvorna koda 4.10) v mapi „WebContent/WEB-INF/“ in „application.xml“ v mapi „WebContent/META-INF“ (primer prikazuje izvorna koda 4.11).

S tem smo zaključili osnovno implementacijo spletnega dela (v tem trenutku imamo definiran portlet le za eno entitetno zrno). Pred namestitvijo na strežnik moramo še dopolniti namestitveni deskriptor projekta „MediusTestWEB“, ki mu dodamo zapis za vključitev mape „lib“ v namestitveno datoteko (na enak način kot pri projektu „MediusTestEJB“). Celotno strukturo projekta v orodju Eclipse prikazuje slika 4.11.

Na koncu nam preostane še namestitev spletnega dela aplikacije.

V idealnem primeru bi si želeli, da bi lahko spletni del aplikacije dodali kot datoteko WAR (za distribucijo spletnih aplikacij se v Javi uporablja stisnjen format WAR) v deskriptor projekta „MediusTestEAR“. Ob kreiranju EAR datoteke bi nato Eclipse potegnil spletni del aplikacije v datoteko EAR in jo skupaj s strežniškim delom namestil na aplikacijski strežnik, portal Liferay pa bi med nameščanjem sam prepoznal portlete, definirane v datoteki „portlet.xml“. Izkaže se, da Eclipse sicer zna pravilno generirati datoteko EAR na ta način, vendar ob nameščanju na aplikacijski strežnik portal Liferay ne zna prepoznati portletov, ki so zapakirani znotraj EAR datoteke. Zato moramo namestitev spletnega dela izvesti drugače. „MediusTestEAR“ projekt pustimo pri prejšnjih nastavitvah in ga normalno namestimo na strežnik znotraj orodja Eclipse. Spletni del aplikacije (torej projekt „MediusTestWAR“)

---

**Izvorna koda 4.9** Primer datoteke portlet.xml potrebne za namestitev portleta, ki prikazuje entitetno zrno „Post“, na portal Liferay.

---

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/
  portlet-app-2.0.xsd" xmlns:xsi="http://www.w3.org
  /2001/XMLSchema-instance" version="2.0"
  xsi:schemaLocation="http://java.sun.com/xml/ns/
  portlet/portlet-app-2.0.xsd http://java.sun.com/xml/
  ns/portlet/portlet-app-2.0.xsd">

  <portlet>
    <portlet-name>Mediustestweb Application portlet<
      /portlet-name>
    <display-name>MediusTestWEB-Post</display-name>

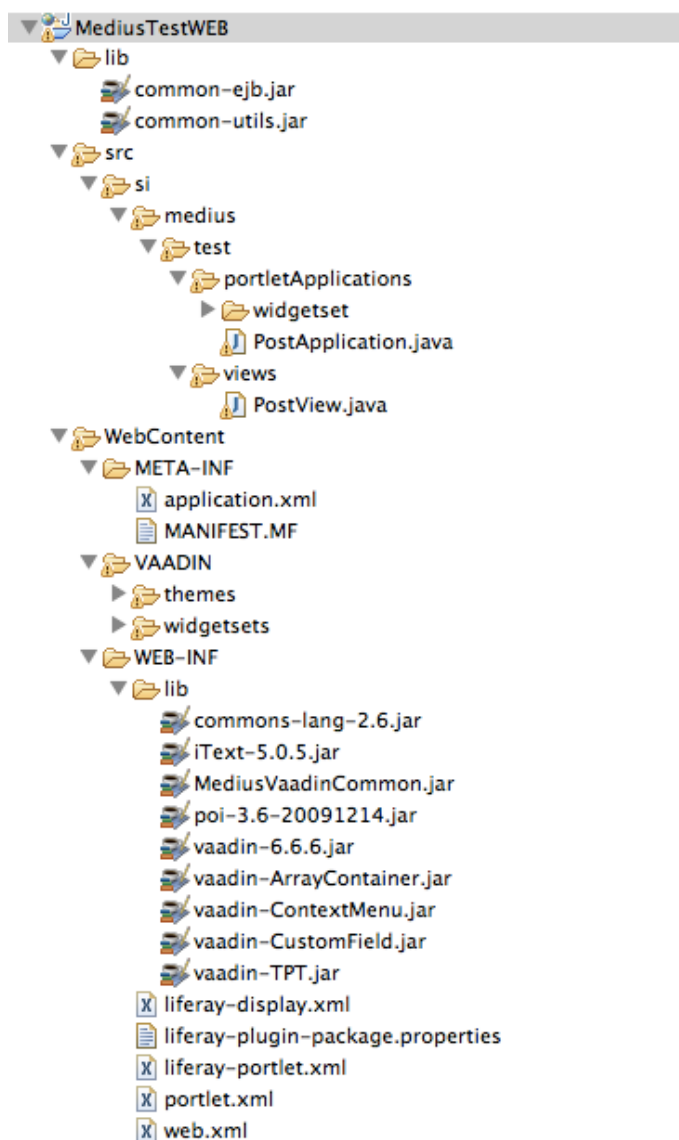
    <portlet-class>com.vaadin.terminal.gwt.server.
      ApplicationPortlet2</portlet-class>
    <init-param>
      <name>application</name>
      <value>si.medius.test.portletApplications.
        PostApplication</value>
    </init-param>

    <init-param>
      <name>widgetset</name>
      <value>si.medius.test.portletApplications.
        widgetset.MediustestwebWidgetset</value>
    </init-param>

    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>view</portlet-mode>
    </supports>

    <portlet-info>
      <title>MediusTestWEB-Post</title>
      <short-title>MediusTestWEB-Post</short-title>
    >
  </portlet>
</portlet-app>
```

---



Slika 4.11: Struktura spletnega projekta „MediusTestWEB“.

---

**Izvorna koda 4.10** Primer datoteke „web.xml“ spletnega dela aplikacije.

---

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java
.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org
/2001/XMLSchema-instance" xsi:schemaLocation="http://
java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/
j2ee/web-app_2_4.xsd">
    <display-name>MediusTestWEB</display-name>

    <context-param>
        <description>
            Vaadin production mode</description>
        <param-name>productionMode</param-name>
        <param-value>false</param-value>
    </context-param>

    <listener>
        <listener-class>com.liferay.portal.
            kernel.servlet.PortletContextListener
        </listener-class>
    </listener>
</web-app>

```

---

pa izvozimo iz Eclipse kot stisnjeno datoteko tipa „WAR“ in jo prekopiramo v mapo „liferay-portal-6.0.5/deploy“ (to je mapa, v kateri Liferay išče datoteke za namestitve). Liferay bo začel z nameščanjem spletne aplikacije, kar bo vidno na standardnem izhodu aplikacijskega strežnika v orodju Eclipse, kjer se bodo izpisala obvestila o nameščanju portleta (primer prikazuje izvorna koda 4.12).

Ko bo Liferay končal z nameščanjem portletov, le-ti še ne bodo vidni na uporabniškem vmesniku portala, saj jih mora na samo stran dodati uporabnik.

Na portalu Liferay je postopek naslednji:

- uporabnik se prijavi v sistem (na preizkusni različici portala Liferay lahko uporabimo avtomatsko prijavo uporabnika „Bruno“, ki je na voljo na vstopni strani portala)
- uporabnik kreira novo stran na portalu (kreiranje prikazuje slika 4.12)

---

**Izvorna koda 4.11** Primer datoteke „application.xml“ splenega dela aplikacije.

---

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema
-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:application="http://java.sun.com/xml/ns
/javaee/application_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/
javaee_http://java.sun.com/xml/ns/javaee/
application_5.xsd"
    id="Application_ID" version="5">
<display-name>MediusTestWEB</display-name>
<module>
    <ejb>ejb.jar</ejb>
</module>
<module>
    <web>
        <web-uri>MediusTestWEB.war</web-
uri>
        <context-root>MediusTestWEB</
context-root>
    </web>
</module>
</application>
```

---

ter se premakne nanjo

- na novoustvarjeno stran doda portlet (V zgornjem administracijskem meniju pritisnemo „Add“, nato „More...“. Prikaže se seznam kategorij, kjer poiščemo kategorijo Vaadin, znotraj katere je portlet. Za dodajanje pritisnemo na gumb „Add“ poleg naziva portleta.)
- portlet se pojavi na strani portala

Slika 4.14 prikazuje portlet, narejen z ogrodjem Medius Vaadin Common na portalu Liferay, kot smo ga definirali v spletnem delu aplikacije. Pri samem izgledu takoj opazimo, da na zgornjem delu manjkajo določeni gumbi, oz. da je celoten portlet nekoliko odrezan na desni strani. Za to obstajata dva razloga:

---

**Izvorna koda 4.12** Obvestila aplikacijskega strežnika ob nameščanju portletov.

---

```
08:40:59,381 INFO [TomcatDeployment] deploy, ctxPath=/
MediusTestWEB
08:41:00,249 INFO [PortletHotDeployListener]
Registering portlets for MediusTestWEB
08:41:00,332 WARNING [AbstractApplicationPortlet]
```

---

Vaadin is running in DEBUG MODE.

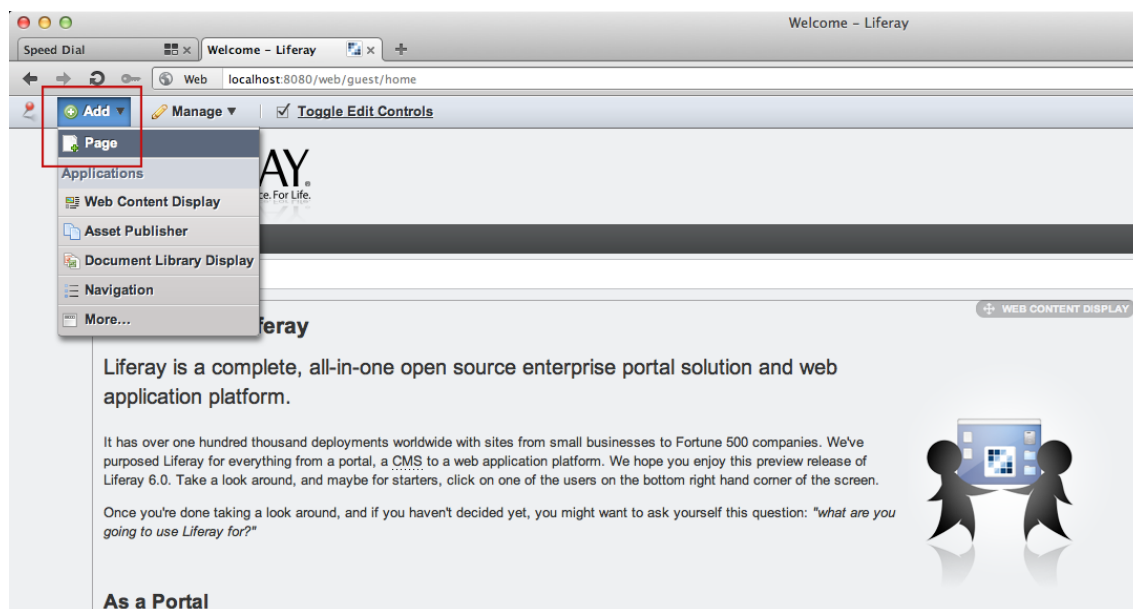
Add `productionMode=true` to `web.xml` to disable debug features.

To show debug window, add `?debug` to your application URL

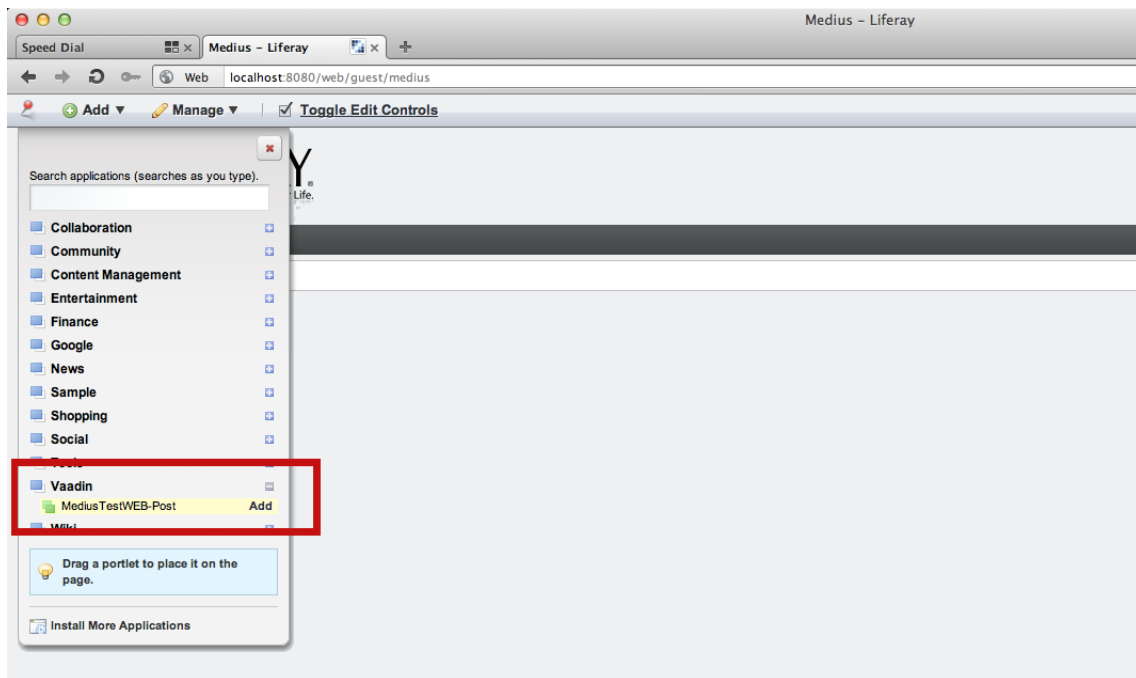
---

```
08:41:00,350 INFO [PortletHotDeployListener] 1 portlet
for MediusTestWEB is available for use
```

---



Slika 4.12: Dodajanje nove strani na portalu Liferay.



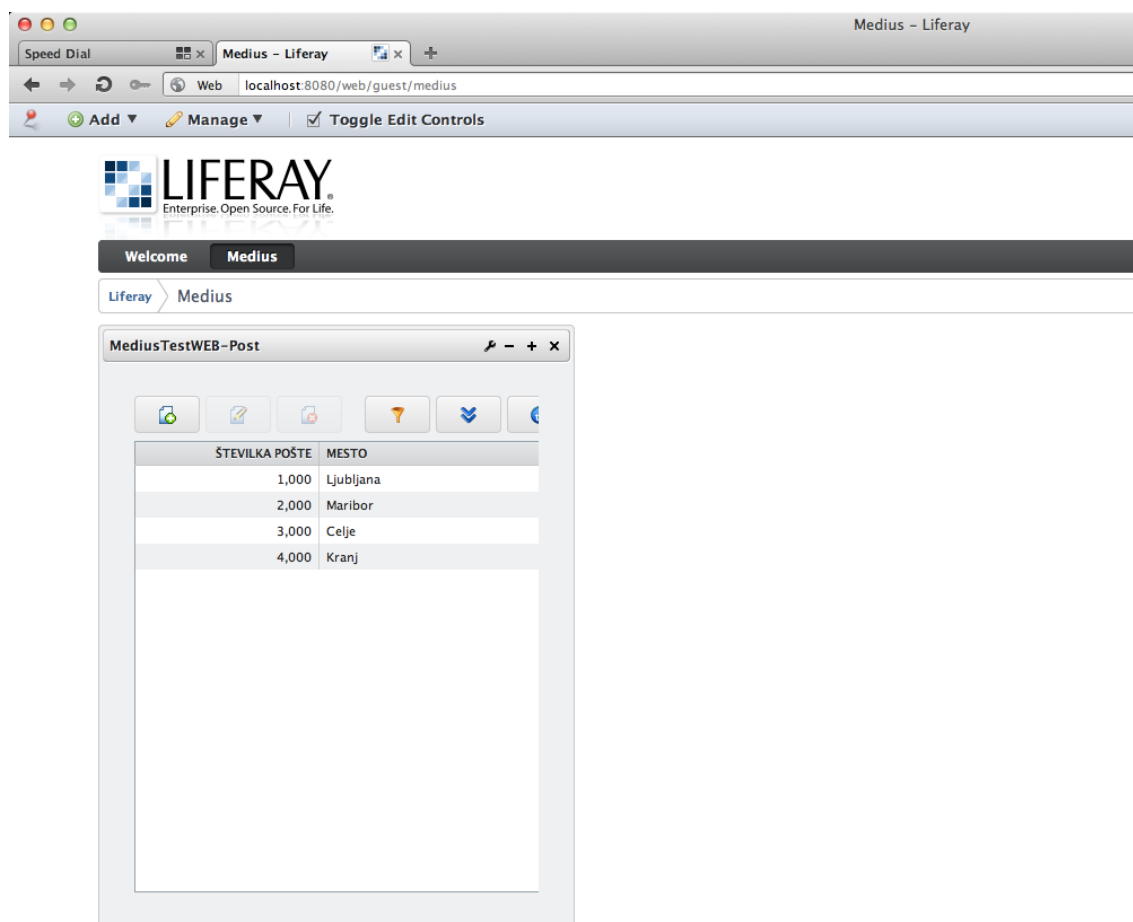
Slika 4.13: Dodajanje portleta na stran portala Liferay.

- V pogledu entitetnega zrna „Post“ smo definirali, da je vsebina pogleda široka 400 slikovnih točk, kar pa je premalo za tako velike gumbе. To spremenimo z nastavitvijo širine na npr. 600 slikovnih točk, dodatno pa moramo še povečati širino tabele (za uveljavljanje sprememb je potrebno ponovno nameščanje spletnega dela aplikacije). Oboje skupaj dosežemo v naslednjih vrsticah kode v razredu „PostView“ spletnega projekta „MediusTestWEB“:

```
setWidth("600px"); //širina celotne vsebine
tableView.setWidth("600px"); //širina tabele
```

- Stran na portalu, ki smo jo na novo naredili, je horizontalno privzeto razdeljena na levi in desni del, pri čimer je levi del bistveno ožji in ima premalo prostora za prikaz portleta. To rešimo tako, da s kazalcem miške primemo portlet na njegovem naslovu in ga premaknemo na desno stran ter osvežimo stran v brskalniku. Alternativna rešitev je, da razporeditev strani nastavimo na en sam stolpec.

Ko vnesemo vse spremembe, dobimo prikaz portleta, kot je na sliki 4.15. To-

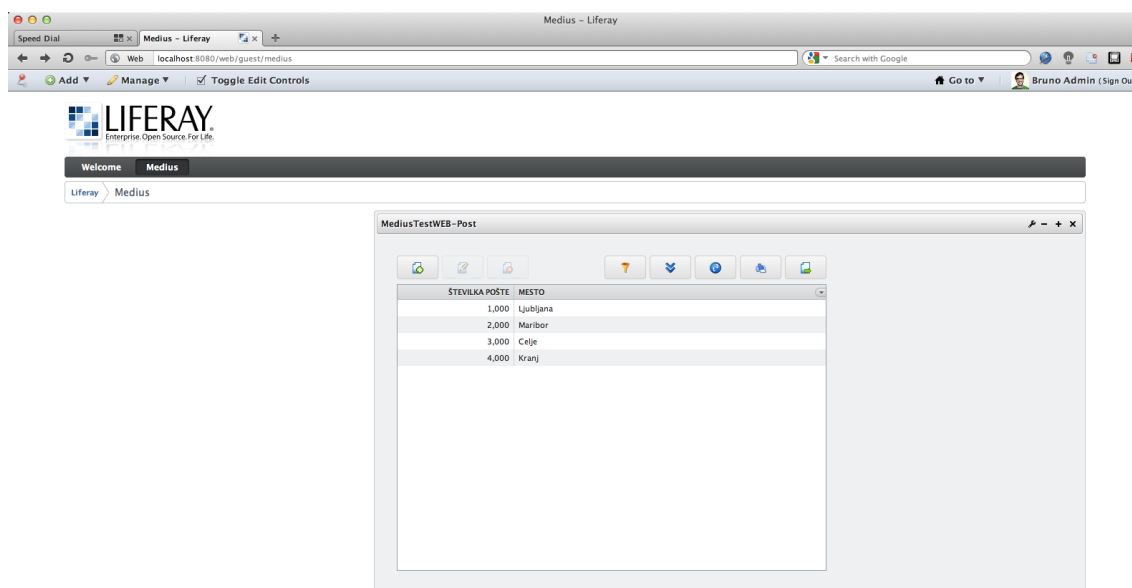


Slika 4.14: Portlet, ki prikazuje entitetno zrno „Post“ na portalu Liferay.

krat je viden celoten portlet z vsemi gumbi. S tem smo zaključili implementacijo prikaza entitetnega zrna „Post“.

Za dodajanje portletov, ki prikazujejo preostala entitetna zrna, dodamo ustrezne tabele in poglede v spletno aplikacijo in dopolnimo deskriptor „portlet.xml“ z ustreznimi vnosi (v eni datoteki „portlet.xml“ imamo lahko vnose za več portletov), posebej moramo paziti, da vsi portleti na portalu uporabljajo enake komponente (t.i. „widgetset“). Dodatno je potrebno dopolniti tudi deskriptorja „liferay-portlet.xml“ in „liferay-display.xml“, ki ju prej nismo posebej omenjali, saj jih je zgeneriral vtičnik za delo z Vaadinom v orodju Eclipse.

Deskriptor „liferay-portlet.xml“ vsebuje zapise, ki povežejo definicije por-



Slika 4.15: Prikaz portleta po vnosu popravkov.

tletov iz deskriptorja „portlet.xml“ in jim določijo še dodatne parametre specifične za portal Liferay. Vse parametre lahko pustimo na privzetih vrednostih, kot jih generira vtičnik, dodamo le zapis za nov portlet, katerega parameter „portlet-name“ nastavimo na vrednost, ki jo ima portlet v deskriptorju „portlet.xml“. Primer z dvema definiranimima portletoma prikazuje izvorna koda 4.13.

Deskriptor „liferay-display“ pa določi, kako bodo reference nameščenih portletov prikazane v meniju portala Liferay (torej v seznamu, kjer uporabnik vidi portlete, ki so na voljo; primer je na sliki 4.13). Izvorna koda 4.14 prikazuje definicijo, ki določa, da se bo kategorija portletov imenovala „Vaadin“, znotraj nje pa bo mogoče izbirati med dvema portletoma, ki sta določena s parametrom „id“, ki ustreza parametru „portlet-name“ v deskriptorju „portlet.xml“.

Ker gre pri implementaciji prikaza za preostala entitetna zrna za enak postopek kot pri entitetnem zrnu „Post“, na tem mestu ne bomo opisovali vsake implementacije posebej.

### 4.3.3 Analiza delovanja

- Podatkovni nivo

Za pripravo in izvedbo skripte, ki je generirala podatkovno bazo, definirano na referenčnem primeru, so bile potrebne zanemarljivo majhne

---

**Izvorna koda 4.13** Primer deskriptorja „liferay-portlet.xml“.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE liferay-portlet-app PUBLIC "-//Liferay//DTD_
  Portlet_Application_4.3.0//EN" "http://www.liferay.
  com/dtd/liferay-portlet-app_4.3.0.dtd">
<liferay-portlet-app>
  <portlet>
    <!-- Matches definition in portlet.xml. -->
    <portlet-name>Mediustestweb Application portlet<
      /portlet-name>
    <instanceable>true</instanceable>
    <ajaxable>false</ajaxable>
  </portlet>
  <portlet>
    <!-- Matches definition in portlet.xml. -->
    <portlet-name>Mediustestweb Customer
      Application portlet</portlet-name>
    <instanceable>true</instanceable>
    <ajaxable>false</ajaxable>
  </portlet>
</liferay-portlet-app>

```

---



---

**Izvorna koda 4.14** Primer deskriptorja „liferay-display“.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE display PUBLIC "-//Liferay//DTD_Display_
  4.0.0//EN" "http://www.liferay.com/dtd/liferay-
  display_4.0.0.dtd">
<display>
  <category name="Vaadin">
    <portlet id="Mediustestweb_Application_portlet"/>
    <portlet id="Mediustestweb_Customer_Application_
      portlet"/>
  </category>
</display>

```

---

spremembe.

- Poslovni in spletni nivo

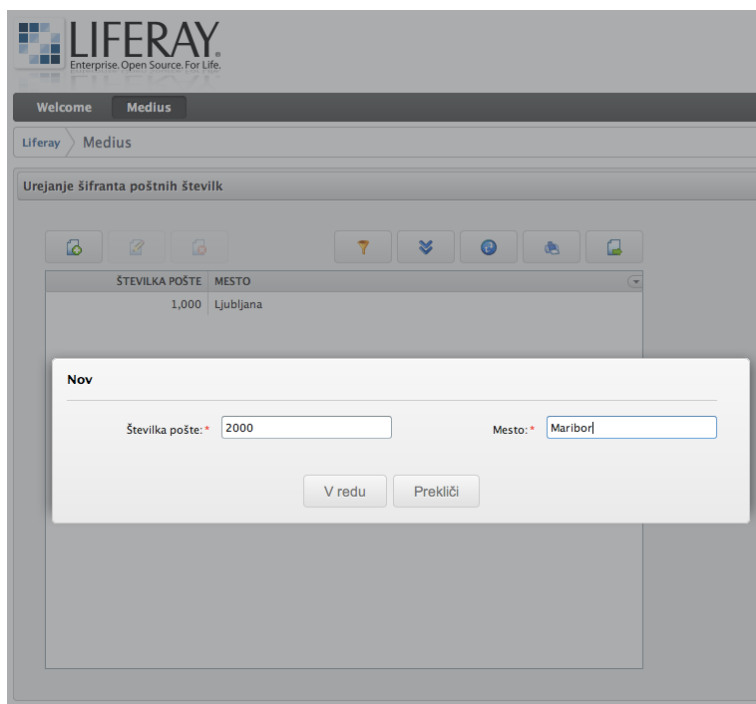
Izpolnjevanje predhodno definiranih kriterijev:

- *delujoči podatkovni viri na aplikacijskem strežniku*  
Podatkovni viri so bili definirani na preprost način s pomočjo skripte in delujejo. Kljub temu pa je potrebno poudariti, da je bilo treba za podporo EJB 3.0 specifikacije nekaj ročnega dela s kopiranjem določenih datotek iz privzete distribucije JBoss aplikacijskega strežnika.
- *delujoča implementacija JPA (dodajanje, urejanje, brisanje podatkov v samo podatkovno bazo ter pridobivanje podatkov iz baze)*  
Privzeti ponudnik Hibernate deluje brezhibno.
- *količina potrebnih sprememb na strežniški kodi aplikacije*  
Na strežniški kodi spremembe niso bile potrebne, kar pa je pričakovano, saj je bil aplikacijski strežnik enak kot pri referenčni implementaciji.
- *količina potrebnih sprememb na spletnem delu aplikacije*  
Potrebne so bile minimalne spremembe za pravilen prikaz portleta (nastavljanje širine). Nekaj dela je bilo tudi s kopiranjem tem in komponent na strežnik ter njihovo konfiguracijo za uporabo na aplikacijskem strežniku.
- *ustrezne nastavitve za prikaz pogledov v portletih na portalu*  
V orodju Vaadin imamo na voljo vtičnik, ki avtomatsko zgenerira potrebne deskriptorje, zato je bil ta del trivialen.
- *kompleksnost vzpostavitve osnovnega portala*  
Osnoven portal dosežemo z razširitvijo stisnjene datoteke, ki vsebuje že ustrezno nastavljen aplikacijski strežnik z nameščenim portalom, zato je bil tudi ta del trivialen.

- Nivo odjemalca

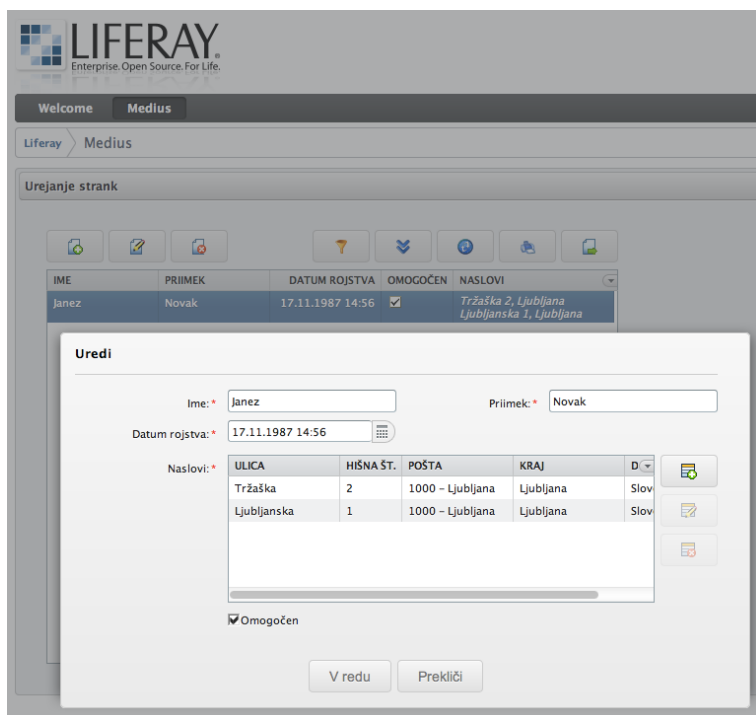
Na nivoju odjemalca preverimo delovanje aplikacije v naslednjih točkah:

- *ustrezen prikaz osnovne aplikacije, vključno z vsemi grafičnimi elementi (slike, barve, preliv, komponente grafičnega vmesnika...)*  
Vsi osnovni portleti (vse slike, komponente, pravilne barve) so po uvedenih dopolnitvah brezhibno prikazani.



Slika 4.16: Primer dodajanja entitete pri entitetnem zrnju „Post“.

- *delovanje dodajanja, urejanja, brisanja entitet na uporabniških maskah (vključno z vsemi razmerji do ostalih entitetnih zrn):*
  - \* entitete šifrantov  
Dodajanje entitet pri šifrantih lahko preverimo na entitetnem zrnju „Post“. Izkaže se, da dodajanje, urejanje in brisanje deluje brezhibno. Slika 4.16 prikazuje primer dodajanja entitete.
  - \* entitete pri razmerju ena proti mnogo:  
Obnašanje entitet pri razmerju ena proti mnogo lahko preverimo pri razmerju entitet „Customer“ in „Address“ (ena stranka ima lahko več naslovov). Izkaže se, da razmerje pri dodajanju nove stranke in dodajanju naslovov stranki deluje. Slika 4.17 prikazuje dodajanje entitete tipa „Customer“, ki ima lahko več entitet „Address“.
  - \* entitete pri razmerju mnogo proti ena:  
To razmerje imamo pri entitetnih zrnih „Address“ in „Post“.

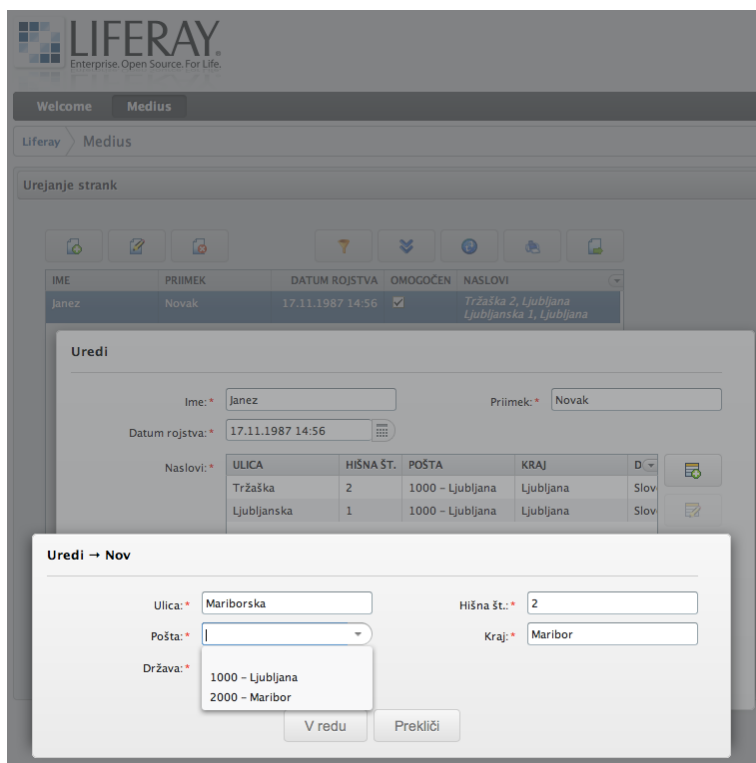


Slika 4.17: Primer dodajanja entitete pri entitetnem zrnju „Customer“.

Ko stranki dodajamo naslove, moramo pri vsakem naslovu izbrati eno pošto (torej več naslovov ima lahko isto pošto), kar v uporabniškem vmesniku pravilno vidimo kot spustni meni, kjer lahko izberemo eno vrednost. Primer prikazuje slika 4.18

- \* entitete pri razmerju mnogo proti mnogo  
To razmerje lahko opazujemo pri entitetnih zrnih „Order“ in „Product“ (naročilo ima lahko več postavk izdelkov, izdelek pa je lahko na postavkah različnih naročil). Tudi to razmerje deluje ustrezno, kar prikazuje slika 4.19.

- *delovanje sortiranja, filtriranja v tabelah*  
Sortiranje in filtriranje deluje v vseh tabelah.
- *delovanje izvoza v datoteke PDF in Microsoft Excel*  
Izvoz deluje pravilno.
- *pravilen prikaz in delovanje portleta pri različnih stanjih okna (mi-*

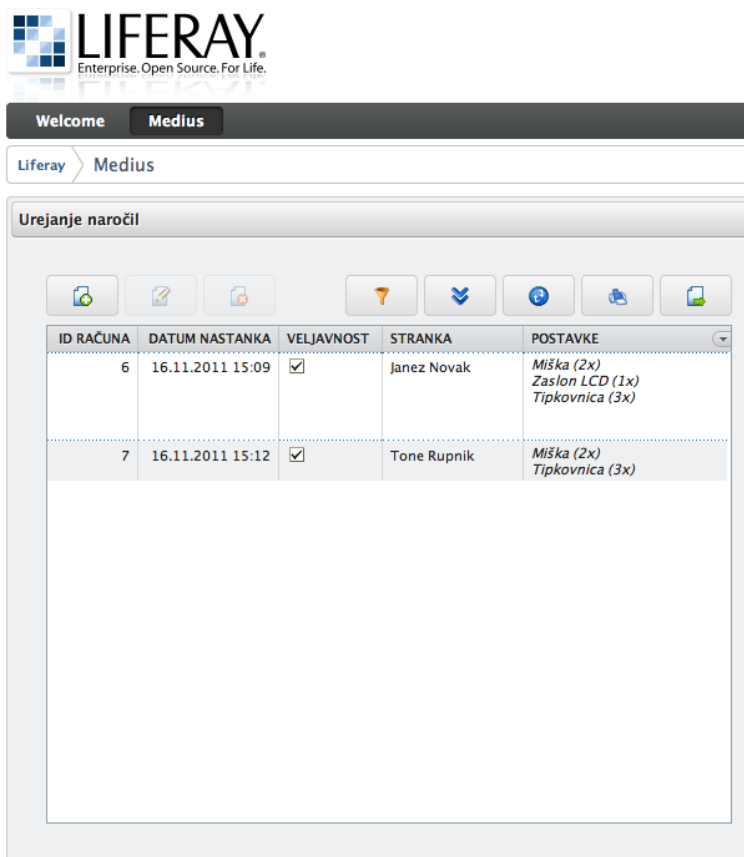


Slika 4.18: Izbira določene pošte pri vnašanju naslovov določene stranke.

*nimizacija, maksimizacija in klasičen prikaz)*

Portlet pravilno deluje, če ga minimiziramo in nato povrnemo v prvotno stanje. Prav tako deluje maksimizirano.

Zaključek analize integracije ogrodja Medius Vaadin Common na portal Liferay je, da je ta portal primeren za integracijo, saj vse funkcionalnosti delujejo brezhibno.



Slika 4.19: Primer portleta, ki prikazuje podatke o naročilih.

## 4.4 Integracija na IBM Websphere Portalu®

Drugi poskus integracije bo potekal na portalu IBM WebSphere Portal. Izbran je bil zato, ker je trenutno eden od vodilnih poslovnih portalov na trgu. Integracija bo potekala na različici Websphere Portal Express V7.0, ki je za namene preizkušanja dostopna na spletni strani podjetja IBM [30].

### 4.4.1 Analiza podprtih standardov

IBM WebSphere Portal V7.0 v privzeti namestitvi podpira naslednje standarde, ki so pomembni za ogrodje Medius Vaadin Common:

- JSR-168

- JSR-286
- JPA 1.0
- EJB 3.0

Izmed naštetih bi bila potencialno problematična lahko le podpora standardu JPA 1.0, saj je trenutno že uveljavljen novejši standard JPA 2.0. Kot se je kasneje izkazalo, je možno na strežnik namestiti dodatek, ki zagotovi tudi podporo standardu JPA 2.0, kar je za nas dodatna prednost, saj bo tako potrebnih manj prilagoditev na poslovnem nivoju.

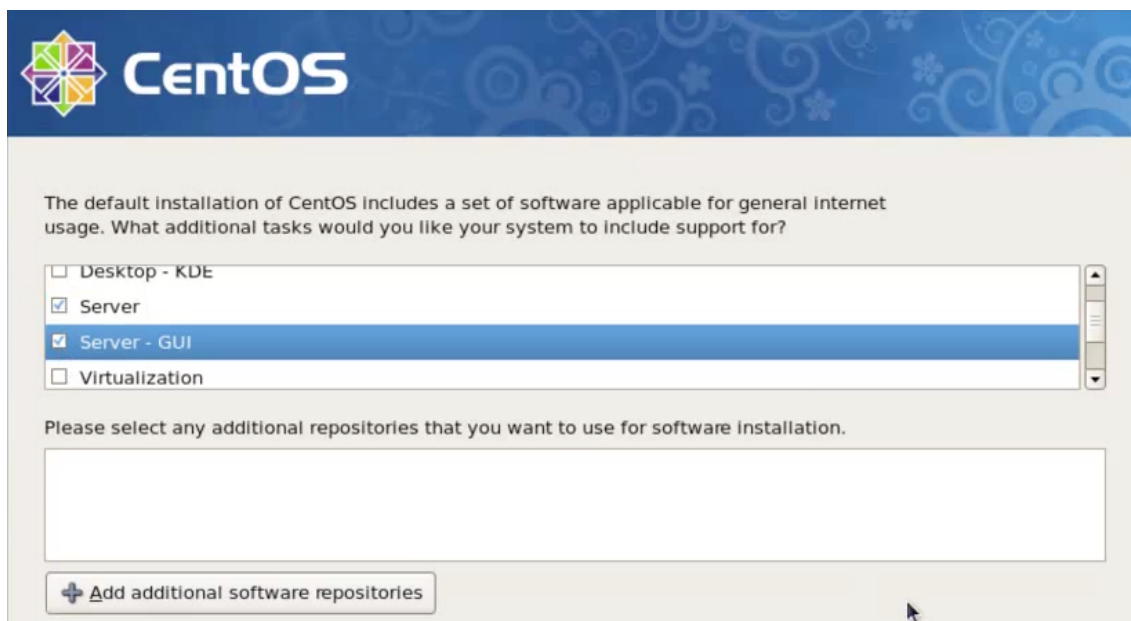
WebSphere Portal je osnovan na aplikacijskem strežniku IBM Websphere Application Server, kar pomeni, da bodo na poslovnem nivoju potrebne dodatne nastavitve, prilagojene temu strežniku. Privzeti ponudnik JPA je „JPA for WebSphere Application Server persistence provider“, ki je pravzaprav nadgrajena različica ponudnika „Apache OpenJPA persistence provider“. Podatkovna baza, ki jo bomo uporabili, je IBM DB2.

#### 4.4.2 Priprava okolja

Konfiguracija okolja za integracijo je naslednja:

- operacijski sistem Linux CentOS 5.7
- aplikacijski strežnik IBM Websphere Application Server
- podatkovna baza IBM DB2 9.7
- portal IBM Websphere Portal Express V7.0
- razvojno orodje (IDE) IBM Rational Application Developer
- orodje za upravljanje podatkovne baze IBM Data Studio

Pripravo okolja začnemo z namestitvijo ustreznega operacijskega sistema. V nasprotju s portalom Liferay, ki ga lahko poganjamo na poljubnem operacijskem sistemu, ki je sposoben poganjati Javo, je WebSphere portal nekoliko bolj specifičen, saj uradno podpira le določene operacijske sisteme. Za nas je najbolj zanimiv operacijski sistem Linux, saj na njem poteka večina razvoja. Med distribucijami Linuxa Websphere Portal uradno podpira komercialno dosegljiva Red Hat Enterprise Linux in SUSE Linux Enterprise. Za potrebe poskusa integracije pa je bil uporabljen CentOS, ki je prosto dostopna distribucija osnovana na kodi Red Hat Enterprise Linux-a ter zagotavlja enake



Slika 4.20: Izbira funkcionalnosti ob nameščanju operacijskega sistema CentOS (različica 5.7).

funkcionalnosti. Operacijski sistem je bil nameščen v virtualnem okolju VM-Ware Fusion.

Med samim nameščanjem operacijskega sistema dobimo v čarovniku, ki nas vodi po namestitvah, možnost izbire funkcionalnosti, ki jih želimo na sistemu. Pri tem moramo za ustrezno delovanje WebSphere Portala dodatno izbrati funkcionalnosti „Server“ in „Server - GUI“, kot to prikazuje slika 4.20.

Po namestitvi operacijskega sistema je potrebno zagotoviti, da so nameščene vse knjižnice, ki jih za delovanje potrebuje WebSphere Portal, poleg tega pa so potrebne še nekatere dodatne nastavitve v samem operacijskem sistemu. V operacijskem sistemu CentOS 5.7 to vključuje:

- nameščanje programskega paketa „compat-libstdc++-33-3.2.3-61.i386“ iz repozitorijev sistema CentOS
- urejanje datoteke „/etc/security/limits.conf“, kjer nastavimo maksimalno število možnih odprtih datotek v operacijskem sistemu (WebSphere Portal med nameščanjem odpre več datotek kot pa jih dovoljuje operacijski sistem) z dodajanjem naslednjih vrstic na koncu datoteke:

```
*      soft      nofiles 4096
*      hard      nofiles 10240
```

- urejanje datoteke „/etc/hosts“, kjer moramo ustvariti navidezni naslov strežnika, na katerem bo potekala namestitve WebSphere Portala. V našem primeru je to localhost, vendar pa je to vrednost, ki je namestitveni program za Websphere Portal ne sprejme, zato v datoteki dodamo vrstico z novim imenom (na tem naslovu bo tudi dosegljiv portal):

```
127.0.0.1      myportal.portal.com
```

S tem smo zaključili osnovno nastavitve sistema. Sledi nameščanje WebSphere Portala. Datoteke in navodila za namestitev so na voljo na spletni strani podjetja IBM [30]. Med izpolnjevanjem parametrov v čarovniku za namestitev posebej pozorno nastavimo parameter „Fully qualified host name“, ki ga nastavimo na naslov, ki smo ga prej dodali v datoteko „hosts“. Namestitveni program nato v operacijski sistem namesti celotno okolje, kar vključuje:

- aplikacijski strežnik IBM Websphere Application Server
- podatkovno bazo IBM DB2
- portal IBM WebSphere Portal

Po uspešni osnovni namestitvi portala moramo za ustrezno delovanje po standardu JPA 2 dodatno nadgraditi nameščeni portal. To storimo z IBM-ovim orodjem IBM Installation Manager, ki nam ponudi nadgraditev s paketom „WebSphere Application Server V7 Feature Pack for OSGi Applications and Java Persistence API 2.0“. Po uspešnem nameščanju paketa moramo biti posebej previdni, da v orodju Installation Manager zaženemo program za nadgradnjo instanc (t.i. „server node“) nameščenih aplikacijskih strežnikov WebSphere ter ustrezni instanci dodelimo novo nameščeno funkcionalnost. Na aplikacijskih strežnikih WebSphere namreč obstaja nabor nastavitev, ki definirajo posamezno instanco, le-tej pa moramo posebej definirati, da mora vključevati tudi funkcionalnost JPA 2.

Za celotno integracijo potrebujemo še razvojno okolje in orodje za urejanje podatkovne baze. V ta namen namestimo razvojno okolje „IBM Rational Application Developer“ in orodje „IBM Data Studio“. Rational Application Developer je pravzaprav nadgrajeno razvojno okolje Eclipse, kateremu so dodali vtičnike za delo z izvajalnim okoljem aplikacijskega strežnika Websphere

Application Server. Kot tak je še posebej primeren za naše ogrodje, saj je delo z njim praktično enako delu z orodjem Eclipse, ki smo ga uporabljali pri integraciji na portal Liferay.

### 4.4.3 Poskus integracije

#### 4.4.3.1 Podatkovni nivo

Na podatkovnem nivoju ponovno najprej iz konceptualne sheme podatkovne baze generiramo logično shemo, kjer v orodju Power Designer nastavimo, da je ciljna baza IBM DB2. Dodatno moramo pri vseh atributih, ki so primarni ključi, nastaviti, da ima atribut lastnost „Identity“, kar pomeni, da bo podatkovna baza sama generirala vrednosti primarnega ključa pri vstavljanju novih entitet. Pri generiranju skripte je orodje Power Designer javilo napako, da so nekatera imena razmerij med entitetnimi tipi predolga, zato jih je bilo potrebno skrajšati.

Na podatkovnem strežniku IBM DB2 je bila nato kreirana nova podatkovna baza (prek terminalskega vmesnika) z imenom „medius“, v orodju Data Studio pa je bila kreirana povezava do te baze z naslednjimi parametri:

- database (vrednost „medius“)
- host (vrednost „localhost“)
- port number (vrednost „50000“)
- username (vrednost „wp70db2“; to je uporabnik, ki ga ob namestitvi podatkovnega strežnika kreira namestitveno orodje)
- password (vrednost „wpadmin“; geslo definiramo v namestitvenem čarovniku ob nameščanju okolja za portal)

Prek te povezave je bila pognana skripta za generiranje strukture podatkovne baze. Ob prvem poskusu je bila javljena napaka in izkazalo se je, da je potrebno odstraniti SQL poizvedbe v prvem delu skripte, ki na začetku počistijo obstoječe table („DROP“ stavki), saj le-teh v našem primeru še ni v bazi. V drugem poskusu je kreiranje uspelo. Vsebina skripte za izdelavo podatkovne baze prikazuje dodatek C.1.

#### 4.4.3.2 Poslovni nivo

Tako kot pri integraciji na portal Liferay, začnemo integracijo na WebSphere Portal z definiranjem podatkovnih virov do podatkovne baze na aplikacijskem strežniku. Prvi korak je zagon aplikacijskega strežnika, na katerem teče WebSphere Portal. Če smo pustili privzeto lokacijo namestitve, strežnik zaženemo tako, da se v terminalu premaknemo v mapo „/opt/IBM/WebSphere/AppServer/bin“ in kot administrator poženemo naslednji ukaz:

```
./startServer.sh WebSphere_Portal
```

Ko strežnik konča z zagonom, v spletnem brskalniku odpremo administracijsko konzolo, ki jo v našem primeru najdemo na naslovu:

```
https://myportal.portal.com:10032/ibm/console
```

Po prijavi (v našem primeru z uporabniškim imenom „wpsadmin“ in enakim geslom) v konzolo v levem meniju odpremo kategorijo „Resources“, znotraj nje izberemo kategorijo „JDBC“ in izberemo povezavo „Data sources“.

Nato iz spustnega menija kot mesto kreiranja podatkovnega vira izberemo „Node=portalnode, Server=WebSphere\_Portal“ ter pritisnemo gumb „New“ za dodajanje novega podatkovnega vira. Prikaže se čarovnik za dodajanje novega podatkovnega vira. V posameznih korakih izpolnimo naslednje parametre:

- Data source name  
Podatkovni vir poimenujemo z imenom „mediusDatasource“ (enako kot smo imenovali podatkovni vir pri integraciji na portal Liferay; s tem dosežemo, da nam ne bo potrebno spreminjati konfiguracijskih datotek).
- JNDI name  
Vrednost nastavimo na „jdbc/mediusDatasource“.
- Select JDBC provider  
Izberemo možnost „Select existing...“ in nato iz spustnega menija vrednost „wpdbJDBC\_db2“.
- Driver type  
Pustimo privzeto vrednost 4.
- Database name  
Vnesemo ime podatkovne baze, v našem primeru „medius“
- Server name  
Ime podatkovnega strežnika nastavimo na „localhost“, saj teče lokalno.

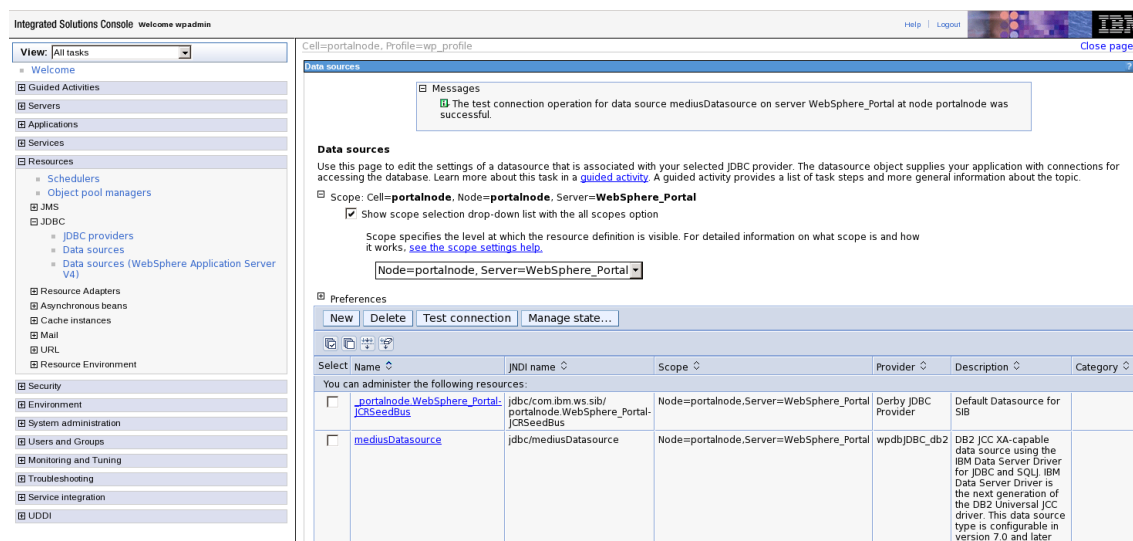
- Port number  
Pustimo privzeto vrednost 50000.
- Authentication alias for XA recovery; Component-managed authentication alias; Container-managed authentication alias  
Pri teh parametrih moramo navesti alias varnostnih nastavitev za dostop do virov. Ker jih v tem trenutku še nismo kreirali, odpremo povezavo „Global J2C authentication alias“ v novem zavihku. Odpre se seznam aliasov za varnostne nastavitve. S pritiskom na „new“ dodamo nov alias z naslednjimi parametri:
  - Alias (vrednost nastavimo na npr. „wp70db2“, saj ga bomo uporabljali za povezavo do podatkovne baze)
  - User ID (vrednost je „wp70db2“)
  - Password (vrednost je „wpadmin“)
 Ko smo dodali nov alias (da se dejansko shrani na strežnik je vnos potrebno še potrditi na vrhu strani s pritiskom na „save to master configuration“), osvežimo zavihek za dodajanje novega podatkovnega vira in kot vrednost v začetku omenjenih parametrov izberemo vnešeni alias (naziv aliasa bo imel še predpono „portalnode“). V primeru, da nas brskalnik vrne na začetno stran podatkovnih virov, bomo morali še enkrat vnesti podatke za podatkovni vir.
- Mapping-configuration alias  
Pustimo privzeto vrednost „(none)“.

Vnos potrdimo na koncu z gumbom „Finish“. Prikaže se seznam podatkovnih virov, v katerem je na novo definirani vir „mediusDatasource“. Da se le-ta shrani na podatkovni strežnik, je vnos potrebno dodatno potrditi s pritiskom na „Save directly to master configuration“ na vrhu strani. Delovanje novega vira lahko preverimo tako, da ga označimo v seznamu in pritisnemo na gumb „Test“. Rezultat preizkusa bo izpisan na vrhu strani kot to prikazuje slika 4.21.

Naslednji korak je kreiranje entitetnih in sejnih zrn, kar storimo v orodju Rational Application Developer (RAD).

Začnemo tako, da kreiramo nov projekt tipa „EJB Project“, ki ga poimenujemo „MediusTestEJB“. V nastavitvah nastavimo naslednje parametre:

- Target runtime (WebSphere Portal v7.0 stub)



Slika 4.21: Test podatkovnega vira mediusDatasource na aplikacijskem strežniku Websphere.

- EJB module version (3.0)
- Configuration (Default Configuration for WebSphere Portal v7.0 stub)

S tem smo projektu nastavili privzete nastavitve za delovanje na WebSphere Portalu. Enako kot pri integraciji na Liferay, tudi tu označimo možnost „Add project to an EAR“, ki ga poimenujemo „MediusTestEAR“. Nato na preostalih straneh sprejememo privzete nastavitve, dokler ne pridemo do zadnje strani. Tu odključamo možnost „Create an EJB Client JAR module“ in obkljukamo „Generate ejb-jar.xml deployment descriptor“. To je tudi razlog zakaj smo tip projekta nastavili na „EJB Project“ in ne na „JPA project“. Pri projektu tipa „JPA“ namreč RAD ne ponudi možnosti kreiranja nekaterih deksriptorjev, ki so potrebni za delovanje ogrodja Medius Vaadin Common na WebSphere Portalu. Kljub temu pa bomo v projektu potrebovali tudi funkcionalnosti „JPA“, zato projektu v nastavitvah pod „Project Facets“ dodamo modul „JPA“, ki ga nastavimo na različico 2.0. Pri tem spodaj izberemo povezavo „Further configuration available“ in odpre se čarovnik, v katerem izberemo naslednje nastavitve:

- Platform (RAD JPA 2.0 Platform)
- JPA implementation (Library Provided by Target Runtime)

- Connection (None)
- Persistent class management (Annotated classes must be listed in persistence.xml)
- obkljukamo možnost „Create mapping file (orm.xml)“

Po potrditvi RAD doda funkcionalnost „JPA“ k obstoječemu projektu (kreira potrebne deskriptorje).

Sledi dodajanje strežniške kode projektu. Najprej v projektu „MediusTestEAR“ kreiramo mapo „lib“ z vsemi potrebnimi knjižnicami, ki jih potrebuje ogrodje „Medius Vaadin Common“. Nato v projekt „MediusTestEJB“ dodamo mapo „src-ejb“, kamor dodamo izvirno kodo entitetnih in sejnih zrn. Projektu „MediusTestEJB“ še dodamo reference do knjižnic v projektu „MediusTestEAR“ (v t.i. „buildpath“). V zadnjem koraku je potrebno nastaviti deskriptorje. Deskriptor persistence.xml nastavimo tako, kot prikazuje izvirna koda 4.15, posebej pazimo na parameter „jta-data-source“, ki ga nastavimo na ime podatkovnega vira, ki smo ga dodali na aplikacijski strežnik.

WebSphere portal zahteva še poseben deskriptor „ejb-jar.xml“, ki nastavi specifične parametre za portal. Primer, ki vsebuje parametre za sejno zrno „PostManager“ prikazuje izvirna koda 4.16. V primeru integracije ogrodja Medius Vaadin Common na Liferay portal smo morali posebej dodati knjižnico za povezavo do MySQL podatkovne baze, saj JBoss aplikacijski strežnik le-te privzeto ne vsebuje. WebSphere pa knjižnico za povezavo do podatkovne baze DB2 že vsebuje, zato dodajanje v projekt in deskriptorje ni potrebno.

Pred samo namestitvijo moramo v projektu „MediusTestEAR“ nastaviti namestitveni deskriptor (možnost „deployment assembly“ v nastavitvah projekta), kjer dodamo vnose za knjižnice v mapi „lib“ v projektu, podobno kot pri integraciji na Liferay. Namestitvenega deskriptorja za projekt „MediusTestEJB“ ni potrebno spreminjati.

Nato lahko poskusimo z namestitvijo strežniškega dela aplikacije. Projekt „MediusTestEAR“ izvozimo iz okolja RAD v obliki datoteke „EAR“. Nameščanje poteka preko administracijske konzole portala. Po prijavi v konzolo v levem meniju odpremo kategorijo „Applications“, znotraj nje izberemo kategorijo „Application Types“ in izberemo povezavo „WebSphere enterprise applications“. Prikaže se seznam aplikacij, ki so že nameščene na aplikacijskem strežniku, kot to prikazuje slika 4.22.

Za nameščanje naše aplikacije pritisnemo na gumb „Install“, ter navedemo pot do datoteke „EAR“. Čarovnik nas nato vodi skozi namestitev, pri čemer sprejememo ponujene privzete vrednosti parametrov, razen v koraku „Map modules to servers“. V tem koraku določimo, na kateri strežnik

---

**Izvorna koda 4.15** Deskriptor persistence.xml za WebSphere Portal.
 

---

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/
  xml/ns/persistence" xmlns:xsi="http://www.w3.org
  /2001/XMLSchema-instance" xsi:schemaLocation="http://
  java.sun.com/xml/ns/persistence http://java.sun.com/
  xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="MediusTestEJB"
    transaction-type="JTA">
    <jta-data-source>jdbc/mediusDatasource</
      jta-data-source>
    <mapping-file>META-INF/orm.xml</mapping-
      file>

    <class>si.medius.test.entity.Address</
      class>
    <class>si.medius.test.entity.Customer</
      class>
    <class>si.medius.test.entity.Order</
      class>
    <class>si.medius.test.entity.OrderItem</
      class>
    <class>si.medius.test.entity.Post</class
      >
    <class>si.medius.test.entity.Product</
      class>

  </persistence-unit>
</persistence>

```

---

---

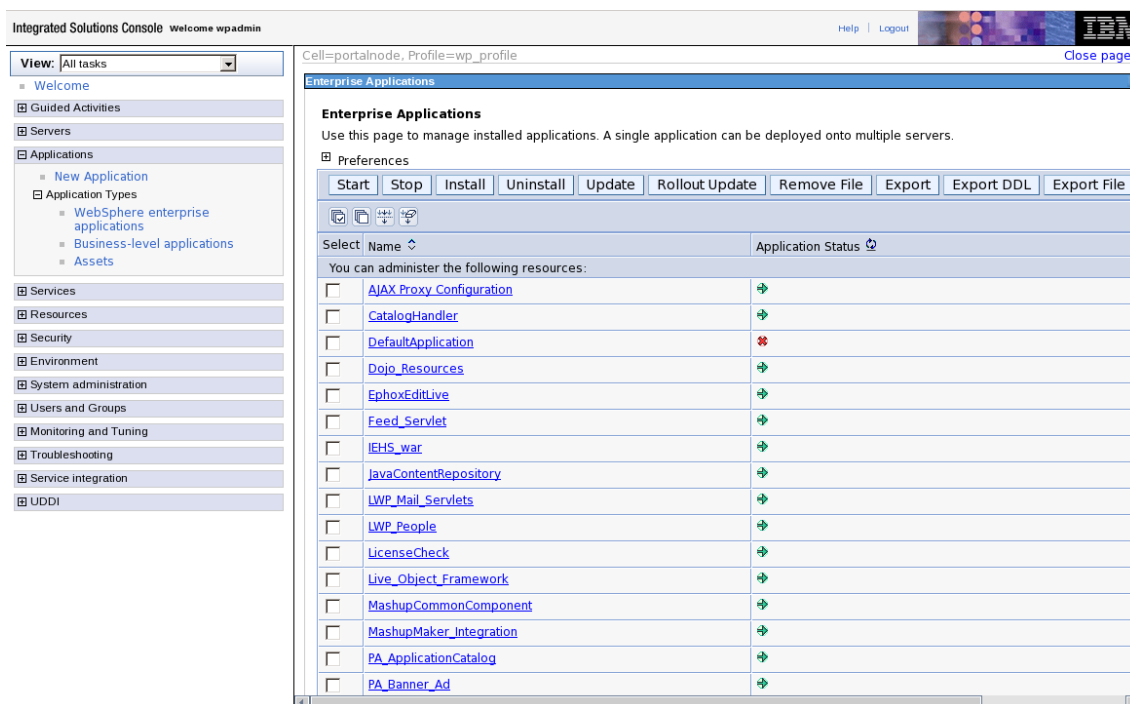
**Izvorna koda 4.16** Primer deskriptorja ejb-jar.xml za WebSphere Portal.

---

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar version="3.0" xmlns="http://java.sun.com/xml/ns
  /javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" xsi:schemaLocation="http://java.sun.com/xml
  /ns/javaee http://java.sun.com/xml/ns/javaee/ejb-
  jar_3_0.xsd">
  <display-name>MediusTestEJB </display-name>

  <enterprise-beans>
    <session>
      <description>PostManager</description>
      <display-name>PostManager</display-name>
      <ejb-name>PostManager</ejb-name>
      <business-local>
        si.medius.test.managers.local.
        PostManagerLocal
      </business-local>
      <business-remote>si.medius.test.managers
        .remote.PostManagerRemote</business-
        remote>
      <ejb-class>si.medius.test.managers.
        PostManager</ejb-class>
      <session-type>Stateless</session-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

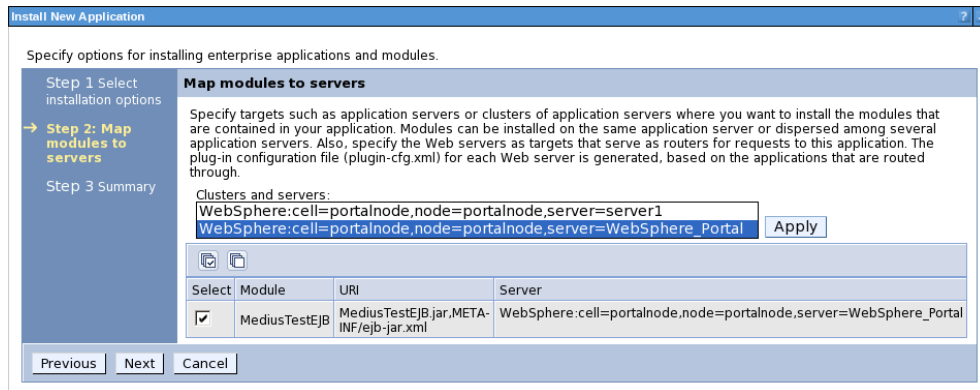
---



Slika 4.22: Vmesnik za upravljanje aplikacij v administracijski konzoli aplikacijskega strežnika WebSphere AS.

se bo aplikacija namestila. Privzeta vrednost je aplikacijski strežnik „server1“, mi pa moramo aplikacijo namestiti na strežnik na katerem teče portal, t.j. „WebSphere.Portal“, zato v tem koraku označimo našo aplikacijo, obklužimo strežnik „WebSphere.Portal“ in pritisnemo gumb „Apply“ (slika 4.23). V vrstici, ki opisuje našo aplikacijo, bomo nato videli popravljeno vrednost parametra „Server“.

V naslednjih korakih sprejmemo privzete nastavitve, strežnik namesti aplikacijo in izpiše nekaj obvestil o nameščanju. V zadnjem koraku moramo še dokončno potrditi namestitev z izbiro povezave „Save directly to master configuration“. Naša aplikacija se bo nato pojavila v seznamu nameščenih aplikacij, kot jih prikazuje slika 4.22. V tem trenutku je koristno pogledati v dnevniške datoteke strežnika (privzeta lokacija dnevniške datoteke za WebSphere Portal je „/opt/IBM/WebSphere/AppServer/profiles/wp\_profile/logs/WebSphere.Portal/SystemOut.log“), kjer se izpisujejo podatki o nameščanju. V našem primeru ob nameščanju aplikacije v datoteki zasledimo naslednji zapis, ki potrjuje uspešno namestitev:



Slika 4.23: Nastavitev strežnika za namestitvev aplikacije.

ADMA7021I: Distribution of application MedusTestEAR completed successfully.

V tem trenutku je aplikacija nameščena na strežniku, ni pa še v delujočem stanju (v seznamu aplikacij v administracijski konzoli je poleg naziva aplikacije rdeč križec). Da aplikacijo dejansko zaženemo, jo označimo v seznamu aplikacij in pritisnemo na gumb „Start“. Strežnik bo inicializiral in zagnal aplikacijo. V dnevniški datoteki bomo opazili naslednje zapise:

ADMN1008I: An attempt is made to start the MediusTestEAR application.

WSVR0190I: Starting composition unit WebSphere:cuname=MediusTestEAR in BLA WebSphere:blaname=MediusTestEAR.

WSVR0200I: Starting application: MediusTestEAR

WSVR0037I: Starting EJB jar: MediusTestEJB.jar

CNTR0167I: The server is binding the si.medius.test.managers.local.PostManagerLocal interface of the PostManager enterprise bean in the MediusTestEJB.jar module of the MediusTestEAR application. The binding location is: ejblocal:MediusTestEAR/MediusTestEJB.jar/PostManager#si.medius.test.managers.local.PostManagerLocal

...

WSVR0057I: EJB jar started: MediusTestEJB.jar

WSVR0221I: Application started: MediusTestEAR

Ti zapisi označujejo referenco, pod katero sedaj na aplikacijskem strežniku spletni del aplikacije lahko najde sejna zrna.

#### 4.4.3.3 Spletni nivo

Preostane nam torej še implementacija spletnega dela aplikacije. Ker je RAD osnovan na orodju Eclipse, lahko tudi v njem namestimo vtičnik za delo z Vaadin projekti. Po namestitvi kreiramo nov projekt tipa „Vaadin project“ in ga poimenujemo „MediusTestWEB“. V različnih korakih čarovnika za dodajanje projekta izberemo naslednje nastavitve:

- Target Runtime (WebSphere Portal v7.0 stub)
- Configuration (Vaadin, Java 6, Servlet 2.4)
- Generic portlet (Portlet 1.0)
- Vaadin version (6.6.6, t.j. različica, ki jo uporablja Medius Vaadin Common v različici, ki je bila uporabljena za integracijo)

Za razliko od portala Liferay smo tukaj izbrali portlet 1.0, torej portlet po specifikaciji JSR-168, saj se je izkazalo, da v primeru portleta 2.0 pride do težav pri prikazu vsebine portletov, ki v tem primeru ne delajo pravilno. Ker je vtičnik prirejen za Liferay, bo generiral deskriptorje za Liferay, ki jih bo dodal v mapo „WebContent/WEB-INF“. Te deskriptorje lahko pobrišemo. Opazimo tudi, da RAD ob kreiranju projekta avtomatsko doda deskriptorja „ibm-web-bnd.xml“ in „ibm-web-ext.xml“, ki pa sta ustrezna, saj sta namenjena WebSphere Portalu.

Razvoj spletnega dela začnemo z dodajanjem reference na projekt „MediusTestEJB“, da lahko RAD ustrezno prepozna entitetna zrna. V naslednjem koraku v mapo „/lib“ dodamo knjižnice, ki so specifične za delovanje ogrodja Medius Vaadin Common na spletni delu aplikacije. V projekt dodamo še mapo „WebContent/VAADIN“, v katero skopiramo teme za ustrezen prikaz grafičnega uporabniškega vmesnika in komponente (t.i. „widgetset“, komponente lahko tudi generiramo s pomočjo vtičnika). V „src“ nato dodamo kodo spletnih pogledov in aplikacij, ki te poglede uporabljajo. Projektu „MediusTestWEB“ je potrebno še dodati referenco na projekt „MediusTestEJB“, da lahko spletni del dostopa do definicij sejnih in entitetnih zrn.

Sledi nastavitvev deskriptorjev. Nastavitvev deskriptorja „portlet.xml“ prikazuje izorna koda 4.17 in je zelo podoben deskriptorju pri integraciji na Liferay portal (razlikuje se le v parametru „portlet-class“, ki ima referenco na razred, ki ustreza specifikaciji JSR-168). Bistvena razlika je v deskriptorju web.xml, ki ga prikazuje izvorna koda 4.18. V njem je namreč potrebno definirati servlet, od katerega je odvisen portlet (pri JSR-286 specifikaciji servlet ni več potreben za delovanje portleta).

---

**Izvorna koda 4.17** Primer deskriptorja „portlet.xml“ za portal WebSphere.

---

```

<?xml version="1.0" encoding="UTF-8"?><portlet-app
  version="1.0" xmlns="http://java.sun.com/xml/ns/
  portlet/portlet-app_1_0.xsd" xmlns:xsi="http://www.w3
  .org/2001/XMLSchema-instance" xsi:schemaLocation="
  http://java.sun.com/xml/ns/portlet/portlet-app_1_0.
  xsd http://java.sun.com/xml/ns/portlet/portlet-
  app_1_0.xsd">

  <portlet>
    <portlet-name>Mediustestweb Post Application
      portlet</portlet-name>
    <display-name>Urejanje poŁ~tnih Ł~tevil</
      display-name>

    <portlet-class>com.vaadin.terminal.gwt.server.
      ApplicationPortlet</portlet-class>
    <init-param>
      <name>application</name>
      <value>PostApplicationServlet</value>
    </init-param>

    <init-param>
      <name>widgetset</name>
      <value>si.medius.test.portletApplications.
        widgetset.MediustestwebWidgetset</value>
    </init-param>

    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>view</portlet-mode>
    </supports>

    <portlet-info>
      <title>Urejanje poŁ~tnih Ł~tevil</title>
      <short-title>Urejanje poŁ~tnih Ł~tevil</
        short-title>
    </portlet-info>
  </portlet>
</portlet-app>

```

---

**Izvorna koda 4.18** Primer deskriptorja „web.xml“ za portal WebSphere.

---

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp.ID" version="2.4" xmlns="http://java
.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org
/2001/XMLSchema-instance" xsi:schemaLocation="http://
java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/
j2ee/web-app_2_4.xsd">
    <display-name>MediusTestWEB</display-name>
    <context-param>
        <description>
            Vaadin production mode</description>
        <param-name>productionMode</param-name>
        <param-value>false</param-value>
    </context-param>
    <servlet>
        <servlet-name>Mediustestweb Post
            Application</servlet-name>
        <servlet-class>com.vaadin.terminal.gwt.
            server.ApplicationServlet</servlet-
            class>
        <init-param>
            <description>
                Vaadin application class to
                start</description>
            <param-name>application</param-
            name>
            <param-value>si.medius.test.
                portletApplications.
                PostApplication</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>Mediustestweb Post
            Application</servlet-name>
        <url-pattern>/PostApplicationServlet/*</
            url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Mediustestweb Post
            Application</servlet-name>
        <url-pattern>/VAADIN/*</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
    >

```

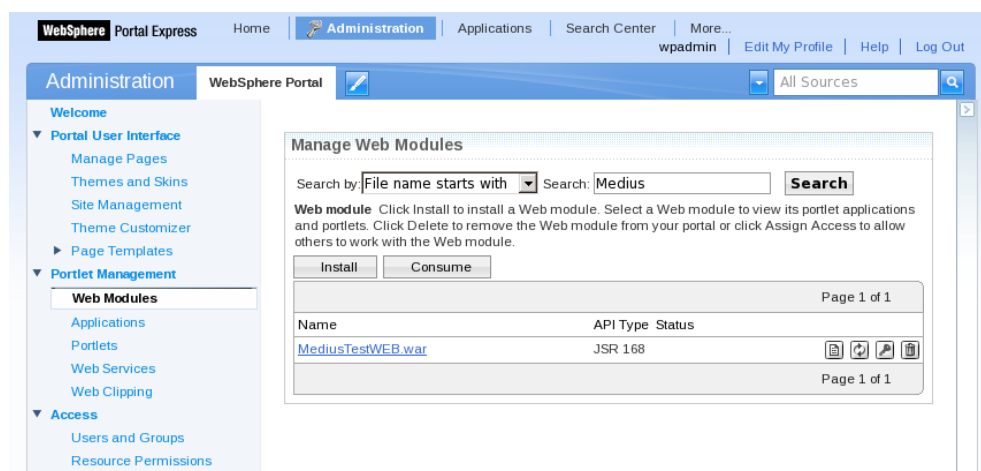
IBM WebSphere portal potrebuje še dva dodatna deskriptorja za pravilno delovanje. To sta „ibm-web-bnd.xml“ in „ibm.web.ext.xml“, ki pa ju je kreiral že RAD ob kreiranju projekta in ju ni potrebno dodatno spreminjati.

S tem smo končali osnovno implementacijo spletnega dela, ki vsebuje en portlet za prikaz entitetnega zrna „Post“. Spletni del projekta dodamo kot referenco v namestitveni deskriptor projekta „MediusTestEAR“ (v nastavitvah projekta pod „deployment assembly“). Ta projekt nato izvozimo v obliki EAR datoteke in jo prek administracijske konzole strežnika namestimo po enakih korakih, kot so bili opisani v poslovnem delu aplikacije (EAR datoteka sedaj vsebuje strežniški in spletni del aplikacije, zato moramo pred nameščanjem pobrisati predhodno nameščeno aplikacijo, ki vsebuje le strežniški del). Aplikacijo nato zaženemo in v dnevniških datotekah med drugim najdemo naslednje zapise:

```

ADMN1008I: An attempt is made to start the MediusTestEAR
           application .
â€œ|
WSVR0200I: Starting application: MediusTestEAR
â€œ|
WSVR0037I: Starting EJB jar: ejb.jar
CNTR0167I: The server is binding the si.medius.test.
           managers.local.PostManagerLocal interface of the
           PostManager enterprise bean in the ejb.jar module of
           the MediusTestEAR application. The binding location
           is: ejblocal:MediusTestEAR/ejb.jar/PostManager#si.
           medius.test.managers.local.PostManagerLocal
â€œ|
WSVR0037I: Starting EJB jar: MediusTestEJB.jar
com.ibm.ws.webcontainer.webapp.WebGroupImpl WebGroup
SRVE0169I: Loading Web Module: MediusTestWEB.
com.ibm.ws.wswebcontainer.VirtualHost addWebApplication
SRVE0250I: Web Module MediusTestWEB has been bound to
           default_host
           [*:80,*:443,*:10000,*:10002,*:10039,*:10029,*:10032].
com.ibm.wps.services.registry.ObjectModelRegistry
           registerPortleregisterWASPortletApplicationtApplication
EJPPG0024I: Web application with context root /
           MediusTestWEB is deployed in the application server
           but not registered with portal.
WSVR0221I: Application started: MediusTestEAR

```



Slika 4.24: Prikaz spletnih aplikacij v administracijskem delu portala WebSphere.

Zapisom o nameščanju strežniške kode so se pridružili zapisi o nameščanju spletnega dela aplikacije, pri čemer je posebej pomemben del, ki pravi „MediusTestWEB is deployed in the application server but not registered with portal“. Aplikacijski strežnik namreč namesti spletni del aplikacije, ki pa ga portal privzeto ne zazna. Za namestitev reference spletnega dela aplikacije odpremo WebSphere Portal (v našem primeru na naslovu <http://myportal.portal.com:10039/wps/myportal/>) in se prijavimo kot administrator (v našem primeru z up. imenom in geslom „wpadmin“). Nato izberemo povezavo „Administration“ ter v levem meniju izberemo povezavo „Import XML“ v kategoriji „Portal Settings“. Tu lahko s pomočjo XML datoteke, kot jo prikazuje izvorna koda 4.19 vnesemo referenco na EAR, ki je nameščen na strežniku.

Po namestitvi preverimo ali se je referenca pravilno vnesla. V administracijskem delu portala izberemo povezavo „Web Modules“ znotraj kategorije „Portlet Management“ in v polje za iskanje vnesemo „Medius“. Če je bil vnos reference uspešen, bo iskalnik našel projekt „MediusTestWEB.war“ (slika 4.24). S pritiskom na povezavo z imenom projekta lahko vidimo nameščene portlete.

Preostane nam še dodajanje portleta na stran v uporabniškem vmesniku portala. V ta namen se postavimo na začetno stran portala (prijavljeni kot administrator). Nato izberemo možnost „Actions“ v zgornjem desnem kotu in v meniju postavko „Edit Page“. Portal preklopi način delovanja v urejanje. Za testne namene ustvarimo novo stran (možnost „New page“ levo zgoraj), ter se premaknemo nanjo. Ponovno preklopimo v način urejanja ter pritisnemo gumb

---

**Izvorna koda 4.19** XML datoteka za vnos reference nameščenega EAR projekta na portal.

---

```
<?xml version="1.0" encoding="UTF-8"?>

<request xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
      xsi:noNamespaceSchemaLocation="PortalConfig_7
        .0.0.xsd"
      type="update" create-oids="true">

  <portal action="locate">
    <web-app action="update" active="true"
      predeployed="true">

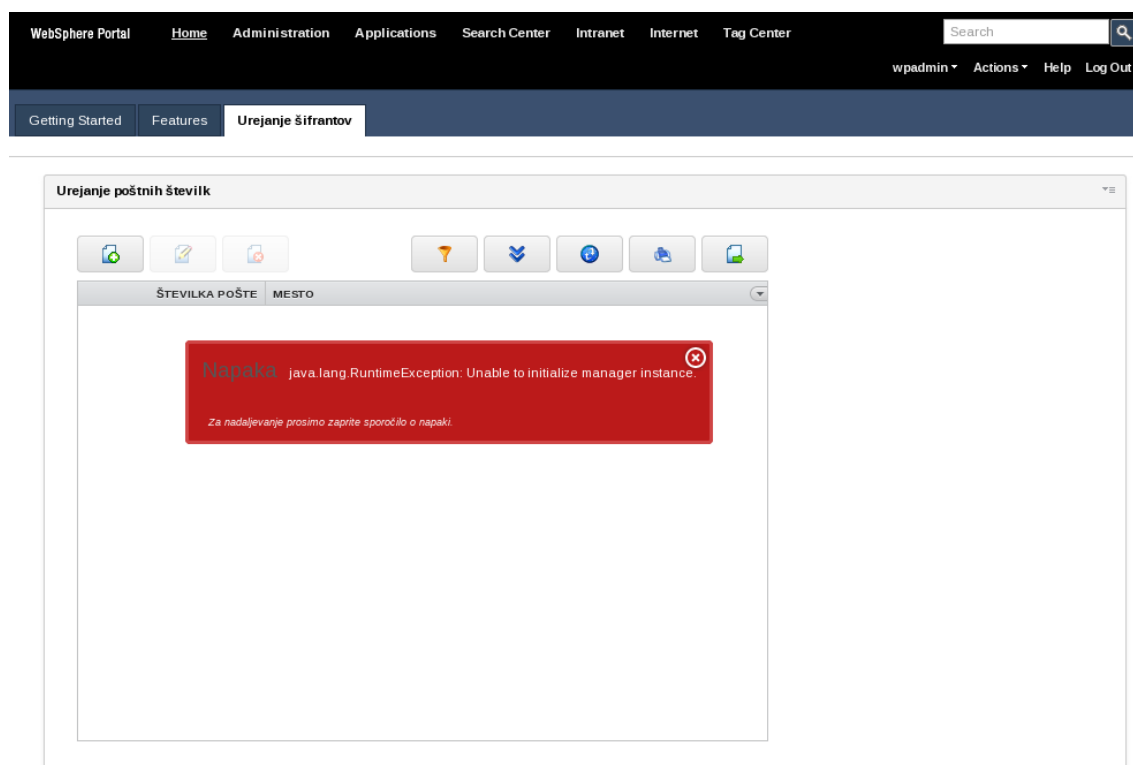
      <url>file://localhost/opt/IBM/WebSphere/
        AppServer/profiles/wp_profile/
        installedApps/portaldnode/MediusTest.ear/
        MediusTestWEB.war</url>
      <context-root>/MediusTestWEB</context-root>
      <display-name>MediusTestWEB</display-name>

      <portlet-app action="update" active="true">
        </portlet-app>
      </web-app>
    </portal>
  </request>
```

---

„Customize“ desno zgoraj. Odpre se pregled možnih komponent za anmestitev na portal. V meniju komponent se premaknemo na „All“ in z iskalnikom poišemo portlet, ki smo ga namestili (v našem primeru se je imenoval „Urejanje šifranta poštних števil“; če ne poznamo imena portleta, ga lahko poiščemo v pregledu spletnih aplikacij, ki ga prikazuje slika 4.24). V desnem delu strani se bo prikazal naslov portleta, ki ga primemo s kazalcem miške in potegnemo na stran spodaj. Dodajanje zaključimo s pritiskom na gumb „Save & Exit“ desno zgoraj. Portal bo preklupil v način prikaza.

V primeru, da je vse pravilno nastavljeno, se portlet pokaže na strani. V našem primeru se portlet sicer prikaže, vendar takoj pride do napake pri



Slika 4.25: Napaka ob prvem poskusu dodajanja portleta na stran portala.

generiranju vsebine, kar je vidno na sliki 4.25.

Če pogledamo v dnevniške datoteke napak (datoteka „SystemErr.log“), najdemo naslednji zapis:

```
java.lang.RuntimeException: java.lang.RuntimeException:
    Unable to initialize manager instance.
    â€œ|
Caused by: javax.naming.NameNotFoundException: Context:
    portalnode/nodes/portalnode/servers/WebSphere_Portal,
    name: MediusTestEAR/PostManager/local: First
    component in name MediusTestEAR/PostManager/local not
    found. [Root exception is org.omg.CosNaming.
    NamingContextPackage.NotFound: IDL:org.omg/CosNaming/
    NamingContext/NotFound:1.0]
```

Iz zapisa je razvidno, da aplikacija na aplikacijskem strežniku išče nameščeno sejno zrno pod nazivom „MediusTestEAR/PostManager/local“, ki pa ga apli-

```
ejblocal:[Ime EAR projekta]/ [Ime EJB datoteke (v
    stisnjenem formatu .jar) v EAR projektu]/ [Ime
    osnovnega manager-ja]#[Ime paketa, kjer se nahaja
    lokalni vmesnik].[Ime lokalnega vmesnika];
```

Slika 4.26: Format JNDI reference na WebSphere portalu

kacijski strežnik ne najde. Do tega pride zato, ker aplikacijski strežnik WebSphere pri nameščanju sejnega zrna generira naziv reference (t.i. „JNDI“) na drugačen način kot JBoss, na katerem je potekala referenčna implementacija. Naša aplikacija pa še vedno išče naziv na način, ki je značilen za JBoss (t.j. v formatu „ime EAR/ime sejnega zrna/ime vmesnika“). Da bi zaobšli to težavo, moramo napisati svojo metodo, ki bo generirala pravilen niz za referenco sejnega zrna. V naši aplikaciji se to nahaja v metodi „initManagerProvider()“, znotraj katere se v referenčni implementaciji uporabi privzeti razred „EJBManagerProvider“ iz ogrodja Medius Vaadin Common, znotraj katerega se generira ustrezen niz za referenco sejnega zrna.

Da bi zaobšli ta problem, v spletnem delu aplikacije dodamo nov razred, ki je razširitev razreda „EJBManagerProvider“ in povozimo metodo „initManager“, znotraj katere dodamo izvorno kodo, ki zgradi ustrezen niz za referenco nameščenega sejnega zrna na aplikacijskem strežniku WebSphere, ki je take oblike kot jo prikazuje primer 4.26. Primer implementacije take metode pa prikazuje izvorna koda 4.20.

Po ponovni namestitvi aplikacije nam portlet tokrat pravilno odpre brez očitnih napak. Ker je tabela prazna, poskusimo z dodajanjem novega zapisa, pri čemer naletimo na novo napako, ki je v dnevniških datotekah strežnika vidna kot niz zapisov:

```
<openjpa-2.0.2-SNAPSHOT-r422266:1084264 fatal general
error> org.apache.openjpa.persistence.
PersistenceException: DSRA9110E: Statement is closed.
{prepstmnt 1459312379 SELECT "SEQUENCEVALUE" FROM "
OPENJPA_SEQUENCE_TABLE" WHERE "ID" = ? FOR READ ONLY
WITH RS USE AND KEEP UPDATE LOCKS [params=?]} [code
=0, state=null]
```

Iz zapisov je razvidno, da gre za težavo pri generiranju vrednosti primarnega ključa zapisa v tabeli. Pri generiranju podatkovne baze smo definirali, da se primarni ključ v tabelah generirajo avtomatsko (t.i. tip „Identity“), tako pa smo nastavili tudi anotacije v entitetah na strežniški strani projekta (z ano-

---

**Izvorna koda 4.20** Izvorna koda metode za generiranje ustrezne reference na sejno zrno nameščeno na aplikcijskem strežniku WebSphere.

---

```
try
{
    String lookupName = null;
    String className = clazz.getSimpleName();

    if (className.endsWith("Remote"))
    {
        String basicClassName = className.substring(0,
            className.indexOf("Remote"));
        lookupName = earName + "/" + ejbName + ".jar/" +
            basicClassName + "#" + interfacesPackageName
            + "." + className;
    }
    else if (className.endsWith("Local"))
    {
        String basicClassName = className.substring(0,
            className.indexOf("Local"));
        lookupName = "ejblocal:" + earName + "/" +
            ejbName + ".jar/" + basicClassName + "#" +
            interfacesPackageName + "." + className;
    }
    else
    {
        throw new RuntimeException("EJB_class_and_
            interface_name_does_not_follow_websphere_
            convention.");
    }

    InitialContext context = new InitialContext();
    return (T) context.lookup(lookupName);
}
catch {...}
```

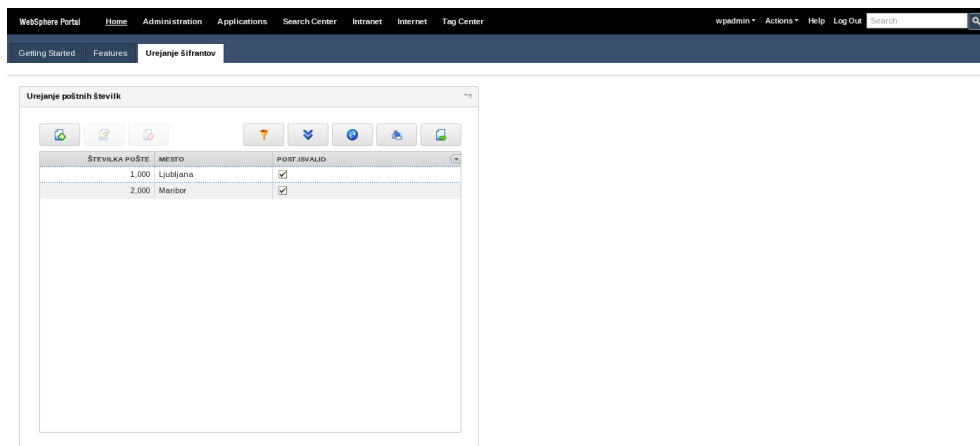
---

```

@Id
@GeneratedValue(strategy=GenerationType.SEQUENCE,
    generator = "sequence")
@SequenceGenerator(name = "sequence", sequenceName="
    SEQ_POST", allocationSize=1) @Column(name="Id_Post"
    , unique=true, nullable=false)
public Integer getId()
{
    ...

```

Slika 4.27: Spremenjene anotacije za generiranje primarnih ključev



Slika 4.28: Portlet za prikaz entitete „Post“ (urejanje poštnih števil) na WebSphere Portal-u.

tacijami „@GeneratedValue(strategy=GenerationType.IDENTITY“)). Kljub intenzivnemu iskanju rešitve za ta problem nisem našel, zato sem poskusil generiranje primarnih ključev zastaviti na drugačen način. Za vsako tabelo sem definiriral posebno sekvenco za generiranje vrednosti primarnih ključev in z orodjem Data Studio ustrezno spremenil nastavitve v tabeli. V entitetah sem za uporabo teh sekvenc popravil anotacije na način, kot prikazuje izvorna koda 4.27.

Tokrat je dodajanje zapisa v tabelo uspelo, pravilno pa je delovalo tudi spreminjanje obstoječega zapisa in brisanje. Slika 4.28 prikazuje portlet za entiteto „Post“.

V tem trenutku imamo uspešno implementiran osnoven portlet, ki vsebuje urejanje preprostega šifranta brez povezanih kompleksnih entitet. Za preostale portlete ponovimo postopek kot je bil opisan uvodoma v tem poglavju.

Med kreiranjem in nameščanjem preostalih portletov lahko naletimo še na dva problema, ki sta vezana na posebnosti WebSphere Portal-a.

Prva posebnost so podrobnosti pri poizvedbah na privzetem ponudniku JPA. Orodje Medius Vaadin Common je zasnovano tako, da so pri nalaganju določene entitete vse njene povezane entitete anotirane z anotacijami, ki vsebujejo naslednji parameter:

```
fetch = FetchType.LAZY
```

To pomeni, da privzeta implementacija metode „getAll()“ v managerjih ob nalaganju entitet iz baze prenese le vrednosti osnovnih parametrov, ne pa tudi vrednosti povezanih entitet.

Primer:

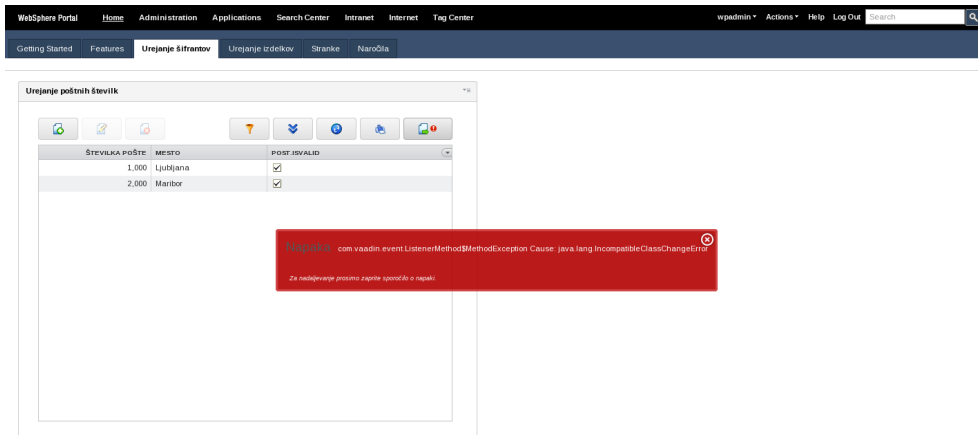
V naši aplikaciji imamo entiteti „Customer“ in „Address“, ki sta povezani z razmerjem ena-proti-mnogo. Metoda „getAll“ zaradi anotacij ob nalaganju prenese le ime, priimek in datum rojstva stranke (entiteta Customer), ne pa tudi seznama njenih naslovov (povezan seznam entitet „Address“).

Taka implementacija je v orodju zato, da se prepreči pojav smrtnih objav v kompleksnih primerih povezav med entitetami. Da bi dobili vrednosti določene povezane entitete (ko jo želimo prikazati na uporabniškem vmesniku), moramo napisati svojo metodo „getAll()“, kjer s posebnim navodilom v poizvedbi povemo, da naj se ob nalaganju osnovne entitete naložijo tudi določene povezane entitete. V referenčni implementaciji (in tudi na portalu Liferay), ki uporablja JPA ponudnika Hibernate, poizvedba za primer entitete „Customer“ izgleda tako:

```
String query = "";  
query += "SELECT c FROM Customer c ";  
query += "LEFT JOIN FETCH c.addresses ca ";  
query += "LEFT JOIN FETCH ca.post p ";  
query += "ORDER BY c.id ASC ";
```

Poizvedba torej vrne vse entitete tipa „Customer“, vključno s seznamom vseh naslovov (entiteta „Address“) in za vsak naslov še pošto na kateri je naslov (entiteta „Post“). Da se to zgodi, moramo poleg „JOIN“ stavka uporabiti še rezervirano besedo „FETCH“ poleg povezane entitete.

Ta poizvedba pa v nespremenjeni obliki ne deluje na ponudniku OpenJPA, ki je na portalu WebSphere. Le-ta namreč ne dovoli, da se večkrat zaporedoma uporabi rezervirana beseda „FETCH“, temveč zahteva bolj natančneje



Slika 4.29: Napaka ob izvozu podatkov iz tabele v format „XLS“.

razdrobljeno poizvedbo. Primer za entiteto „Customer“, ki je funkcionalno enaka prejšnji poizvedbi pa je sledeč:

```
String query = "";
query += "SELECT c FROM Customer c ";
query += "LEFT JOIN FETCH c.addresses ";
query += "LEFT JOIN c.addresses ca ";
query += "LEFT JOIN FETCH ca.post ";
query += "ORDER BY c.id ASC ";
```

Za razliko od ponudnika „Hibernate“ moramo torej za vsako povezano entiteto najprej določiti „JOIN“ stavek, šele nato lahko uporabimo tudi rezervirano besedo „FETCH“.

Druga posebnost pa je problem, do katerega pridemo, ko želimo izvoziti podatke iz tabel v format „XLS“ za delo v programih za preglednice. Ko želimo to storiti, nam na spletnem vmesniku aplikacija javi napako, kot to prikazuje slika 4.29.

Če pogledamo v dnevniške datoteke strežnika, najdemo naslednjo napako:

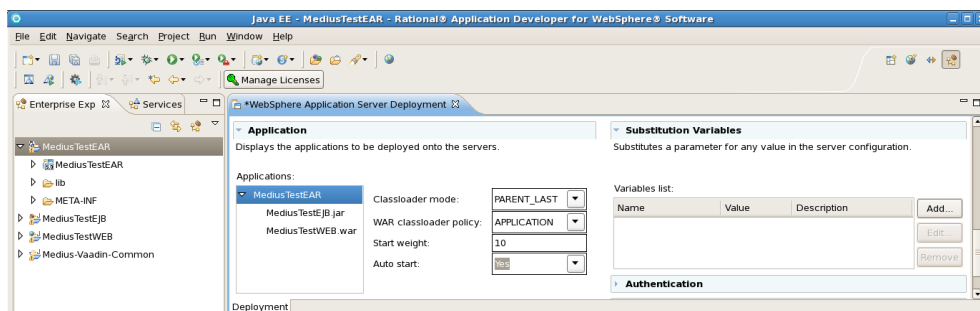
```
Caused by: java.lang.IncompatibleClassChangeError
    at si.medius.common.export.ExportXLSAction.
        generateContent(ExportXLSAction.java:84)
```

Izkaže se, da gre za problem v načinu, kako aplikacijski strežnik WebSphere nalaga posamezne programske knjižnice, ki jih vsebuje naš projekt. Če naleti na neko programsko knjižnico, ki jo vsebuje že sam aplikacijski strežnik, bo v času izvajanja aplikacije (ko bo ta zahtevala določene funkcionalnosti te

knjižnice) uporabil že nameščeno knjižnico. Do problema pride, ko je različica na aplikacijskem strežniku starejša od različice, ki je dodana projektu. V tem primeru WebSphere ne zna razločiti, da gre pravzaprav za drugačno različico in kljub temu aplikaciji ponudi različico, ki je nameščena na aplikacijskem strežniku, aplikacija pa zazna, da gre za nekompatibilno različico in sproži napako.

Temu se lahko izognemo na več načinov. Prvi način je, da uporabljamo enake različice knjižnic, kot so nameščene na WebSphere aplikacijskem strežniku, vendar se v tem primeru odpovemo funkcionalnostim, ki jih mogoče ponujajo novejšje različice. Zato je bolje, da spremenimo obnašanje strežnika na ta način, da bo najprej naložil knjižnice, ki jih vsebuje projekt, šele nato pa svoje lastne. To lahko na aplikacijskem strežniku nastavimo za vsako aplikacijo posebej in sicer na dva načina:

- Nastavitev na že nameščeni aplikaciji na aplikacijskem strežniku.  
Aplikacijski strežnik omogoča, da v nastavitvah nameščenih aplikacij spremenimo način nalaganja knjižnic za določeno aplikacijo. To storimo tako, da se prijavimo v administracijsko konzolo strežnika in v meniju izberemo postavko „Applications“ in nato „WebSphere enterprise applications“. Aplikacijski strežnik nato pokaže seznam nameščenih aplikacij, v kateri v našem primeru izberemo „MediusTestEAR“, aplikacijski strežnik pa nam nato pokaže možne nastavitve za to aplikacijo. Izmed ponujenih možnosti izberemo „Class loading and update detection“, kjer vrednost parametra „Class loader order“ nastavimo iz „Classes loaded with parent class loader first“ na „Classes loaded with local class loader first (parent last)“. S tem strežniku povemo, naj da prednost programskim knjižnicam, ki so nameščene v samem projektu. S to nastavitvijo smo nastavili obnašanje v korenem projektu, lahko pa se zgodi, da ima spletni del projekta („MediusTestWEB“) še posebej nastavljeno obnašanje nalaganja programskih knjižnic, kot se je izkazalo tudi v našem primeru. Zato se vrnemo nazaj na nastavitve projekta „MediusTestEAR“ in izberemo možnost „Manage Modules“. Strežnik prikaže strežniški („MediusTestEJB“) in spletni del projekta („MediusTestWEB“), katerega tudi izberemo. Nato izmed ponujenih parametrov nastavimo parameter „Class loader order“ na enak način kot v korenem projektu. Vse nastavitve še shranimo s pritiskom na gumb „Apply“ in potrdimo njihovo takojšnjo uveljavitev s pritiskom možnosti „Save to master configuration“, ki jo ponudi aplikacijski strežnik. Po ponovnem zagonu aplikacije poskusimo z izvozom podatkov v format „XLS“, ki tokrat uspe, kot prikazuje slika 4.31. Slabost te rešitve je v tem, da



Slika 4.30: Nastavitev zaporedja nalaganja programskih knjižnic v orodju Rational Application Developer.

če v razvojnem okolju spremenimo aplikacijo in jo ponovno namestimo, te nastavitve izgubimo in jih moramo ponovno uveljaviti. To slabost odpravlja drugi način nastavitve.

- Nastavitev aplikacije v razvojnem okolju.

Če želimo, da se način nalaganja programskih knjižnic avtomatsko nastavi ob vsakem nalaganju na strežnik, moramo v razvojnem okolju Rational Application Developer nastaviti naš projekt. To storimo tako, da se z miško premaknemo na ime projekta „MediusTestEAR“, pritisnemo desno tipko in izberemo možnost „JavaEE“ in nato „Open WebSphere Application Server Deployment“. Odpre se seznam nastavitve za naš projekt v katerem poiščemo postavko „Application“, znotraj nje pa izberemo ime korenskega projekta „MediusTestEAR“ in poiščemo nastavitve „Class loader mode“, ki jo nastavimo na „PARENT\_LAST“, kot to prikazuje slika 4.30.

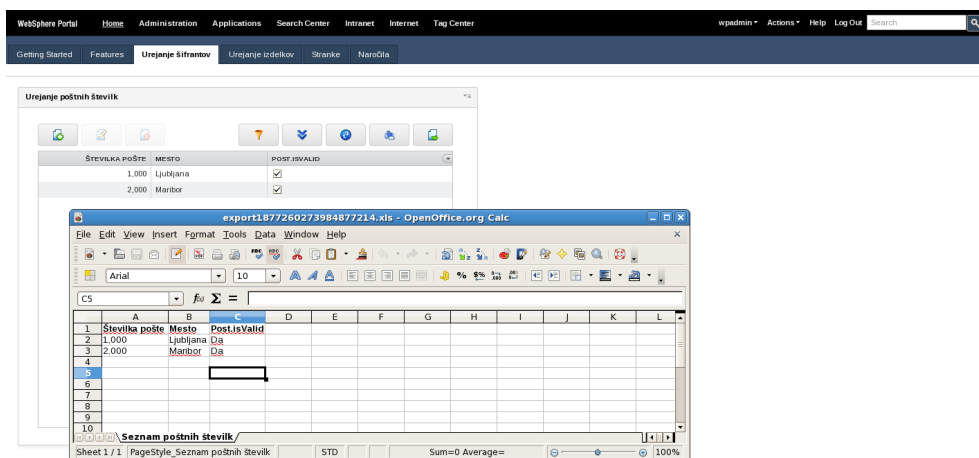
Ker korenski imenik vsebuje tudi referenco na spletni del, lahko na istem mestu in na enak način nastavimo tudi spletni del projekta. Nastavitve shranimo ter ponovno izvozimo in namestimo aplikacijo na aplikacijski strežnik. Ker so nastavitve zdaj shranjene kot del projekta, se bodo avtomatsko ohranile z vsako namestitvijo na strežnik.

S tem smo tudi uspešno zaključili z integracijo ogrodja Medius Vaadin Common na portal.

#### 4.4.4 Analiza delovanja

- Podatkovni nivo

Za pripravo in izvedbo skripte, ki je generirala podatkovno bazo, defi-



Slika 4.31: Uspešen izvoz podatkov v „XLS“ datoteko na WebSphere portalu.

nirano na referenčnem primeru, so bile potrebne le manjše spremembe. Poleg tega se je izkazalo, da je za kasnejše delovanje s privzetim ponudnikom JPA potrebno kreirati sekvence za generiranje primarnih ključev, saj preprostejša možnost s stolpci tipa „Identity“ ne deluje. Kljub temu je po kreiranih sekvencah povezava s podatkovno bazo ter izvajanje poizvedb potekalo pravilno, zato lahko integracijo na tem nivoju označimo kot uspešno.

- Poslovni in spletni nivo  
Izpolnjevanje predhodno definiranih kriterijev:
  - *delujoči podatkovni viri na aplikacijskem strežniku*  
Podatkovni viri so bili definirani prek spletnega vmesnika administracijske konzole na aplikacijskem strežniku. V primerjavi z aplikacijskim strežnikom JBoss, na katerem je tekel Liferay, je nastavev tu nekoliko bolj zapletena, saj je za delovanje potrebno nastaviti več podrobnosti. Kljub temu pa je nastavev uspela in podatkovni viri so delovali brez težav.
  - *delujoča implementacija JPA (dodajanje, urejanje, brisanje podatkov v samo podatkovno bazo ter pridobivanje podatkov iz baze)*  
Privzeta implementacija OpenJPA deluje pravilno. Da pa je bilo moč uporabiti specifikacijo JPA 2.0, smo morali opraviti kar precej dela, saj aplikacijski strežnik privzeto te specifikacije ne podpira

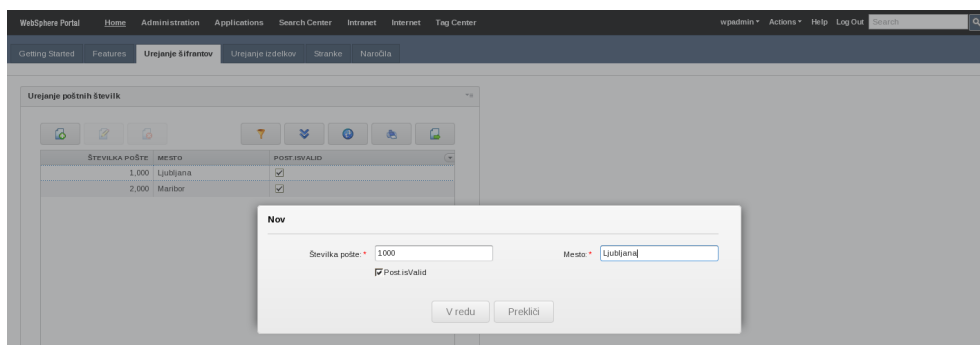
in je bila zato potrebna namestitev dodatne funkcionalnosti „WebSphere Application Server V7 Feature Pack for OSGi Applications and Java Persistence API 2.0“. Po namestitvi pa s samo implementacijo ponudnika JPA 2.0 ni bilo težav.

- *količina potrebnih sprememb na strežniški kodi aplikacije*  
Na strežniški kodi spremembe so bile zaradi posebnosti privzete implementacije OpenJPA potrebne manjše spremembe pri poizvedbah, ki vsebujejo povezane entitete.
- *količina potrebnih sprememb na spletnem delu aplikacije*  
Potrebne so bile dodatne nastavitve servlet-ov, saj se je izkazalo, da Vaadin ne deluje pravilno s specifikacijo JSR-286, zato je bilo potrebno uporabiti nekoliko starejšo različico, ki pa za svoje delovanje potrebuje tudi servlet-e. Za pravilno delovanje je bilo uporabljeno razvojno okolje Rational Application Developer, ki je avtomatsko poskrbelo za dodajanje nekaterih namestitvenih deskriptorjev za spletni del projekta, zato večjih težav tu ni bilo. Prav tako v primerjavi s portalom Liferay ni bilo potrebno ročno kopiranje tem in komponent na strežnik, ampak je strežnik le-te sam prepoznal in pravilno uporabil.
- *ustrezne nastavitve za prikaz pogledov v portletih na portalu*  
Uporabljen je bil enak vtičnik za delo z Vaadinom, kot pri integraciji na Liferay, pri čemer je bilo potem potrebno izvesti nekaj manjših ročnih nastavitvev deskriptorjev, ki pa so bile enostavne.
- *kompleksnost vzpostavitve osnovnega portala*  
Vzpostavitev osnovnega portala je bila zelo kompleksna. Izbrati je bilo potrebno kompatibilen operacijski sistem in izvesti več dodatnih nastavitvev že pred samo namestitvijo. Sama namestitev je potekala brez težav, kljub temu pa je količina dela, da prideš do delujočega osnovnega portala v primerjavi s portalom Liferay bistveno večja. Poudariti je potrebno tudi to, da različica WebSphere Portal 7.0 Express Edition deluje le tri mesece, nato pa je potrebno kupiti licenco.

- Nivo odjemalca

Na nivoju odjemalca preverimo delovanje aplikacije v naslednjih točkah:

- *ustrezen prikaz osnovne aplikacije, vključno z vsemi grafičnimi elementi (slike, barve, preliv, komponente grafičnega vmesnika...)*



Slika 4.32: Dodajanje entitete v primeru sejnega zrna „Post“ na WebSphere portalu.

Vsi portleti (vse slike, komponente, pravilne barve) so brezhibno prikazani.

- *delovanje dodajanja, urejanja, brisanja entitet na uporabniških maskah (vključno z vsemi razmerji do ostalih entitetnih zrn):*

- \* entitete šifrantov

Dodajanje entitet pri šifrantih (npr. na entitetnem zrnju „Post“) ter njihovo urejanje in brisanje deluje brezhibno. Slika 4.32 prikazuje primer dodajanja entitete.

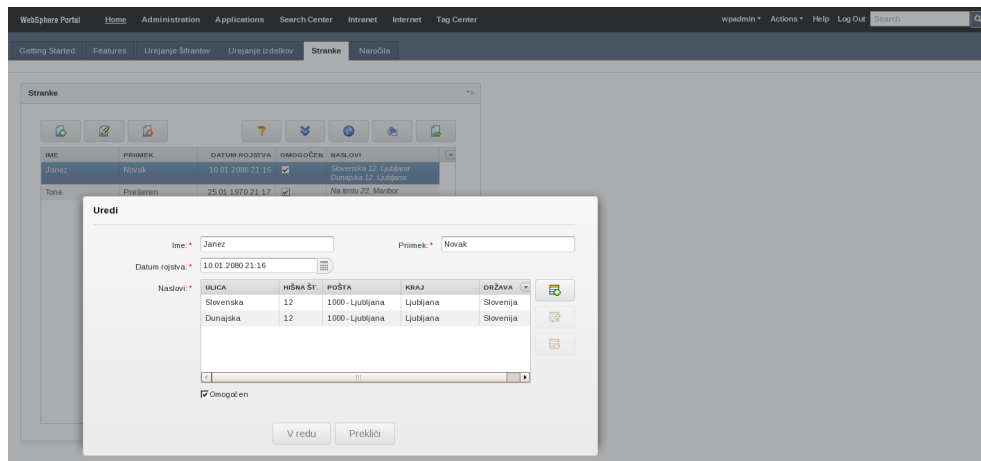
- \* entitete pri razmerju ena proti mnogo:

Obnašanje entitet pri razmerju ena proti mnogo lahko preverimo pri razmerju entitet „Customer“ in „Address“ (ena stranka ima lahko več naslovov). Tudi to razmerje na portalu WebSphere deluje pravilno. Slika 4.33 prikazuje urejanje entitete tipa „Customer“, ki ima lahko več entitet tipa „Address“.

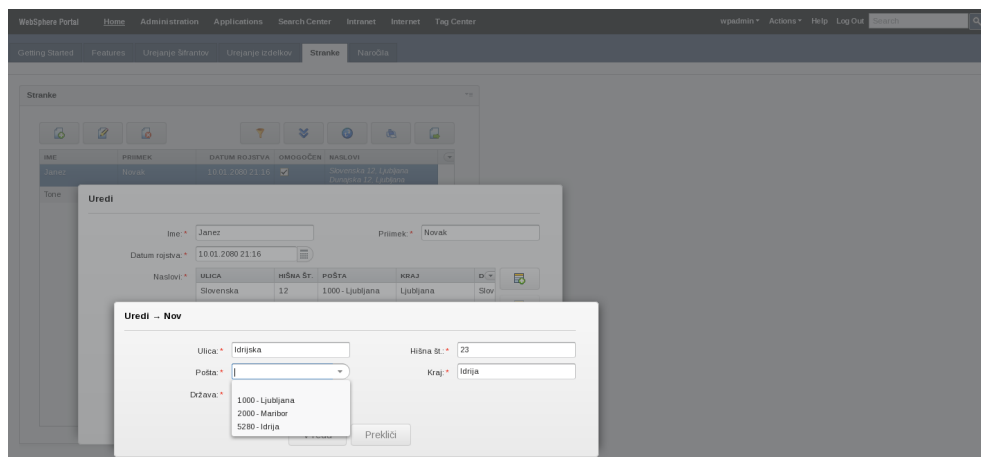
- \* entitete pri razmerju mnogo proti ena:

To razmerje imamo pri entitetnih zrnih „Address“ in „Post“. Ko stranki dodajamo naslove, moramo pri vsakem naslovu izbrati eno pošto (torej več naslovov ima lahko isto pošto), kar v uporabniškem vmesniku na WebSphere portalu ustrezno vidimo kot spustni meni, kjer lahko izberemo eno vrednost. Primer prikazuje slika 4.34

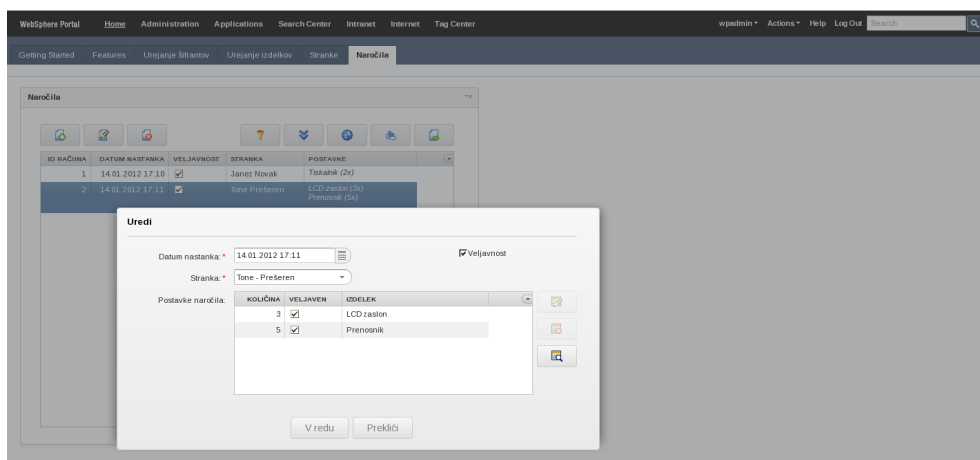
- \* entitete pri razmerju mnogo proti mnogo



Slika 4.33: Primer urejanja entitete tipa „Address“ na WebSphere portalu.



Slika 4.34: Izbira pošte iz spustnega menija na WebSphere portalu.



Slika 4.35: Primer razmerja mnogo proti mnogo pri entitetah tipa „Order“ in „Product“ na WebSphere portalu.

To razmerje lahko opazujemo pri entitetnih zrnih „Order“ in „Product“ (naročilo ima lahko več postavk izdelkov, izdelek pa je lahko na postavkah različnih naročil). Tudi to razmerje na portalu WebSphere deluje pravilno, kar prikazuje slika 4.35.

- *delovanje sortiranja, filtriranja v tabelah*  
Sortiranje in filtriranje deluje pravilno.
- *delovanje izvoza v datoteke PDF in Microsoft Excel*  
Izvoz po vnešenih spremembah glede nalaganja programskih knjižnic deluje pravilno.
- *pravilen prikaz in delovanje portleta pri različnih stanjih okna (minimizacija, maksimizacija in klasičen prikaz)*  
Portlet pravilno deluje v vseh različnih stanjih.

Na koncu lahko zaključimo, da je WebSphere Portal različice 7.0 primeren za integracijo ogrodja Medius Vaadin Common, saj vse funkcionalnosti delujejo. Je pa v primerjavi z integracijo na portal Liferay integracija na WebSphere portal bistveno bolj zahtevna, kar pa je bilo pričakovano zaradi razlik v delovanju aplikacijskih strežnikov JBoss in WebSphere.

## Poglavje 5

# Zaključek

V obeh primerih se je izkazalo, da je ogrodje Medius Vaadin Common primerno za integracijo v obliki portletov na portal, zato lahko zaključimo, da je bil poskus integracije uspešen.

Da smo to dosegli, smo morali na posameznih arhitekturnih nivojih izvesti razmeroma malo prilagoditev. V tem kontekstu je posebej pomembno, da so bile te prilagoditve vezane skoraj izključno na posebnosti v povezavi s ciljno podatkovno bazo in ciljnim aplikacijskim strežnikom. Pri tem so bile praktično vse prilagoditve potrebne zaradi različno implementiranih podrobnosti v povezavi s konfiguracijo podatkovnih virov, konfiguracijo namestitvenih deskriptorjev portletov za portale in vzpostavitevijo pravilno pripravljenega operacijskega sistema in na njem nameščene osnovne postavitve portala. To pomeni, da jedra našega ogrodja ni bilo potrebno spreminjati, kar je posledica temeljitega načrtovanja ogrodja in sledenje uveljavljenim standardom in priporočilom pri razvoju ogrodja na platformi Java EE.

Na osnovi izkušenj, ki smo jih pridobili pri poskusu integracije lahko tudi vidimo, da so različne implemetacije standardov povezanih z Javo EE (JPA 2.0, EJB 3.0, JSR-286) pri posameznih ponudnikih aplikacijskih sistemov dovolj dobre, da je mogoče na njihovi osnovi uspešno načrtovati in implementirati kompleksne informacijske sisteme.

*Liferay in Liferay Portal sta zaščiteni znamki podjetja Liferay, Inc.*

*JBoss in JBoss Application Server sta zaščiteni znamki podjetja Red Hat, Inc.*

*IBM, IBM WebSphere, IBM WebSphere Portal, IBM WebSphere Application Server, IBM Rational Application Developer in IBM Data Studio so zaščitene znamke podjetja IBM Corporation.*

*Vse ostale blagovne znamke so last pripadajočih podjetij.*

# Dodatek A

## Izvorna koda referenčne implementacije

### A.1 Skripta za izdelavo podatkovne baze MySQL

```
/*  
=====
```

---

```
*/  
/* DBMS name:      MySQL 5.0  
/* Created on:      14.11.2011 9:25:30 */  
*/  
=====
```

---

```
*/  
  
drop table if exists Address;  
  
drop table if exists Customer;  
  
drop table if exists 'Order';  
  
drop table if exists OrderItem;  
  
drop table if exists Post;
```

```
drop table if exists Product;
```

```
/*
```

---

```
*/
```

```
/* Table: Address
```

```
*/
```

```
/*
```

---

```
*/
```

```
create table Address
```

```
(
```

```
    Id_Address          int not null,
```

```
    Id_Post             int not null,
```

```
    Id_Customer         int,
```

```
    Street              varchar(500) not null,
```

```
    HouseNumber         varchar(500) not null,
```

```
    City_Address        varchar(500),
```

```
    State               varchar(500) not null,
```

```
    Valid_Address       bool,
```

```
    primary key (Id_Address)
```

```
);
```

```
/*
```

---

```
*/
```

```
/* Table: Customer
```

```
*/
```

```
/*
```

---

```
*/
```

```
create table Customer
```

```
(
```

```
    Id_Customer         int not null,
```

```
    FirstName           varchar(500) not null,
```

```
    SecondName          varchar(500) not null,
```

```
    DateOfBirth         timestamp not null,
```

```
    Valid_Customer      bool,
```

```
    primary key (Id_Customer)
```

```
);
```

```
/*
```

```
    */  
/* Table: 'Order'
```

```
    */
```

```
/*
```

```
    */  
create table 'Order'
```

```
(  
    Id_Order          int not null,  
    Id_Customer       int not null,  
    DateCreated       timestamp not null,  
    DateConcluded     timestamp,  
    DatePaid          timestamp,  
    Valid_Order       bool not null,  
    primary key (Id_Order)
```

```
);
```

```
/*
```

```
    */  
/* Table: OrderItem
```

```
    */
```

```
/*
```

```
    */  
create table OrderItem
```

```
(  
    Id_OrderItem      int not null,  
    Id_Order          int not null,  
    Id_Product        int not null,  
    Quantity          int not null,  
    Valid_OrderItem   bool not null,  
    primary key (Id_OrderItem)
```

```
);
```

```
/*
=====
*/
/* Table: Post
*/
*/
=====
*/
create table Post
(
    Id_Post          int not null,
    Number           int not null,
    CityPost         varchar(500) not null,
    Valid_Post       bool not null,
    primary key (Id_Post)
);

/*
=====
*/
/* Table: Product
*/
*/
=====
*/
create table Product
(
    Id_Product       int not null,
    'Label'          varchar(500) not null,
    QuantityInStock  int not null,
    QuantityReserved int,
    Valid_Product    bool not null,
    primary key (Id_Product)
);

alter table Address add constraint
    FK_Relationship_AddressPost foreign key (Id_Post)
    references Post (Id_Post) on delete restrict on
    update restrict;
```

```
alter table Address add constraint
    FK_Relationship_CustomerAddress foreign key (
        Id_Customer)
        references Customer (Id_Customer) on delete
            restrict on update restrict;

alter table 'Order' add constraint
    FK_Relationship_CustomerOrder foreign key (
        Id_Customer)
        references Customer (Id_Customer) on delete
            restrict on update restrict;

alter table OrderItem add constraint
    FK_Relationship_OrderItemProduct foreign key (
        Id_Product)
        references Product (Id_Product) on delete restrict
            on update restrict;

alter table OrderItem add constraint
    FK_Relationship_OrderOrderItem foreign key (Id_Order)
        references 'Order' (Id_Order) on delete restrict
            on update restrict;
```

## A.2 Izvorna koda entitetnih zrn

### A.2.1 Entitetno zrno Post

```
package si.medius.test.entity;

import java.io.Serializable;
import java.util.List;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
```

```
import si.medius.common.annotation.editor.  
    FieldPresentation;  
import si.medius.common.annotation.lookup.  
    LookupPresentation;  
import si.medius.common.annotation.table.  
    ColumnPresentation;  
import si.medius.test.managers.local.PostManagerLocal;  
  
@Entity  
@Table(name = "Post")  
@LookupPresentation(format = "#{number} - #{city}",  
    managerClass = PostManagerLocal.class)  
public class Post implements Serializable  
{  
    private Integer id;  
    private Integer number;  
    private String city;  
  
    private List<Address> addresses;  
  
    public Post()  
    {  
        super();  
    }  
  
    public Post(Integer id, Integer number, String city)  
    {  
        super();  
        this.id = id;  
        this.number = number;  
        this.city = city;  
    }  
  
    @OneToMany(mappedBy = "post")  
    public List<Address> getAddresses()  
    {  
        return addresses;  
    }  
}
```

```
public void setAddresses(final List<Address>
    addresses)
{
    this.addresses = addresses;
}

@Id
@GeneratedValue
public Integer getId()
{
    return id;
}

public void setId(Integer id)
{
    this.id = id;
}

@ColumnPresentation(order = 1, width = 150)
@FieldPresentation(order = 1, required = true)
public Integer getNumber()
{
    return number;
}

public void setNumber(Integer number)
{
    this.number = number;
}

@ColumnPresentation(order = 2)
@FieldPresentation(order = 2, required = true)
public String getCity()
{
    return city;
}

public void setCity(String city)
{

```

```
        this.city = city;
    }

    @Override
    public int hashCode()
    {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((city == null) ? 0 :
            city.hashCode());
        result = prime * result + ((id == null) ? 0 : id
            .hashCode());
        result = prime * result + ((number == null) ? 0
            : number.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj)
    {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Post other = (Post) obj;
        if (city == null)
        {
            if (other.city != null)
                return false;
        }
        else if (!city.equals(other.city))
            return false;
        if (id == null)
        {
            if (other.id != null)
                return false;
        }
    }
```

```

        else if (!id.equals(other.id))
            return false;
        if (number == null)
        {
            if (other.number != null)
                return false;
        }
        else if (!number.equals(other.number))
            return false;
        return true;
    }
}

```

### A.2.2 Entitetno zrno Address

```

package si.medius.test.entity;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import javax.persistence.Transient;

import si.medius.common.annotation.editor.
    FieldPresentation;
import si.medius.common.annotation.lookup.
    LookupPresentation;
import si.medius.common.annotation.options.FieldType;
import si.medius.common.annotation.table.
    ColumnPresentation;
import si.medius.test.managers.local.AddressManagerLocal
    ;

@Entity
@Table(name = "Address")

```

```
@LookupPresentation(format = "#{street}_#{houseNumber},_
    #{city}", managerClass = AddressManagerLocal.class)
public class Address implements Serializable
{
    private Integer id;
    private String street;
    private String houseNumber;
    private String city;
    private String state;

    private Post post;
    private Customer customer;

    private String post_string;

    public Address()
    {
        super();
    }

    public Address(Integer id, String street, String
        houseNumber, String city, String state, Post post
        , Customer customer, String post_string)
    {
        super();
        this.id = id;
        this.street = street;
        this.houseNumber = houseNumber;
        this.city = city;
        this.state = state;
        this.post = post;
        this.customer = customer;
        this.post_string = post_string;
    }

    @Id
    @GeneratedValue
    public Integer getId()
    {
```

```
        return id;
    }

    public void setId(Integer id)
    {
        this.id = id;
    }

    @ColumnPresentation(order = 2, width = 100)
    @FieldPresentation(order = 2, required = true)
    public String getStreet()
    {
        return street;
    }

    public void setStreet(String street)
    {
        this.street = street;
    }

    @ColumnPresentation(order = 5, width = 100)
    @FieldPresentation(order = 5, required = true)
    public String getCity()
    {
        return city;
    }

    public void setCity(String city)
    {
        this.city = city;
    }

    @ColumnPresentation(order = 6, width = 80)
    @FieldPresentation(order = 6, required = true)
    public String getState()
    {
        return state;
    }
}
```

```
public void setState(String state)
{
    this.state = state;
}

// @ColumnPresentation(order = 4, followOnLevel =
//     LevelType.SIMPLE)
@FieldPresentation(order = 4, required = true, type
    = FieldType.LOOKUP_SIMPLE)
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "fk_idPost", referencedColumnName
    = "Id", nullable = false)
public Post getPost()
{
    return post;
}

public void setPost(Post post)
{
    this.post = post;
}

@ColumnPresentation(order = 4, width = 100)
@Transient
public String getPost_string()
{
    return this.post.getNumber() + "└─┘" + this.post
        .getCity();
}

public void setPost_string(String post_string)
{
    this.post_string = post_string;
}

@ColumnPresentation(order = 3, width = 50)
@FieldPresentation(order = 3, required = true)
public String getHouseNumber()
{

```

```

        return houseNumber;
    }

    public void setHouseNumber(String houseNumber)
    {
        this.houseNumber = houseNumber;
    }

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "fk_idCustomer",
        referencedColumnName = "Id", nullable = true)
    public Customer getCustomer()
    {
        return customer;
    }

    public void setCustomer(Customer customer)
    {
        this.customer = customer;
    }

    @Override
    public int hashCode()
    {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((city == null) ? 0 :
            city.hashCode());
        result = prime * result + ((houseNumber == null)
            ? 0 : houseNumber.hashCode());
        result = prime * result + ((id == null) ? 0 : id
            .hashCode());
        result = prime * result + ((post == null) ? 0 :
            post.hashCode());
        result = prime * result + ((post_string == null)
            ? 0 : post_string.hashCode());
        result = prime * result + ((state == null) ? 0 :
            state.hashCode());
        result = prime * result + ((street == null) ? 0

```

```
        : street.hashCode());
    return result;
}

@Override
public boolean equals(Object obj)
{
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Address other = (Address) obj;
    if (city == null)
    {
        if (other.city != null)
            return false;
    }
    else if (!city.equals(other.city))
        return false;
    if (houseNumber == null)
    {
        if (other.houseNumber != null)
            return false;
    }
    else if (!houseNumber.equals(other.houseNumber))
        return false;
    if (id == null)
    {
        if (other.id != null)
            return false;
    }
    else if (!id.equals(other.id))
        return false;
    if (post == null)
    {
        if (other.post != null)
            return false;
    }
}
```

```

    }
    else if (!post.equals(other.post))
        return false;
    if (post_string == null)
    {
        if (other.post_string != null)
            return false;
    }
    else if (!post_string.equals(other.post_string))
        return false;
    if (state == null)
    {
        if (other.state != null)
            return false;
    }
    else if (!state.equals(other.state))
        return false;
    if (street == null)
    {
        if (other.street != null)
            return false;
    }
    else if (!street.equals(other.street))
        return false;
    return true;
}

}

```

### A.2.3 Entitetno zrno Customer

```

package si.medius.test.entity;

import java.io.Serializable;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;

import javax.persistence.Entity;
import javax.persistence.FetchType;

```

```

import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;

import si.medius.common.annotation.editor.
    FieldPresentation;
import si.medius.common.annotation.lookup.
    LookupPresentation;
import si.medius.common.annotation.options.FieldType;
import si.medius.common.annotation.table.
    ColumnPresentation;
import si.medius.test.managers.local.
    CustomerManagerLocal;

@Entity
@Table(name = "Customer")
@LookupPresentation(format = "#{firstName} --#{
    secondName}", managerClass = CustomerManagerLocal.
    class)
public class Customer implements Serializable
{
    private Integer id;
    private String firstName;
    private String secondName;
    private Date dateOfBirth;
    private Boolean valid = true;

    private Set<Address> addresses = new HashSet<Address
        >();

    public Customer()
    {
        super();
    }

    public Customer(Integer id, String firstName, String
        secondName, Date dateOfBirth, Boolean valid, Set
        <Address> addresses)

```

```
{
    super();
    this.id = id;
    this.firstName = firstName;
    this.secondName = secondName;
    this.dateOfBirth = dateOfBirth;
    this.valid = valid;
    this.addresses = addresses;
}

@Id
@GeneratedValue
public Integer getId()
{
    return id;
}

public void setId(Integer id)
{
    this.id = id;
}

@ColumnPresentation(order = 1, width = 100)
@FieldPresentation(order = 1, required = true)
public String getFirstName()
{
    return firstName;
}

public void setFirstName(String firstName)
{
    this.firstName = firstName;
}

@ColumnPresentation(order = 2, width = 100)
@FieldPresentation(order = 2, required = true)
public String getSecondName()
{
    return secondName;
}
```

```
}

public void setSecondName(String secondName)
{
    this.secondName = secondName;
}

@ColumnPresentation(order = 3, width = 100)
@FieldPresentation(order = 3, required = true)
public Date getDateOfBirth()
{
    return dateOfBirth;
}

public void setDateOfBirth(Date dateOfBirth)
{
    this.dateOfBirth = dateOfBirth;
}

@ColumnPresentation(order = 5, width = 60)
@FieldPresentation(order = 5, required = true)
public Boolean getValid()
{
    return valid;
}

public void setValid(Boolean valid)
{
    this.valid = valid;
}

// @FieldPresentation(order = 4, required = true,
//     type = FieldType.LOOKUP_SIMPLE)
// @ManyToOne(fetch = FetchType.LAZY)
// @JoinColumn(name = "fk_idAddress",
//     referencedColumnName = "Id", nullable = false)
// public Address getAddress()
// {
//     return address;
// }
```

```

// }
//
// public void setAddress(Address address)
// {
//     this.address = address;
// }
//
// @ColumnPresentation(order = 4)
// @Transient
// public String getAddressString()
// {
//     String s = "";
//     s += address.getStreet();
//     s += " ";
//     s += address.getHouseNumber();
//     s += ", ";
//     s += address.getCity();
//
//     return s;
// }
//

@FieldPresentation(order = 4, required = true, type
    = FieldType.LOOKUP_MANY, lookupClass = Address.
    class)
@OneToMany(fetch = FetchType.LAZY, mappedBy = "
    customer")
public Set<Address> getAddresses()
{
    return addresses;
}

public void setAddresses(Set<Address> addresses)
{
    this.addresses = addresses;
}

@Override
public int hashCode()

```

```
{
    final int prime = 31;
    int result = 1;
    result = prime * result + ((addresses == null) ?
        0 : addresses.hashCode());
    result = prime * result + ((dateOfBirth == null)
        ? 0 : dateOfBirth.hashCode());
    result = prime * result + ((firstName == null) ?
        0 : firstName.hashCode());
    result = prime * result + ((id == null) ? 0 : id
        .hashCode());
    result = prime * result + ((secondName == null)
        ? 0 : secondName.hashCode());
    result = prime * result + ((valid == null) ? 0 :
        valid.hashCode());
    return result;
}
```

@Override

```
public boolean equals(Object obj)
{
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Customer other = (Customer) obj;
    if (addresses == null)
    {
        if (other.addresses != null)
            return false;
    }
    else if (!addresses.equals(other.addresses))
        return false;
    if (dateOfBirth == null)
    {
        if (other.dateOfBirth != null)
            return false;
    }
}
```

```

    }
    else if (!dateOfBirth.equals(other.dateOfBirth))
        return false;
    if (firstName == null)
    {
        if (other.firstName != null)
            return false;
    }
    else if (!firstName.equals(other.firstName))
        return false;
    if (id == null)
    {
        if (other.id != null)
            return false;
    }
    else if (!id.equals(other.id))
        return false;
    if (secondName == null)
    {
        if (other.secondName != null)
            return false;
    }
    else if (!secondName.equals(other.secondName))
        return false;
    if (valid == null)
    {
        if (other.valid != null)
            return false;
    }
    else if (!valid.equals(other.valid))
        return false;
    return true;
}
}

```

#### A.2.4 Entitetno zrno Order

```
package si.medius.test.entity;
```

```

import java.io.Serializable;
import java.util.Date;
import java.util.LinkedList;
import java.util.List;

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.Transient;

import si.medius.common.annotation.editor.
    FieldPresentation;
import si.medius.common.annotation.lookup.
    LookupPresentation;
import si.medius.common.annotation.options.FieldType;
import si.medius.common.annotation.table.
    ColumnPresentation;
import si.medius.test.managers.local.OrderManagerLocal;

@Entity
@Table(name = "Order")
@LookupPresentation(format = "#{id}_({#{customer.
    firstName}_-#{customer.secondName})" , managerClass =
    OrderManagerLocal.class)
public class Order implements Serializable
{

    private Integer id;
    private Date dateCreated = new Date();
    private Date dateConcluded;
    private Date datePayed;
    private Boolean valid = true;

    private Customer customer;

```

```
private String customerString;

private List<OrderItem> orderItems = new LinkedList<
    OrderItem>();

public Order()
{
    super();
}

public Order(Integer id, Date dateCreated, Date
    dateConcluded, Date datePayed, Boolean valid,
    Customer customer)
{
    super();
    this.id = id;
    this.dateCreated = dateCreated;
    this.dateConcluded = dateConcluded;
    this.datePayed = datePayed;
    this.valid = valid;
    this.customer = customer;
}

@Id
@GeneratedValue
@ColumnPresentation(order = 1, width = 60)
public Integer getId()
{
    return id;
}

public void setId(Integer id)
{
    this.id = id;
}

@ColumnPresentation(order = 2, width = 100)
@FieldPresentation(order = 1, required = true)
```

```
public Date getDateCreated()
{
    return dateCreated;
}

public void setDateCreated(Date dateCreated)
{
    this.dateCreated = dateCreated;
}

public Date getDateConcluded()
{
    return dateConcluded;
}

public void setDateConcluded(Date dateConcluded)
{
    this.dateConcluded = dateConcluded;
}

public Date getDatePayed()
{
    return datePayed;
}

public void setDatePayed(Date datePayed)
{
    this.datePayed = datePayed;
}

@ColumnPresentation(order = 3, width = 70)
@FieldPresentation(order = 2, required = true)
public Boolean getValid()
{
    return valid;
}

public void setValid(Boolean valid)
{

```

```
        this.valid = valid;
    }

    @FieldPresentation(order = 3, required = true, type
        = FieldType.LOOKUP_SIMPLE)
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "fk_idCustomer",
        referencedColumnName = "Id", nullable = false)
    public Customer getCustomer()
    {
        return customer;
    }

    public void setCustomer(Customer customer)
    {
        this.customer = customer;
    }

    @Transient
    @ColumnPresentation(order = 4, width = 100)
    public String getCustomerString()
    {
        if (customer != null)
        {
            return customer.getFirstName() + " " +
                customer.getSecondName();
        }
        else
        {
            return "";
        }
    }

    public void setCustomerString(String customerString)
    {
        this.customerString = customerString;
    }

    @OneToMany(mappedBy = "order", fetch = FetchType.
```

```
    LAZY)
    @FieldPresentation(order = 4, manyToManyBackProperty
        = "order", type = FieldType.LOOKUP_MANYTOMANY,
        manyToManyProperty = "product", manyToManyClass =
            OrderItem.class, lookupClass = Product.class)
    public List<OrderItem> getOrderItems()
    {
        return orderItems;
    }

    public void setOrderItems(List<OrderItem> orderItems
        )
    {
        this.orderItems = orderItems;
    }

    @Override
    public int hashCode()
    {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((customer == null) ?
            0 : customer.hashCode());
        result = prime * result + ((dateConcluded ==
            null) ? 0 : dateConcluded.hashCode());
        result = prime * result + ((dateCreated == null)
            ? 0 : dateCreated.hashCode());
        result = prime * result + ((datePayed == null) ?
            0 : datePayed.hashCode());
        result = prime * result + ((id == null) ? 0 : id
            .hashCode());
        result = prime * result + ((valid == null) ? 0 :
            valid.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj)
    {
```

```

        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Order other = (Order) obj;
        if (id == null)
        {
            if (other.id != null)
                return false;
        }
        else if (!id.equals(other.id))
            return false;
        return true;
    }
}

```

### A.2.5 Entitetno zrno OrderItem

```

package si.medius.test.entity;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import javax.persistence.Transient;

import si.medius.common.annotation.editor.
    FieldPresentation;
import si.medius.common.annotation.options.FieldType;
import si.medius.common.annotation.table.
    ColumnPresentation;

```

```
@Entity
@Table(name = "OrderItem")
public class OrderItem implements Serializable
{
    private Integer id;
    private Integer quantity;
    private Boolean valid = true;

    private Order order = new Order();
    private Product product = new Product();

    public OrderItem()
    {
        super();
    }

    public OrderItem(Integer id, Integer quantity,
        Boolean valid, Order order, Product product)
    {
        super();
        this.id = id;
        this.quantity = quantity;
        this.valid = valid;
        this.order = order;
        this.product = product;
    }

    @Id
    @GeneratedValue
    public Integer getId()
    {
        return id;
    }

    public void setId(Integer id)
    {
        this.id = id;
    }
}
```

```
@ColumnPresentation(order = 1, width = 60)
@FieldPresentation(order = 1, required = true)
public Integer getQuantity()
{
    return quantity;
}

public void setQuantity(Integer quantity)
{
    this.quantity = quantity;
}

@ColumnPresentation(order = 2, width = 70)
@FieldPresentation(order = 2, required = true)
public Boolean getValid()
{
    return valid;
}

public void setValid(Boolean valid)
{
    this.valid = valid;
}

// @FieldPresentation(order = 3, required = true,
//     type = FieldType.LOOKUP_SIMPLE)
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "fk_idOrder",
    referencedColumnName = "Id", nullable = false)
public Order getOrder()
{
    return order;
}

public void setOrder(Order order)
{
    this.order = order;
}
```

```
@FieldPresentation(order = 4, required = true, type
    = FieldType.LOOKUP_SIMPLE)
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "fk_idProduct",
    referencedColumnName = "Id", nullable = false)
public Product getProduct()
{
    return product;
}

public void setProduct(Product product)
{
    this.product = product;
}

@ColumnPresentation(order = 3, width = 199)
@Transient
public String getProductString()
{
    return product.getLabel();
}

@Override
public int hashCode()
{
    final int prime = 31;
    int result = 1;
    result = prime * result + ((id == null) ? 0 : id
        .hashCode());
    result = prime * result + ((order == null) ? 0 :
        order.hashCode());
    result = prime * result + ((product == null) ? 0
        : product.hashCode());
    result = prime * result + ((quantity == null) ?
        0 : quantity.hashCode());
    result = prime * result + ((valid == null) ? 0 :
        valid.hashCode());
    return result;
}
```

```

@Override
public boolean equals(Object obj)
{
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    OrderItem other = (OrderItem) obj;
    if (id == null)
    {
        if (other.id != null)
            return false;
    }
    else if (!id.equals(other.id))
        return false;
    if (product == null)
    {
        if (other.product != null)
            return false;
    }
    else if (!product.equals(other.product))
        return false;
    return true;
}
}

```

### A.2.6 Entitetno zrno Product

```

package si.medius.test.entity;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

```

```

import si.medius.common.annotation.editor.
    FieldPresentation;
import si.medius.common.annotation.lookup.
    LookupPresentation;
import si.medius.common.annotation.table.
    ColumnPresentation;
import si.medius.test.managers.local.ProductManagerLocal
    ;

@Entity
@Table(name = "Product")
@LookupPresentation(format = "#{id} - #{label}",
    managerClass = ProductManagerLocal.class)
public class Product implements Serializable
{
    private Integer id;
    private String label;
    private Integer quantityInStock;
    private Integer quantityReserved;
    private Boolean valid = true;

    public Product()
    {
        super();
    }

    public Product(Integer id, String label, Integer
        quantityInStock, Integer quantityReserved,
        Boolean valid)
    {
        super();
        this.id = id;
        this.label = label;
        this.quantityInStock = quantityInStock;
        this.quantityReserved = quantityReserved;
        this.valid = valid;
    }
}

```

```
@Id
@GeneratedValue
@ColumnPresentation(order = 1, width = 50)
public Integer getId()
{
    return id;
}

public void setId(Integer id)
{
    this.id = id;
}

@ColumnPresentation(order = 1, width = 190)
@FieldPresentation(order = 1, required = true)
public String getLabel()
{
    return label;
}

public void setLabel(String label)
{
    this.label = label;
}

@ColumnPresentation(order = 2, width = 80)
@FieldPresentation(order = 2, required = true)
public Integer getQuantityInStock()
{
    return quantityInStock;
}

public void setQuantityInStock(Integer
    quantityInStock)
{
    this.quantityInStock = quantityInStock;
}

@ColumnPresentation(order = 3, width = 90)
```

```
@FieldPresentation(order = 3, required = true)
public Integer getQuantityReserved()
{
    return quantityReserved;
}

public void setQuantityReserved(Integer
    quantityReserved)
{
    this.quantityReserved = quantityReserved;
}

@ColumnPresentation(order = 4, width = 73)
@FieldPresentation(order = 4, required = true)
public Boolean getValid()
{
    return valid;
}

public void setValid(Boolean valid)
{
    this.valid = valid;
}

@Override
public int hashCode()
{
    final int prime = 31;
    int result = 1;
    result = prime * result + ((id == null) ? 0 : id
        .hashCode());
    result = prime * result + ((label == null) ? 0 :
        label.hashCode());
    result = prime * result + ((quantityInStock ==
        null) ? 0 : quantityInStock.hashCode());
    result = prime * result + ((quantityReserved ==
        null) ? 0 : quantityReserved.hashCode());
    result = prime * result + ((valid == null) ? 0 :
        valid.hashCode());
}
```

```
        return result;
    }

    @Override
    public boolean equals(Object obj)
    {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Product other = (Product) obj;
        if (id == null)
        {
            if (other.id != null)
                return false;
        }
        else if (!id.equals(other.id))
            return false;
        if (label == null)
        {
            if (other.label != null)
                return false;
        }
        else if (!label.equals(other.label))
            return false;
        if (valid == null)
        {
            if (other.valid != null)
                return false;
        }
        else if (!valid.equals(other.valid))
            return false;
        return true;
    }
}
```

## Dodatek B

# Izvorna koda integracije na portal Liferay®

### B.1 Namestitvena datoteka persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/
  xml/ns/persistence" xmlns:xsi="http://www.w3.org
  /2001/XMLSchema-instance" xsi:schemaLocation="http://
  java.sun.com/xml/ns/persistence http://java.sun.com/
  xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="MediusTestEJB">

    <provider>org.hibernate.ejb.
      HibernatePersistence</provider>
    <jta-data-source>mediusDatasource</jta-
      data-source>
    <jar-file>/mysql-connector-java-5.0.8-
      bin.jar</jar-file>

    <class>si.medius.test.entity.Address</
      class>
    <class>si.medius.test.entity.Customer</
      class>
    <class>si.medius.test.entity.Order</
      class>
    <class>si.medius.test.entity.OrderItem</
```

```
        class>
    <class>si.medius.test.entity.Post</class>
    >
    <class>si.medius.test.entity.Product</
    class>

    <properties>
        <property name="hibernate.
            dialect" value="org.hibernate
                .dialect.MySQLDialect" />
        <property name="hibernate.
            hbm2ddl.auto" value="update" /
            >
        <property name="hibernate.
            show_sql" value="true" />
        <property name="hibernate.
            format_sql" value="true" />
        <property name="hibernate.
            default_catalog" value="
                medius" />
    </properties>

    </persistence-unit>
</persistence>
```

## Dodatek C

# Izvorna koda integracije na portal IBM Websphere Portal®

### C.1 Skripta za izdelavo podatkovne baze IBM DB2®

---

---

— *DBMS name:*            *IBM DB2 UDB 9.x Common Server*  
— *Created on:*            *17.11.2011 12:04:04*

---

---

---

---

— *Table:* "Address"

---

---

```
create table "Address"
(
    "Id_Address"            INTEGER                    not null
                      generated by default as identity ,
```

```

"Id_Post"           INTEGER           not null ,
"Id_Customer"       INTEGER,
"Street"            VARCHAR(500)       not null ,
"HouseNumber"       VARCHAR(500)       not null ,
"City_Address"      VARCHAR(500) ,
"State"             VARCHAR(500)       not null ,
"Valid_Address"     SMALLINT,
constraint "P_Identifier_1" primary key ("Id_Address"
)
);

```

---

```

— Index: "R_AddressPost_FK"

```

---

```

create index "R_AddressPost_FK" on "Address" (
    "Id_Post"           ASC
);

```

---

```

— Index: "R_CustAddr_FK"

```

---

```

create index "R_CustAddr_FK" on "Address" (
    "Id_Customer"       ASC
);

```

---

```

— Table: "Customer"

```

---

```

create table "Customer"
(
    "Id_Customer"          INTEGER                not null
                        generated by default as identity ,
    "FirstName"            VARCHAR(500)           not null ,
    "SecondName"           VARCHAR(500)           not null ,
    "DateOfBirth"          TIMESTAMP              not null ,
    "Valid_Customer"       SMALLINT,
    constraint "P_Identifier_1" primary key ("Id_Customer"
                        ")
);

```

---

— *Table: "Order"*

---

```

create table "Order"
(
    "Id_Order"             INTEGER                not null
                        generated by default as identity ,
    "Id_Customer"          INTEGER                not null ,
    "DateCreated"          TIMESTAMP              not null ,
    "DateConcluded"        TIMESTAMP,
    "DatePayed"            TIMESTAMP,
    "Valid_Order"          SMALLINT               not null ,
    constraint "P_Identifier_1" primary key ("Id_Order")
);

```

---

— *Index: "R\_CustomerOrder\_FK"*

---

```

create index "R_CustomerOrder_FK" on "Order" (
    "Id_Customer"          ASC
);

```

---



---

```

— Table: "OrderItem"

```

---



---

```

create table "OrderItem"
(
    "Id_OrderItem"          INTEGER          not null
                        generated by default as identity ,
    "Id_Order"              INTEGER          not null ,
    "Id_Product"            INTEGER          not null ,
    "Quantity"              INTEGER          not null ,
    "Valid_OrderItem"       SMALLINT         not null ,
    constraint "P_Identifier_1" primary key ("
        Id_OrderItem")
);

```

---



---

```

— Index: "R_OrdOItem_FK"

```

---



---

```

create index "R_OrdOItem_FK" on "OrderItem" (
    "Id_Order"              ASC
);

```

---



---

```

— Index: "R_OItemProd_FK"

```

---



---

---

---

```
create index "R_OItemProd_FK" on "OrderItem" (  
    "Id_Product"          ASC  
);
```

---

---

```
— Table: "Post"  
—
```

---

---

```
create table "Post"  
(  
    "Id_Post"          INTEGER          not null  
        generated by default as identity ,  
    "Number"          INTEGER          not null ,  
    "CityPost"        VARCHAR(500)     not null ,  
    "Valid_Post"      SMALLINT         not null ,  
    constraint "P_Identifier_1" primary key ("Id_Post")  
);
```

---

---

---

---

```
— Table: "Product"  
—
```

---

---

```
create table "Product"  
(  
    "Id_Product"      INTEGER          not null  
        generated by default as identity ,  
    "Label"           VARCHAR(500)     not null ,  
    "QuantityInStock" INTEGER          not null ,  
    "QuantityReserved" INTEGER,   
    "Valid_Product"   SMALLINT         not null ,  
    constraint "P_Identifier_1" primary key ("Id_Product")  
);
```

```

    )
);

alter table "Address"
  add constraint "F_Relationship_Add" foreign key ("
    Id_Post")
    references "Post" ("Id_Post")
    on delete restrict on update restrict;

alter table "Address"
  add constraint "F_Relationship_Cus" foreign key ("
    Id_Customer")
    references "Customer" ("Id_Customer")
    on delete restrict on update restrict;

alter table "Order"
  add constraint "F_Relationship_Cus" foreign key ("
    Id_Customer")
    references "Customer" ("Id_Customer")
    on delete restrict on update restrict;

alter table "OrderItem"
  add constraint "F_R_OrderItemProdu" foreign key ("
    Id_Product")
    references "Product" ("Id_Product")
    on delete restrict on update restrict;

alter table "OrderItem"
  add constraint "F_R_OrderOrderItem" foreign key ("
    Id_Order")
    references "Order" ("Id_Order")
    on delete restrict on update restrict;

```

# Slike

|      |                                                                                                                                                                                                                                            |    |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.1  | Grafični prikaz trinivojske arhitekture z nekaterimi tehnologijami Java EE, ki se uporabljajo na posameznem nivoju . . . . .                                                                                                               | 15 |
| 2.2  | Primer strani portala Liferay, ki na levi strani prikazuje portleta za pregled sestankov in elektronske pošte . . . . .                                                                                                                    | 17 |
| 2.3  | Enostaven grafičen prikaz agregacije fragmentov, ki jih generirajo portleti A, B in C, v stran na portalu . . . . .                                                                                                                        | 19 |
| 2.4  | Grafični prikaz življenjskega cikla portleta [3] . . . . .                                                                                                                                                                                 | 20 |
| 2.5  | Diagram zaporedja za primer osvežitve strani (zgornja polovica) in določeno akcijo (spodnja polovica) [3][6] . . . . .                                                                                                                     | 22 |
| 2.6  | Primer strukture razširjene WAR datoteke . . . . .                                                                                                                                                                                         | 25 |
| 2.7  | Primer datoteke „portlet.xml“ . . . . .                                                                                                                                                                                                    | 27 |
| 2.8  | Diagram zaporedja pri pošiljanju dogodka med portleti . . . . .                                                                                                                                                                            | 28 |
| 2.9  | Definicija dogodka v datoteki portlet.xml [26] . . . . .                                                                                                                                                                                   | 28 |
| 2.10 | Agregacija fragmentov iz portletov implementiranih po standardu WSRP [27] . . . . .                                                                                                                                                        | 31 |
| 2.11 | Vstopna stran IBM WebSphere Portal-a (različice Express) . . . . .                                                                                                                                                                         | 33 |
| 3.1  | Primer osnovne maske za prikaz entitetnega zrna „Post“. Zgoraj levo so gumbi za dodajanje, urejanje in brisanje podatkov. Zgoraj desno pa so gumbi za filtriranje, sortiranje, osveževanje, ter izvoz podatkov v različne formate. . . . . | 37 |
| 3.2  | Primer vnosne maske, ki vsebuje vnosna polja za vse podatke, ki so potrebni za generiranje instance entitetnega zrna „Post“. Rdeče zvezdice označujejo, da je podatek obvezen. . . . .                                                     | 38 |
| 3.3  | Koncept ogrodja Medius Vaddin Common po posameznih arhitekturnih nivojih . . . . .                                                                                                                                                         | 39 |
| 3.4  | Primer generirane maske za prikaz entitetnega zrna „Post“ na osnovi izvirne kode 3.4 . . . . .                                                                                                                                             | 47 |

|      |                                                                                                                                                                                         |    |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.5  | Primer generirane vnosne maske za kreiranje nove instance entiteta zrna „Post“ . . . . .                                                                                                | 47 |
| 3.6  | Primer vnosne maske zgrajeni na izvorni kodi z anotacijo <code>@FieldPresentation</code> , ki vsebuje parameter „type“ z vrednostjo „FieldType.LOOKUP_SIMPLE“ . . . . .                 | 49 |
| 3.7  | Primer vnosne maske zgrajeni na izvorni kodi z anotacijo <code>@FieldPresentation</code> , ki vsebuje parameter „type“ z vrednostjo „FieldType.LOOKUP_MANY“ . . . . .                   | 51 |
| 3.8  | Primer vnosne maske za naročila zgrajeni na izvorni kodi z anotacijo <code>@FieldPresentation</code> , ki vsebuje parameter „type“ z vrednostjo „FieldType.LOOKUP_MANYTOMANY“ . . . . . | 55 |
| 3.9  | Primer maske za izbiro izdelkov (v vlogi definiranja desnega dela relacije mnogo proti mnogo) . . . . .                                                                                 | 56 |
| 3.10 | Arhitektura gradnje uporabniškega vmesnika . . . . .                                                                                                                                    | 57 |
| 3.11 | Izgled tipične aplikacije razvite z ogrodjem Medius Vaadin Common . . . . .                                                                                                             | 58 |
| 4.1  | Konceptualni nivo podatkovne baze . . . . .                                                                                                                                             | 61 |
| 4.2  | Primer pogleda za entitetno zrno Post. . . . .                                                                                                                                          | 66 |
| 4.3  | Primer pogleda za entitetno zrno Product. . . . .                                                                                                                                       | 67 |
| 4.4  | Primer pogleda za entitetno zrno Customer. . . . .                                                                                                                                      | 68 |
| 4.5  | Primer pogleda za entitetno zrno Order. . . . .                                                                                                                                         | 69 |
| 4.6  | Manjkajoče datoteke znotraj „default“ mape na distribuciji JBoss, ki je osnova namestitve portala Liferay . . . . .                                                                     | 74 |
| 4.7  | Prikaz nastavitvev za JPA projekt v razvojnem okolju Eclipse . . . . .                                                                                                                  | 75 |
| 4.8  | Struktura projekta MediusTestEJB. . . . .                                                                                                                                               | 76 |
| 4.9  | Namestitvena deskriptorja projekta MediusTestEJB (zgoraj) in MediusTestEAR (spodaj). . . . .                                                                                            | 78 |
| 4.10 | Nastavitve za spletni del aplikacije. . . . .                                                                                                                                           | 81 |
| 4.11 | Struktura spletnega projekta „MediusTestWEB“. . . . .                                                                                                                                   | 86 |
| 4.12 | Dodajanje nove strani na portalu Liferay. . . . .                                                                                                                                       | 89 |
| 4.13 | Dodajanje portleta na stran portala Liferay. . . . .                                                                                                                                    | 90 |
| 4.14 | Portlet, ki prikazuje entitetno zrno „Post“ na portalu Liferay. . . . .                                                                                                                 | 91 |
| 4.15 | Prikaz portleta po vnosu popravkov. . . . .                                                                                                                                             | 92 |
| 4.16 | Primer dodajanja entitete pri entitetnem zrnju „Post“. . . . .                                                                                                                          | 95 |
| 4.17 | Primer dodajanja entitete pri entitetnem zrnju „Customer“. . . . .                                                                                                                      | 96 |
| 4.18 | Izbira določene pošte pri vnašanju naslovov določene stranke. . . . .                                                                                                                   | 97 |
| 4.19 | Primer portleta, ki prikazuje podatke o naročilih. . . . .                                                                                                                              | 98 |

|      |                                                                                                    |     |
|------|----------------------------------------------------------------------------------------------------|-----|
| 4.20 | Izbira funkcionalnosti ob nameščanju operacijskega sistema CentOS (različica 5.7).                 | 100 |
| 4.21 | Test podatkovnega vira mediusDatasource na aplikacijskem strežniku Websphere.                      | 105 |
| 4.22 | Vmesnik za upravljanje aplikacij v administracijski konzoli aplikacijskega strežnika WebSphere AS. | 109 |
| 4.23 | Nastavitev strežnika za namestitev aplikacije.                                                     | 110 |
| 4.24 | Prikaz spletnih aplikacij v administracijskem delu portala WebSphere.                              | 115 |
| 4.25 | Napaka ob prvem poskusu dodajanja portleta na stran portala.                                       | 117 |
| 4.26 | Format JNDI reference na WebSphere portalu                                                         | 118 |
| 4.27 | Spremenjene anotacije za generiranje primarnih ključev                                             | 120 |
| 4.28 | Portlet za prikaz entitete „Post“ (urejanje poštних števil) na WebSphere Portal-u.                 | 120 |
| 4.29 | Napaka ob izvozu podatkov iz tabele v format „XLS“.                                                | 122 |
| 4.30 | Nastavitev zaporedja nalaganja programskih knjižnic v orodju Rational Application Developer.       | 124 |
| 4.31 | Uspešen izvoz podatkov v „XLS“ datoteko na WebSphere portalu.                                      | 125 |
| 4.32 | Dodajanje entitete v primeru sejnega zrna „Post“ na WebSphere portalu.                             | 127 |
| 4.33 | Primer urejanja entitete tipa „Address“ na WebSphere portalu.                                      | 128 |
| 4.34 | Izbira pošte iz spustnega menija na WebSphere portalu.                                             | 128 |
| 4.35 | Primer razmerja mnogo proti mnogo pri entitetah tipa „Order“ in „Product“ na WebSphere portalu.    | 129 |

# List of Algorithms

|      |                                                                                                                                             |    |
|------|---------------------------------------------------------------------------------------------------------------------------------------------|----|
| 3.1  | Izsek skripte za generiranje tabele Post v podatkovni bazi Oracle                                                                           | 41 |
| 3.2  | Definicija entitetnega zrna na osnovi tabele Post iz relacijske podatkovne baze.                                                            | 42 |
| 3.3  | Definicija sejnega zrna za entiteto Post.                                                                                                   | 43 |
| 3.4  | Primer anotacij „getter“ metod za entitetno zrno „Post“.                                                                                    | 46 |
| 3.5  | Primer anotacij „getter“ metod za entitetno zrno „Address“.                                                                                 | 48 |
| 3.6  | Anotacije entitetnih zrn „Customer“ in „Address“, ki so potrebne za generiranje polja tipa „FieldType.LOOKUP_MANY“                          | 50 |
| 3.7  | Anotacije entitetnega zrna „Order“, ki so potrebne za generiranje polja tipa „FieldType.LOOKUP_MANYTOMANY“                                  | 53 |
| 3.8  | Anotacije entitetnega zrna „OrderItem“, ki so potrebne za generiranje polja tipa „FieldType.LOOKUP_MANYTOMANY“                              | 54 |
| 4.1  | Nastavitvena datoteka za persistence.xml za Hibernate                                                                                       | 63 |
| 4.2  | Nastavitvena datoteka za podatkovni vir do MySQL podatkovne baze                                                                            | 64 |
| 4.3  | Primer definicije pogleda na spletnem delu aplikacije.                                                                                      | 65 |
| 4.4  | Napaka ob prvem poskusu definiranja podatkovnega vira na JBoss strežniku                                                                    | 72 |
| 4.5  | Sporočilo na standardnem izhodu strežnika JBoss ki pove, da je bil ustvarjen podatkovni vir do podatkovne baze z imenom „mediusDatasource“. | 73 |
| 4.6  | Prikaz sporočil strežnika o uspešnem nameščanju sejnih zrn.                                                                                 | 79 |
| 4.7  | Primer spletnega pogleda za entitetno zrno „Post“.                                                                                          | 82 |
| 4.8  | Primer osnovne aplikacije za entitetno zrno „Post“.                                                                                         | 83 |
| 4.9  | Primer datoteke portlet.xml potrebne za namestitev portleta, ki prikazuje entitetno zrno „Post“, na portal Liferay.                         | 85 |
| 4.10 | Primer datoteke „web.xml“ spletnega dela aplikacije.                                                                                        | 87 |
| 4.11 | Primer datoteke „application.xml“ spletnega dela aplikacije.                                                                                | 88 |
| 4.12 | Obvestila aplikacijskega strežnika ob nameščanju portletov.                                                                                 | 89 |
| 4.13 | Primer deskriptorja „liferay-portlet.xml“.                                                                                                  | 93 |

|      |                                                                                                                            |     |
|------|----------------------------------------------------------------------------------------------------------------------------|-----|
| 4.14 | Primer deskriptorja „liferay-display“ . . . . .                                                                            | 93  |
| 4.15 | Deskriptor persistence.xml za WebSphere Portal. . . . .                                                                    | 107 |
| 4.16 | Primer deskriptorja ejb-jar.xml za WebSphere Portal. . . . .                                                               | 108 |
| 4.17 | Primer deskriptorja „portlet.xml“ za portal WebSphere. . . . .                                                             | 112 |
| 4.18 | Primer deskriptorja „web.xml“ za portal WebSphere. . . . .                                                                 | 113 |
| 4.19 | XML datoteka za vnos reference nameščenega EAR projekta na portal. . . . .                                                 | 116 |
| 4.20 | Izvorna koda metode za generiranje ustrezne reference na sejno zrno nameščeno na aplikcijskem strežniku WebSphere. . . . . | 119 |

# Literatura

- [1] Joseph M. Firestone Ph.D., *Enterprise Information Portals and Knowledge Management*, Oxford: Butterworth-Heinemann, 2002, pogl. 1,2
- [2] Kevin Roebuck, *Enterprise Portal: High-impact Strategies - What You Need Know: Definitions, Adaptations, Impact, Benefits, Maturity, Vendors*, Tebbo, 2011
- [3] Jonas X. Yuan, *Liferay Portal 5.2 Systems Developement*, Birmingham: Packt Publishing, 2009, poglavje 2
- [4] M. Gronross, *Book of Vaadin: Vaadin 6.4*, Turku, Finland: Vadin Ltd, 2010
- [5] Viktor Brajak, *Avtomatsko generiranje grafičnega uporabniškega vmesnika z označevanjem strežniške javanske kode*, Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Ljubljana 2011

## **Spletni viri:**

- [6] (2011) Comparing the JSR 168 Java Portlet Specification with the IBM Portlet API. Dostopno na:  
[http://www.ibm.com/developerworks/websphere/library/techarticles/0312\\_hepper/hepper.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0312_hepper/hepper.html)
- [7] (2011) Your First Cup: An Introduction to Java Platform, Enterprise Edition. Dostopno na:  
<http://download.oracle.com/javaee/6/firstcup/doc/gkhoy.html>
- [8] (2011) Liferay: Technical Specifications. Dostopno na:  
<http://www.liferay.com/products/liferay-portal/tech-specs>
- [9] (2011) SAP Netweaver Standards Support. Dostopno na:  
<http://www.sap.com/platform/netweaver/standardssupport/index.epx>

- [10] (2010) JBoss Enterprise Portal Platform. Dostopno na:  
<http://www.jboss.com/pdf/epp-5-datasheet.pdf>
- [11] (2011) WebSphere Portal Features and benefits. Dostopno na:  
<http://www-01.ibm.com/software/genservers/portal/features/>
- [12] (2010) Microsoft SharePoint 2010 Evaluation Guide. Dostopno na:  
<http://go.microsoft.com/?linkid=9727428>
- [13] (2011) Oracle Webcenter Portal. Dostopno na:  
<http://www.oracle.com/technetwork/middleware/webcenter/portal/index.html>
- [14] (2004) Understanding the Java Portlet Specification. Dostopno na:  
<http://www.developer.com/java/web/article.php/3366111/Understanding-the-Java-Portlet-Specification.htm>
- [15] The Java Portlet API (JSR 168). Dostopno na:  
[http://media.wiley.com/product\\_data/excerpt/13/04714695/0471469513.pdf](http://media.wiley.com/product_data/excerpt/13/04714695/0471469513.pdf)
- [16] (2003) Java™ Portlet Specification. Version 1.0. Dostopno na:  
[http://download.oracle.com/otn-pub/jcp/PORTLET\\_1.0-FR-SPEC-G-F/portlet-1\\_0-fr-spec.pdf](http://download.oracle.com/otn-pub/jcp/PORTLET_1.0-FR-SPEC-G-F/portlet-1_0-fr-spec.pdf)
- [17] (2008) JSR 286: Portlet Specification 2.0. Dostopno na:  
<http://jcp.org/en/jsr/detail?id=286>
- [18] (2008) What's new in the Java Portlet Specification V2.0 (JSR 286)?  
Dostopno na:  
[http://www.ibm.com/developerworks/websphere/library/techarticles/0803\\_hepper/0803\\_hepper.html](http://www.ibm.com/developerworks/websphere/library/techarticles/0803_hepper/0803_hepper.html)
- [19] (2011) Stories. Dostopno na:  
<http://www.liferay.com/products/liferay-portal/stories>
- [20] (2011) Gartner: Magic Quadrants for Horizontal Portals. Dostopno na:  
<http://www.gartner.com/technology/reprints.do?id=1-17RTIFE&ct=111025&st=sg>
- [21] (2011) Case studies for WebSphere Portal for Multiplatforms. Dostopno na:  
[http://www-01.ibm.com/software/success/cssdb.nsf/lateststoriesVW?OpenView&Count=10&RestrictToCategory=wsportal&cty=en\\_us](http://www-01.ibm.com/software/success/cssdb.nsf/lateststoriesVW?OpenView&Count=10&RestrictToCategory=wsportal&cty=en_us)

- [22] (2011) Portal Solution Blog. Oracle WebCenter Explained! Dostopno na: <http://blogs.perficient.com/portals/2011/10/03/oracle-webcenter-explained/>
- [23] (2011) Oracle Customer Successes. Dostopno na: <http://www.oracle.com/us/corporate/customers/customersearch/>
- [24] [index.html?ssNextRow=31&xProductServices=14/57/2949](http://www.oracle.com/us/corporate/customers/customersearch/index.html?ssNextRow=31&xProductServices=14/57/2949)
- [25] (2011) Practical Guide for Oracle WebCenter PS3 Security Implementation. Dostopno na: <http://andrejusb.blogspot.com/2011/04/practical-guide-for-oracle-webcenter.html>
- [26] (2008) JavaTM Portlet Specification. Version 2.0. Dostopno na: <http://download.oracle.com/otn-pub/jcp/portlet-2.0-fr-oth-JSpec/portlet-2.0-fr.zip>
- [27] (2005) Introduction to Web Services for Remote Portlets. Dostopno na: <http://www.ibm.com/developerworks/library/ws-wsrp/>
- [28] (2011) Liferay Portal. Dostopno na: <http://sourceforge.net/projects/lportal/files/Liferay%20Portal/6.0.5/liferay-portal-jboss-6.0.5.zip/download>
- [29] (2011) Vaadin Plug-in for Eclipse. Dostopno na: <http://vaadin.com/eclipse>
- [30] (2011) Trial download: WebSphere Portal 7.0. Dostopno na: [http://www14.software.ibm.com/webapp/download/preconfig.jsp?id=2010-12-07+03%3A32%3A48.419020R&S\\_TACT=&S\\_CMP=](http://www14.software.ibm.com/webapp/download/preconfig.jsp?id=2010-12-07+03%3A32%3A48.419020R&S_TACT=&S_CMP=)