Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Boštjan Štor

Uporaba načrtovalskih vzorcev in tehnologije Java EE na primeru razvoja aplikacije skladišča

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE

RAČUNALNIŠTVO IN INFORMATIKA

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana 2013

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00062/2012
Datum: 04.12.2012

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

Kandidat: **BOŠTJAN ŠTOR**

Naslov: **UPORABA NAČRTOVALSKIH VZORCEV IN TEHNOLOGIJE JAVA EE NA PRIMERU RAZVOJA APLIKACIJE SKLADIŠČA**

DESIGN PATTERNS AND JAVA EE TECHNOLOGIES USAGE FOR WAREHOUSE APPLICATION DEVELOPMENT

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Analizirajte in predstavite ključne načrtovalske vzorce za razvoj programske opreme iz kataloga GoF ter identificirajte primere uporabe načrtovalskih vzorcev. Na primeru spletne aplikacije Skladišče izdelajte načrt (design) aplikacije in pri tem uporabite načrtovalske vzorce. Slednje prikažite na razrednem diagramu. Na osnovi podrobnega načrta identificirajte kandidatne tehnologije platforme Java EE in implementirajte vzorčno spletno aplikacijo.

Mentor:

prof. dr. Matjaž B. Jurič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Boštjan Štor, z vpisno številko 63070167, sem avtor diplomskega dela z naslovom:

Uporaba načrtovalskih vzorcev in tehnologije Java EE na primeru razvoja aplikacije skladišča

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Matjaža Branka Juriča;
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano verzijo diplomskega dela.
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki ″Dela FRI″.

V Ljubljani, dne 28. februarja 2013 Podpis avtorja:

Zahvala

Zahvaljujem se svojemu mentorju prof. dr. Matjažu Branku Juriču za nasvete, vložen trud in pomoč pri izdelavi diplomskega dela. Javnemu zavodu ARNES se zahvaljujem za možnost razvoja spletne aplikacije v obliki, primerni za prikaz teoretičnih tem na praktičnem primeru.

Poleg tega se zahvaljujem svoji družini za podporo in spodbude pri izdelavi diplomskega dela.

Kazalo

Seznam uporabljenih kratic

REST – prenos reprezentativnega stanja (REpresentational State Transfer)

JPA – Java Persistence API

EJB – Entepreis Java Bean

JAX-RS – Java API za RESTful spletne storitve

SQL – Structured Query Language

JSF – Java Server Faces

CDI – Context Dependency Injection

JEE – Java Enterprise Edition

DDV – Davek na Dodano Vrednost

RDBMS – Relation DataBase Management System

GNU – GNU operacijski sistem katerega kratica pomeni GNU's not Unix

GPL – General Public Licence (splošna javna licenca)

ORM – Object Relational Mapping

CSS – Cascading Style Sheet

HTML – Hyper Text Markup Language

XHTML – Extensible HTML

API – Application Programming Interface

AJAX – Asynchronous JavaScript and XML

XML – Extensible Markup Language

DAO – Database Access Object

JTA – Java Transacttion API (Java Transakcijski API)

IaaS – Infrastructure as a Service (Storitve v oblaku)

SVN – Subversion (Sistem za nadzor na verzijami programske opreme)

LDAP – Lightweight directory access protocol (Protokol za preprost dostop do direktorija)

Povzetek

V diplomskem delu so predstavljeni načrtovalski vzorci za razvoj programske opreme ter njihova uporaba pri razvoju spletne aplikacije Skladišče. Načrtovalski vzorci predstavljajo splošne rešitve problemov, ki se jim je enostavno izogniti na začetku razvoja. Spletna aplikacija nadzoruje lokacijo omrežne in multimedijske opreme ter omogoča dodajanje, urejanje, pošiljanje, prejemanje, rezerviranje in načrtovanje nabave naprav. Omogočeno je tako tiskanje dokumentov, ki se generirajo pri pošiljanju in prejemanju naprav, kot tudi izvoz rezultatov iskanja in načrtovanja nabave. Aplikacija, ki je bila razvita v sodelovanju z javnim zavodom ARNES, lajša nadzor nad lokacijami naprav in avtomatizira nekatere korake pri pošiljanju naprav. V delu so predstavljene tudi tehnologije, ki so bile uporabljene za realizacijo aplikacije, in sicer Java Enterprise Edition, ki vsebuje EJB, JSF in JAX-RS.

Ključne besede:

Načrtovalski vzorci, JEE, EJB, JSF

Abstract

This thesis presents design patterns for software development and their use in the development of a web application Skladišče. Design patterns are general solutions for problems which are easily avoided at the begining of the development. Web application controls the location of the network and multimedia equipment, and allows adding, editing, recieving, reserving and planning acquisition of devices. It is also possible to print documents that are generated by sending and recieving devices, as well as export search results and planning of purchase. The web application, that was developet in cooperation with ARNES facilitates control of device location and automates some steps in sending devices. The thesis also presents technologies that were used for the realization of the application, including Java Enterprise Edition, which includes EJB, JSF and JAX-RS.

Key words:

Design patterns, JEE, EJB, JSF

1 Uvod

V diplomskem delu bomo opisali načrtovalske vzorce za razvoj programske opreme. Uporabo nekaterih načrtovalskih vzorcev bomo prikazali na primeru spletne aplikacije, ki je bila razvita za namene diplomske naloge in se bo tudi kasneje uporabljala v komercialne namene, za nadzor omrežne in multimedijske opreme. Načrtovalski vzorci na prvi pogled izgledajo kot pravila lepega programiranja, vendar poleg lepše berljivosti in preglednosti kode pripomorejo k lažjemu razvoju, saj se v primeru, da razvijamo po njihovem vzoru izognemo marsikateremu problemu, ki bi nam lahko povzročal velike preglavice v kasnejših stopnjah razvoja. Zaradi uporabe načrtovalskih vzorcev, bo imela naša aplikacija drugačno strukturo in drugačno obliko razredov, kot bi jo imela, če načrtovalskih vzorcev ne bi uporabili. Uporabo načrtovalskih vzorcev bomo prikazali na primeru spletne aplikacije Skladišče. Skladišče je namenjeno sledenju omrežne in multimedijske opreme, ki jo pripravimo za delovanje na določeni organizaciji. Spletna aplikacija mora omogočati dodajanje tipov naprav, za katere skrbimo, razpisov preko katerih nabavimo naprave ter seveda urejanje in brisanje le teh. Nato naprave dodajamo v sistem, jih pošiljamo med organizacijami in prejemamo nazaj. Za vsako transakcijo mora biti papirna sled, tako da mora aplikacija izdajati in prejemati dokumente, iz katerih je razvidno, kam smo poslali naprave in pod katerimi pogoji. Naprave je možno tudi le posoditi. Naknadno se je pojavila potreba po še dveh dodatnih funkcionalnostih. Dodali smo še možnost rezervacije tipa naprav in planiranje nabave naprav glede na predvidena sredstva. Odločili smo se, da je to aplikacijo najbolje razviti v Javi. Z vidika tehnologij, ki nam jih ponuja Java, smo uporabili Enterprise Java Beans iz modula Java Enterprise Edition. Za povezavo s podatkovno bazo smo izbrali Java Persistence API, katerega smo dobili v paketih ponudnika Hibernate. Zaradi uporabe Jave EE se bo spletna aplikacija izvajala na aplikacijskem strežniku. Obstaja velika izbira aplikacijskih strežnikov in mi smo se odločili za GlassFish 3.1.2.. GlassFish ima odprtokodno verzijo in se je za naše potrebe izkazala kot dobra izbira. Za potrebe dela z datotekami, za prenos datotek na strežnik in iz njega, smo morali uporabiti nekaj JavaScript prijemov, kjer so se za komunikacijo s strežnikom, najbolje izkazale spletne storitve REST. Ker uporabljamo Javo, smo uporabili tehnologijo JAX-RS. Da je uporabniški vmesnik lepši in da deluje bolj tekoče, smo se odločili uporabiti enega izmed ogrodij za izdelavo spletne strani z uporabo Java Server Faces. Za JSF smo se odločili, ker je del Jave EE. Uporabili smo ogrodje ICEfaces. Kasneje se je izkazalo, da je bila to slaba izbira in bi bilo bolje uporabiti PrimeFaces. Tako kot pri vsakem programu pa ne smemo pozabiti na testiranje. Za začetno testiranje, da vidimo, če metode delujejo pravilno, smo uporabili testno ogrodje jUnit. Vsa orodja opisana v tem kratkem povzetku, so opisana v nadaljevanju diplomskega dela.

2 Načrtovalski vzorci

V razvoju programske opreme so načrtovalski vzorci [4] nekakšne splošne oblike programiranja, oziroma rešitve problemov. Načrtovalski vzorci niso vezani na posamezen programski jezik in prav tako niso definirani s programsko ali psevdo kodo. Načrtovalski vzorci so sklop navodil oziroma pravil, kako pisati programsko kodo, da je lepša, lažje berljiva in nam zapovedujejo, na kakšen način morajo biti razredi in objekti povezani. Prav tako nam povedo, kaj bi moral posamezen modul aplikacije vsebovati. Ker je naša aplikacija objektno orientirana, bo večina uporabljenih načrtovalskih vzorcev govorila o sestavi aplikacije ter povezavah med objekti in razredi v aplikaciji. Načrtovalski vzorci se ukvarjajo samo s sestavo aplikacije in ne z arhitekturo na kateri aplikacija deluje. Vzorci, po katerih deluje cel sistem, na katerem se izvaja naša aplikacija, so arhitekturni vzorci.

Načrtovalski vzorci lahko pospešijo razvoj aplikacij s tem, da nam dajo nekaj vzorcev, kako zasnovati aplikacijo in nas s tem rešijo nekaterih problemov, ki bi jih drugače videli šele pri implementaciji funkcionalnosti. Če razvijamo več aplikacij, je pametno uporabiti načrtovalske vzorce pri vseh, saj je na takšen način vzdrževanje lažje. Če so vse aplikacije so narejene po istem vzorcu, takoj prepoznamo večje dele aplikacije in opazimo, kje se kaj dogaja. V primeru da delamo v skupini, imamo celoten sistem narejen v istem stilu, zato v primeru odsotnosti enega izmed članov skupine drugim ni težko prevzeti dela na aplikaciji, oziroma ne izgubljajo časa z ugotavljanjem načina dela odsotnega člana skupine.

Glede na uporabo načrtovalske vzorce razdelimo v več skupin [5]:

- ustvarjalni (ang. Creational) (tabela 1),
- strukturni (ang. Structural) (tabela 2),
- značajski (ang. Behavioral) (tabela 3),
- vzporednostni (ang. Concurrency) (tabela 4).

2.1 Opis načrtovalskih vzorcev

2.1.1 Ustvarjalni

Ime	Opis
Abstraktna tovarna (ang. Abstract factory)	Priskrbi vmesnik za ustvarjanje skupine povezanih ali odvisnih objektov, ne da bi točno določili njihove razrede.
Graditelj (ang. Builder)	Loči kreiranje kompleksnega objekta od njegove predstavitve, tako da lahko iz istega procesa kreiranja nastane več predstavitev.
Metoda tovarne	Definira vmesnik za kreiranje objekta, ampak dovoli podrazredom

(ang. Factory method)	da prevzamejo inicializacijo objekta. Metoda tovarne dovoli, da razred prepusti inicializacijo objekta podrazredom.
Lena inicializacija (ang. Lazy initialization)	Taktika odlaganja kreiranja objekta, računanja vrednosti ali katerega drugega požrešnega procesa, dokler ga res ne potrebujemo.
Multiton	Zagotavlja da ima razred le imenovane inicializacije, in priskrbi globalno točko dostopa do njih.
Bazen objektov (ang. Object pool)	Izogne se dragim zasedanjem in sproščanjem virov tako, da reciklira objekte, ki niso več v uporabi.
Prototip (ang. Prototype)	Specificira takšne objekte, ki se kreirajo pri uporabi prototipne instance in ustvari nov objekt tako, da se kopira prototip.
Pridobitev vira v inicializaciji (ang. Resource acquisition in initialization)	Poskrbi, da so viri pravilno sproščeni tako, da jih vežemo na življenjski cikel primernih objektov.
Edinec (ang. Singelton)	Poskrbi, da ima razred samo eno instanco in priskrbi globalno točko dostopa do nje.

Tabela 1: Ustvarjalni vzorci

2.1.2 Strukturni

Ime	**Opis**
Adapter	Pretvori vmesnik razreda v nov vmesnik, ki ga odjemalec pričakuje. Adapter dovoli razredom, ki drugače nebi mogli zaradi nekompatibilnih vmesnikov, delati skupaj.
Most (ang. Bridge)	Loči abstrakcijo od implementacije, tako da se lahko neodvisno razlikujeta. Most uporablja enkapsulacijo in združevanje, lahko pa uporabi tudi dedovanje, da loči odgovornosti v različne razrede.
Kompozit (ang. Composite)	Postavi objekte v drevesno strukturo tako, da predstavljajo hierarhijo. Omogoča odjemalcu, da obravnava posamezne objekte kompozicije enakovredno.
Dekorator (ang. Decorator)	Pripne dodatne odgovornosti objektu, ne da bi spremenil njegov vmesnik. Dekoratorji zagotavljajo fleksibilno alternativo podrazredom za dodajanje dodatnih funkcionalnosti.

Fasada (ang. Facade)	Zagotavlja enoten vmesnik v naboru vmesnikov v podsistemu. Fasada definira vmesnik na višjem nivoju, zaradi katerega lažje uporabljamo podsistem.
Zrno (ang. Flyweight)	Uporabi deljenje, zaradi katerega učinkovito podpira večje število podobnih objektov.
Prednji krmilnik (ang. Front controller)	Ta načrtovalski vzorec se uporablja pri načrtovanju spletnih aplikacij. Zagotavlja centralizirano vstopno točko za serviranje zahtev.
Modul (ang. Module)	Združi več povezanih elementov, razredov, edincev, metod in globalnih elementov v enotno konceptualno entiteto.
Namestnik (ang. Proxy)	Zagotavlja nadomestek drugega objekta in kontrolira dostop do njega.

Tabela 2: Strukturni vzorci

2.1.3 Značajski

Ime	Opis
Črna tabla (ang. Blackboard)	Splošni opazovalec, ki omogoča več bralnih in pisalnih dostopov. Prenaša informacije po celem sistemu.
Veriga odgovornosti (ang. Chain of responsibility)	Izogiba se povezovanju pošiljatelja zahteve z njenim prejemnikom tako, da daje možnost več objektom, da razreši zahtevo. Poveže prejemne objekte in jim posreduje zahtevo, dokler je en ne razreši.
Ukaz (ang. Command)	Zahtevo zapakira v objekt, tako lahko parametriziramo odjemalce z različnimi zahtevki. Zahtevke postavimo v vrsto, ali jih logiramo in s tem podpremo tudi zahtevke, ki jih ne moremo razrešiti.
Tolmač (ang. Interpreter)	Pri danem jeziku definiramo predstavitev slovnice skupaj s tolmačem, ki uporabi to predstavitev, da prevede stavke jezika.
Pregledovalnik (ang. Iterator)	Zagotavlja pregled elementov v seznamu, ne da bi odkril, kako so elementi med seboj povezani.
Posrednik (ang. Mediator)	Definira objekt, ki določa, v kakšnem odnosu so objekti. Posrednik uporablja bolj svobodno povezovanje objektov, tako da ne kličejo drug drugega direktno.
Memento	Ne da bi kršili enkapsulacijo, shranimo interno stanje objekta izven tega ovoja, tako da lahko objekt kasneje obnovimo.

Prazen objekt (ang. Null object)	Izogibamo se null vrednosti objekta in raje uporabimo nek privzet objekt.
Opazovalec ali objavi/naroči (ang. Observer or publish/subscribe)	Definira ena proti več odvisnost med objekti, kjer se mora sprememba enega objekta poznati v vseh ostalih.
Hlapec (ang. Servant)	Definira skupne funkcionalnosti za skupino razredov
Specifikacija (ang. Specification)	Sestavljiva jasna poslovna logika.
Stanje (ang. State)	Dovoli objektu da spremeni svoj značaj, ko se spremeni njegovo interno stanje. Objekt bo videti, kot da je spremenil svoj razred.
Strategija (ang. Strategy)	Definira skupino algoritmov, ki se lahko zamenjujejo glede na to kakšen odjemalec uporablja storitev.
Predloga (ang. Template method)	Definira ogrodje algoritma ali operacije in prelaga določene korake na podrazrede. Ta metoda dovoljuje da podrazredi spremenijo določene korake algoritma, ne da bi spremenili njegovo strukturo.
Obiskovalec (ang. Visitor)	Predstavlja operacijo, ki mora bit izvedena na elementih strukture objekta. Obiskovalec dovoljuje, da definiramo novo operacijo, ne da bi spremenili razrede elementov nad katerimi deluje.

Tabela 3: Značajski vzorci

2.1.4 Vzporednostni

Ime	**Opis**
Aktivni objekt (ang. Active object)	Loči izvajanje metode od njenega klica. Cilj je uvesti vzporednost, tako da asinhrono kličemo metode in da planer(ang. Sheduler) obravnava zahteve.
Prekladanje (ang. Balking)	Akcijo izvedemo samo takrat ko je objekt v določenem stanju.
Prevajalne lastnosti (ang. Building properties)	Kombiniramo več opazovalcev, ki nastavljajo lastnosti v posameznih objektih tako, da so sinhronizirani ali nadzorovani v nekem smislu.
Zaklepanje z	Zmanjšamo količino akcij, ki se morajo zgoditi predem lahko

dvojnim preverjanjem (ang. Double-checked locking)	uporabimo zaklepanje.
Dogodkovno asinhronski (ang. Event-based asynchronous)	Obravnava probleme, ki nastanejo pri asinhronem vzorcu v več-nitnih sistemih.
Varovano prekinjanje (ang. Guarded suspension)	Obravnava operacije, ki so odvisne od zaklepanja in določenega stanja v objektu.
Zaklepanje (ang. Lock)	Ena nit zaklene vir tako, da ga ostale niti ne morejo uporabiti.
Sporočila (ang. Messaging design pattern)	Dovoljuje izmenjavo informacij med posameznimi komponentami in aplikacijami.
Objekt opazovalec (ang. Monitor object)	Objekt katerega metode so podvržene medsebojnemu izključevanju in s tem prepreči, da bi istočasno do ene metode dostopalo več objektov.
Reaktor (ang. Reactor)	Omogoča asinhronski dostop do virov, ki jih je potrebno uporabljati sinhrono.
Bralno-pisalno zaklepanje (ang. Read-write lock)	Dovoljuje istočasne bralne dostope vendar le en pisalni dostop naenkrat.
Planer (ang. Scheduler)	Nadzira, kdaj lahko niti izvajajo kodo namenjeno za eno nit istočasno.
Bazen niti (ang. Thread pool)	Skupina niti, ki je kreirana za posamezne naloge. Običajno so organizirane v vrsti. Po navadi je veliko več zahtev kot pa je niti.
Shranjevanje glede na nit (ang. Thread-specific storage)	Statičen ali globalen spomin lokaliziran na nit.

Tabela 4: Vzporednostni vzorci

2.2 Uporaba načrtovalskih vzorcev na aplikaciji

V naši aplikaciji je uporabljenih nekaj načrtovalskih vzorcev, ki vidno izboljšajo obliko ali delovanje aplikacije in so jasno razvidni pri analizi aplikacije. Glede na tip programske opreme, ki jo razvijamo, so nekateri načrtovalski vzorci pomembnejši za implementacijo, kot drugi. Za spletne aplikacije, kjer je pomembna skalabilnost so pomembnejši vzorci, ki skrbijo za vzpostavljaje in dodeljevanje računalniških virov, kot so povezave do baze in spletnih storitev. Zato je bolj smotrno implementirati načrtovalske vzorce, ki opisujejo bazene (ang. Pool) omenjenih virov, kot recimo vzorce, ki opisujejo zaklepanje in sinhronizacijo. Slednji so bolj pomembni pri kakšnih sistemskih programih, kot so uporabniški vmesniki za spreminjanje konfiguracijskih datotek. Iz teh razlogov so nekateri načrtovalski vzorci lepše vidni kot drugi in nekateri še niso prišli na vrsto za implementacijo.

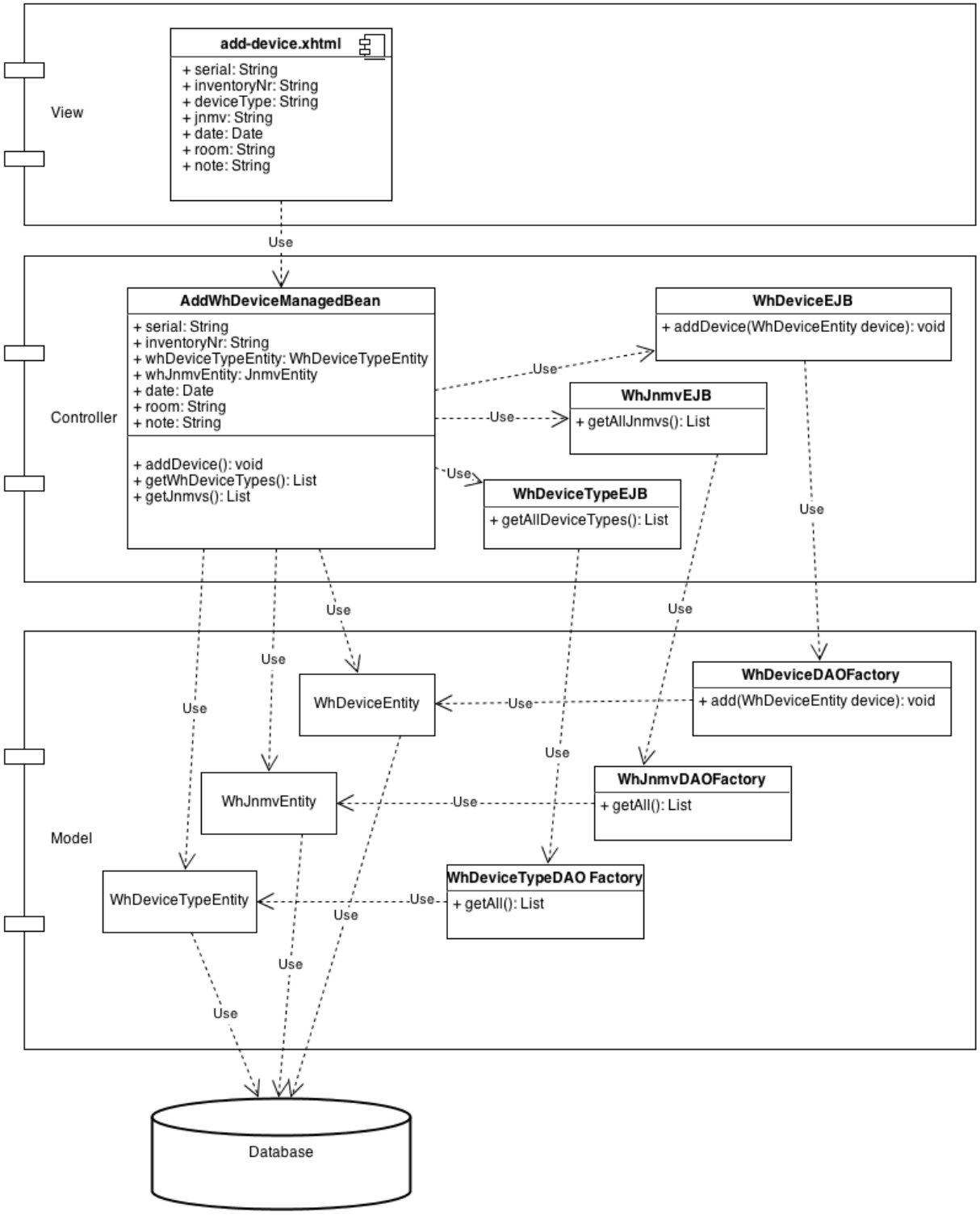
2.2.1 Ustvarjalni vzorci

Od ustvarjalnih vzorcev smo uporabili metodo tovarne, leno inicializacijo, multiton, bazen objektov, edinec in pridobitev vira v inicializaciji. Metodo tovarne smo uporabili v DAO modulu aplikacije. Ta modul je sestavljen iz JPA entitet, DAO razredov, DAO tovarn in vmesnikov za DAO razrede ter DAO tovarne. Tovarna je razred, ki ustvarja druge razrede. Tako smo ločili metode, ki delujejo nad eno instanco JPA entitete v DAO razred in metode, ki delujejo nad več instancami JPA entitete v DAO tovarno. V DAO razredu so metode spreminjanja in brisanja JPA entitet in v DAO tovarni so metode iskanja in dodajanja. Poleg metod iskanja so v DAO tovarnah metode za pridobivanje DAO razredov. DAO razred iz tovarne dobimo tako, da pokličemo metodo, ki vrača DAO razred in sprejema identifikacijsko številko vnosa v bazi, katera napolni JPA entiteto. Statične metode za iskanje po JPA entitetah smo prestavili v tovarne, saj je potrebno za statične metode vsakič posebej pripraviti povezavo do baze, ki se sedaj drži znotraj DAO tovarne. Lena inicializacija pride do izraza pri pridobivanju podatkov iz podatkovne baze in prikazu končnemu uporabniku. Dobimo ogromno količino podatkov, katerih ne moremo vseh prikazati. Namesto, da takoj zgradimo prikaz velike količine podatkov, ki traja nekaj časa, iz podatkovne baze preberemo manj podatkov in po potrebi naredimo še eno poizvedbo, če potrebujemo več podatkov. Ta načrtovalski vzorec se lepo vidi pri razdeljevanju tabele podatkov na več strani. Za vsako stran ki jo želimo prikazati naredimo novo poizvedbo in prenesemo manj podatkov naenkrat. Uporaba tega načrtovalskega vzorca v spletni aplikaciji navadno pomeni uporabo AJAX tehnologije, ki s pomočjo klicev iz odjemalca preko HTTP protokola zahteva dodatne podatke iz strežnika. Načrtovalski vzorec multiton zahteva, da je objekt imenovan in da imamo globalno točko dostopa do njega. Na tak način dostopamo do EJB objektov. EJB objekti so navadni Java objekti, ki so anotirani z anotacijami, ki jih prepozna aplikacijski strežnik. Aplikacijski strežnik si pri sebi ustvari eno ali več instanc EJB objekta in pripravi globalno točko za dostop do njih, tako lahko do ELB-jev dostopamo od koder koli v aplikaciji. Za dostop do njih potrebujemo le vmesnik željenega objekta. EJB objekti so imenovani in jih dobimo tako, da aplikacijskemu strežniku preko anotacije povemo, da potrebujemo objekt. Bazen objektov nam zagotavlja, da objektov ne kreiramo ob vsaki zahtevi, ampak jih

recikliramo in ko pride nova zahteva po objektu, se le ta pridobi iz bazena že kreiranih objektov ter tako prihranimo čas kreiranja objektov. Pri pridobitvi vira v inicializaciji skrbimo, da vira ne zasedamo predolgo. Ta načrtovalski vzorec je viden pri kontrolerjih za JSF strani. Upravljalna zrna (ang. Managed bean), oziroma CDI zrna (ang. Bean) imajo določen življenjski cikel (ang. Scope). Strežniku povemo, kako dolgo potrebujemo željeni objekt. Objekt se lahko zavrže po izvedbi posamezne zahteve, poteku seje, ali pa ostane aktiven celotno delovanje aplikacije. Načrtovalski vzorec edinec opisuje, da je za vzdrževanje globalnih vrednosti, ki morajo biti zmeraj dostopne uporabimo objekt, ki je aktiven celotno delovanje aplikacije in se uporabniku ponudi zmeraj isto instanco tega objekta. Načrtovalski vzorec edinec je najbolje viden pri edinec tipu EJB-ja. Edinci navadno skrbijo za branje konfiguracijskih datotek in držanje podatkov prebranih iz njih.

2.2.2 Strukturni vzorci

Od strukturnih načrtovalskih vzorcev smo uporabili adapter, prednji krmilnik in modul. Načrtovalski vzorec adapter opisuje spremembo splošnega vmesnika razreda z drugim vmesnikom. Uporaba adapterja se vidi pri uporabi več vmesnikov za dostop do EJB objekta. Za našo aplikacijo sedaj še ne potrebujemo dveh vmesnikov, vendar pa EJB objekt vseeno zahteva vsaj en vmesnik. Načrtovalski vzorec prednji krmilnik predstavlja centralno vstopno točko za serviranje zahtev. Prednji krmilnik je del MVC arhitekture, ki predstavlja ločevanje aplikacije na tri dele. Uporaba MVC arhitekture je prikazana z razrednim diagramom za sliki 1. Model vsebuje programsko kodo za dostop do podatkovne baze. Prednji krmilnik pripravi podatke za prikaz in pogled (ang. View) prikaže podatke, ki jih je pripravil prednji krmilnik uporabniku. V našem primeru prednji krmilnik predstavljajo upravljalna zrna in CDI zrna, ki pripravljajo podatke za prikaz v JSF straneh. Načrtovalski vzorec modul predstavlja razdelitev programa na več delov, lahko tudi projektov. Smotrno je ločiti aplikacijo na strežnik in na odjemalca, v tem primeru se na strani odjemalca znebimo dostopa do podatkovne baze. To v našem primeru dosežemo tako, da ločimo modul z EJB objekti in spletnega odjemalca, v katerem so prednji krmilniki in spletne strani. V EJB modulu dostopamo do DAO projekta, ki je prav tako modul zase.



Slika 1: Razredni diagram za dodajanje naprave

2.2.3 Značajski vzorci

Od značajskih vzorcev smo uporabili pregledovalnik, memento, prazen objekt, specifikacijo, stanje in predlogo (ang. Template). Načrtovalski vzorec pregledovalnik opisuje, da se pri uporabi podatkovnih struktur, kot so polja in seznami, ne poglabljamo v njihovo sestavo,

ampak za dostop do podatkov uporabimo tako imenovani pregledovalnik, ki nam vrača posamezne objekte iz te podatkovne strukture. V našem primeru metode, ki vračajo podatkovne strukture vrnejo strukturo tipa java.Util.Collection, ki je dostopna preko vmesnika. Objekte iz nje dobimo v for zanki, ki se s pomočjo pregledovalnika sprehodi skozi strukturo ali pa jo podamo direktno v JSF, JSF servlet jo prebere in podatke prikaže v spletni strani. Načrtovalski vzorec memento opisuje ohranitev stanja objekta, zato da ga lahko kasneje uporabimo. Prejšnje stanje objekta shranimo zato, da ga lahko ob neuspeli operaciji obnovimo. V našem primeru se uporaba vzorca memento vidi pri uporabi transakcij nad podatkovno bazo. V primeru da spremembe nad podatkovno bazo niso uspešne, je kljub nekaterim spremembam možno obnoviti prejšnje stanje. Načrtovalski vzorec praznega objekta opisuje, da praznih (ang. Null) objektov v aplikaciji ne smemo dovoljevati. V našem primeru smo na strani odjemalca poskrbeli, da ne uporablja praznih objektov, če bi pa nekdo drug uporabil modul naše aplikacije, metode takšen objekt prestrežejo in vrnejo izjemo. Izjeme v primeru praznih objektov moramo nadzorovati samo za to, da imamo nadzor nad tem, kaj se zgodi v primeru, da se pojavijo napačni podatki. Načrtovalski vzorec specifikacija opisuje, da moramo pri razvoju aplikaciji imeti v naprej znana pravila programiranja in vse zahtevane funkcionalnosti. V našem primeru so v specifikaciji opisan sistemska arhitektura in orodja, ki jih lahko uporabimo, oblika različnih tipov razredov, uporaba načrtovalskih vzorcev, pravila testiranja aplikacije in vse funkcionalnosti, ki jih mora aplikacija implementirati. Načrtovalski vzorec stanje opisuje, da se lahko nek objekt obnaša na več različnih načinov v odvisnosti od vhodnih podatkov. Za pošiljanje in posojanje naprav uporabimo isti kontroler. Glede na stran vstopa do funkcionalnosti kontrolerja se kontroler odloči, kako bo predstavil podatke in kaj bo naredil ob klicu akcij. Načrtovalski vzorec predloga opisuje uporabo ogrodja, v katerega postavimo svojo vsebino. V predlogi so definirane stvari, ki se ne spreminjajo in prostor za dinamično vsebino. Z uporabo predloge se rešimo ponavljanja programske kode v več datotekah. V našem primeru smo uporabili predlogo pri ustvarjanju uporabniškega vmesnika. Definirali smo glavo, nogo in stranski meni, ki se morajo prikazati ob klicu vsake strani. Nahajajo se v datoteki, ki zgradi elemente predloge okoli dinamične vsebine.

2.2.4 Vzporednostni vzorci

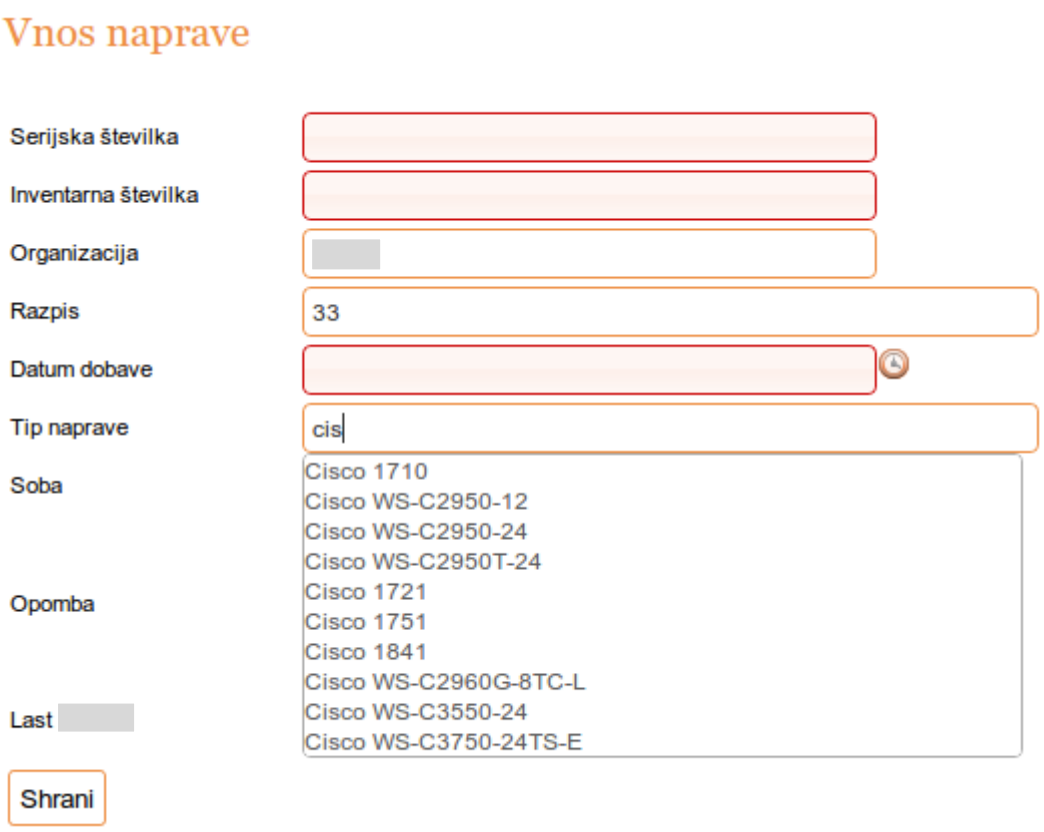
Od vzporednostnih vzorcev smo uporabili prevajalne lastnosti, zaklepanje in bralno-pisalno zaklepanje. Načrtovalski vzorec prevajalne lastnosti opisuje globalne lastnosti, ki nosijo podatke o raznih virih, na katere se mora povezovati aplikacija. V našem primeru so te lastnosti zapisane konfiguracijskih datotekah. Ime datoteke ima obliko ime_strežnika.properties. V objektu, ki dostopa do teh datotek, se prebere ime strežnika, na katerem teče aplikacija in se naloži lastnosti za posamezni strežnik. Več datotek imamo, ker je potrebno aplikacijo z različnimi lastnostmi poganjati na lokalnem strežniku, testnem strežniku in produkcijskem strežniku. Načrtovalska vzorca zaklepanje in bralno-pisalno zaklepanje preprečujeta, da bi se izvajalo več pisalnih dostopov nad istim objektom in bi se tako izgubljali podatki tistega, ki prej zaključi pisanje. Bralno-pisalno zaklepanje se izvaja v sklopu transakcij na MySQL strežniku. Za odpiranje in zapiranje transakcij, njihovo shranjevanje in zavračanje skrbi aplikacijski strežnik. Bralni dostopi so vsi dovoljeni sočasno,

medtem, ko se lahko sočasno zgodi le en pisalni dostop. Transakcijo in potrebnost transakcije definiramo s pomočjo anotacij na metodah EJB objektov (Primer: @TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)).

3 Opis spletne aplikacije

Za praktičen prikaz uporabe načrtovalskih vzorcev in uporabe Jave EE smo razvili spletno aplikacijo. Za lažji opis tehnologij uporabljenih v razvoju aplikacije si bomo najprej pogledali opis delovanja spletne aplikacije.

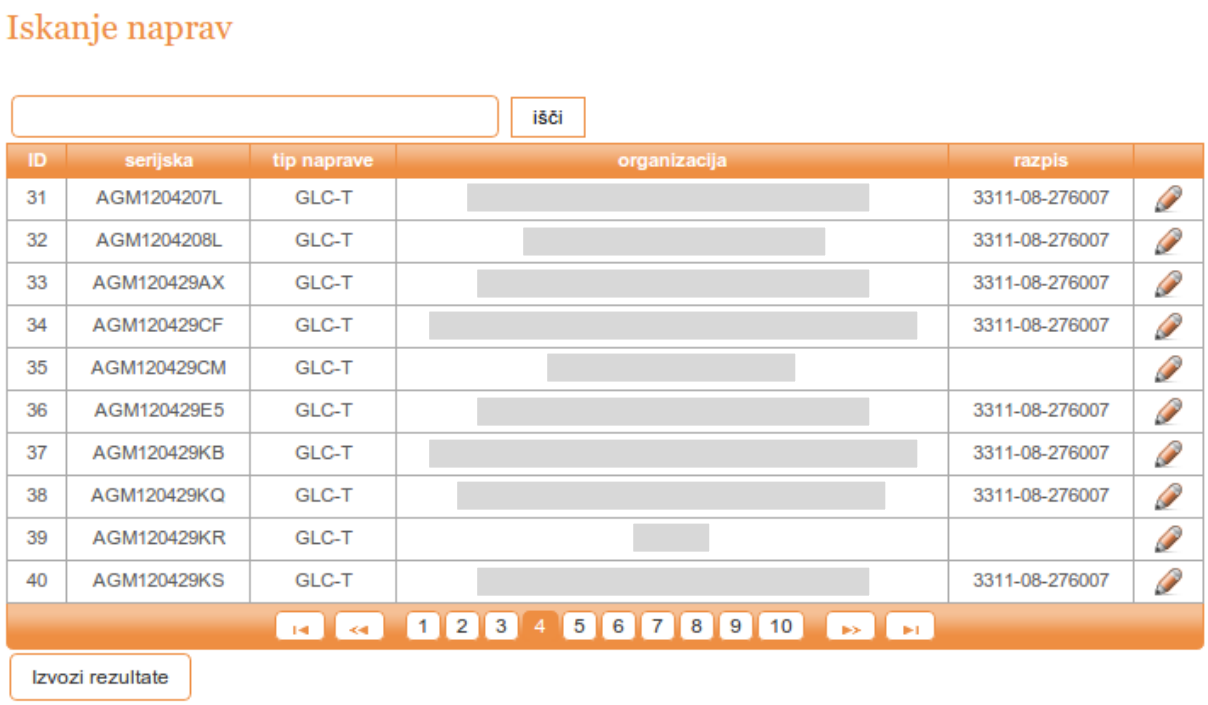
V podjetju, ki se ukvarja s konfiguracijo strojne opreme, to so predvsem usmerjevalniki (ang. Router), stikala (ang. Switch) in ostala omrežna oprema, nastopi problem nadzora nad lokacijo opreme. V našem primeru je situacija taka, da na podlagi neke letne vsote denarja rezervirane za nabavo opreme nabavimo opremo, jo konfiguriramo za končnega uporabnika (organizacijo) in nato opremo pošljemo. Aplikacija mora omogočati vpis novih naprav in urejanje le teh (slika 2).



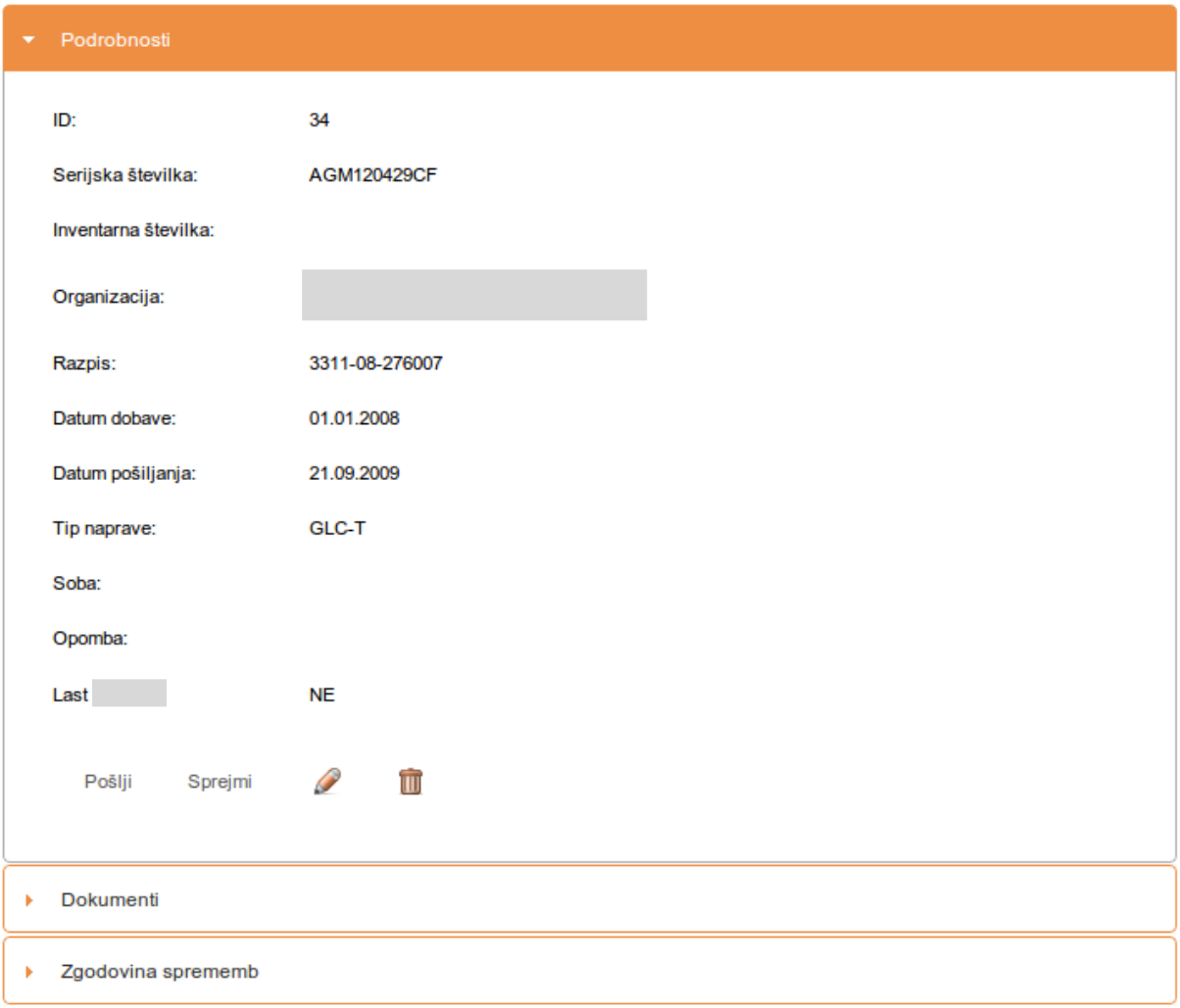
Slika 2: Vnos naprave

Pri vsakem vpisu naprave določimo tip naprave in razpis preko katerega je bila nabavljena. Razpis ni obvezen, saj med opremo spadajo tudi razni kabli, vmesniki itd., ki pa niso nujno vezani na razpis. V primeru, da moramo opraviti začeten vnos naprav, lahko uvozimo naprave iz XLS datoteke. Uvoz iz XLS datoteke lahko uporabimo tudi kasneje, če moramo vpisati v aplikacijo večjo količino naprav naenkrat. Razpisi in tipi naprav so zapisani vsak v svojem šifrantu, ki morata imeti možnost urejanja.

Preden lahko pošljemo napravo jo moramo seveda najti. Ko v iskalniku naprav (slika 3) najdemo napravo, kliknemo na podrobnosti naprave (slika 4), kjer vidimo podatke o napravi, zgodovino pošiljanja naprave, možnost urejanja in brisanja ter seveda pošiljanja oziroma sprejemanja.

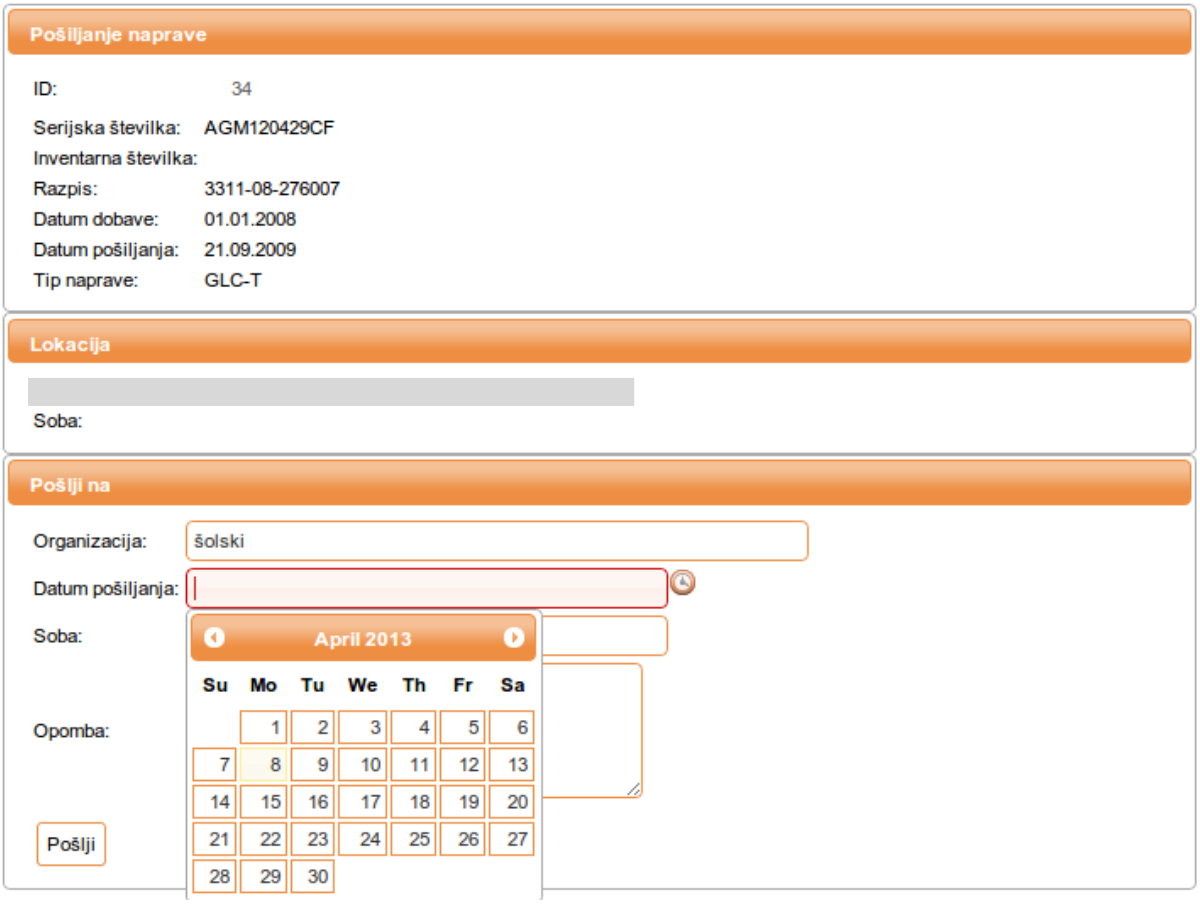


Slika 3: Iskanje naprav



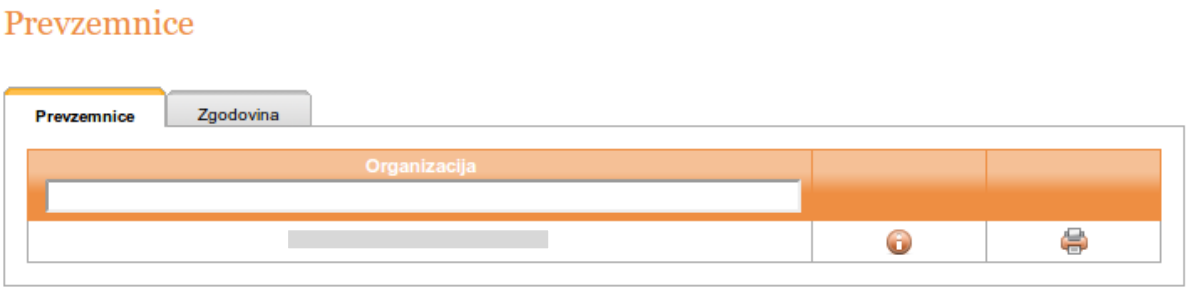
Slika 4: Podrobnosti naprave

Pred izbrisom naprave nas aplikacija vpraša če to res želimo storiti. V primeru urejanja dobimo podoben obrazec kot je za dodajanje naprave, le da so podatki že vpisani. Če izberemo pošiljanje, se prestavimo na naslednjo stran (slika 5), na kateri izberemo organizacijo (prejemnika) in datum (čas pošiljanja).



Slika 5: Pošiljanje naprave

Pri sprejemu naprave se zgodi podobno, le da ne vpisujemo organizacije. Pri pošiljanju, se vključi v seznam poslanih naprav na določeno organizacijo. Ta seznam se uporabi za tiskanje prevzemnice. Prevzemnica je vezana na organizacijo in na njej so vse naprave, ki jih želimo poslati na neko organizacijo, vendar za njih še ni bila natisnjena prevzemnica. Ko pošljemo vse naprave za neko organizacijo, gremo na drug vmesnik, na katerem se pojavijo vse organizacije, ki smo jim na novo poslali opremo (slika 6). Z izbiro organizacije, se nam na računalnik prenese prevzemnica v PDF formatu (slika 7).



Slika 6: Generiranje prevzemnice

PREVZEM OPREME

Organizacija:

PREVZETA OPREMA

Opis opreme	Serijska	Inventarna št.	Last Arnesa
GLC-LH-SM	230489790	23665	DA
WIC 1B S/T	19911241		NE

Datum prevzema:

Opremo predal:

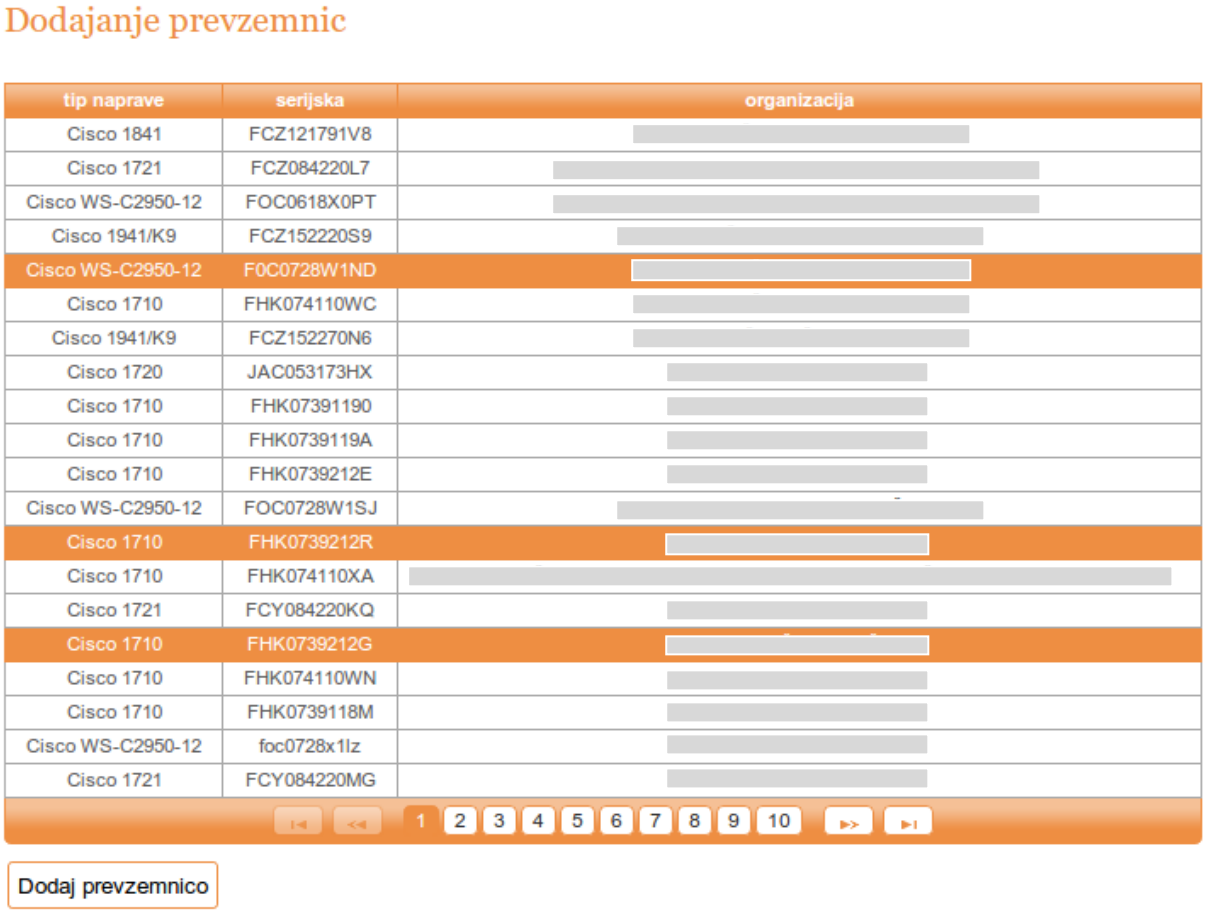
Boštjan Štor

Opremo prevzel:

Slika 7: Prevzemnica

To prevzemnico natisnemo, jo podpišemo in skupaj z napravami pošljemo na organizacijo. Naprav ne moremo sprejemati nazaj, dokler ni v sistemu podpisane prevzemnice od prejemnika. Za dodajanje prevzemnic k napravam potrebujemo vmesnik (slika 8), na katerem

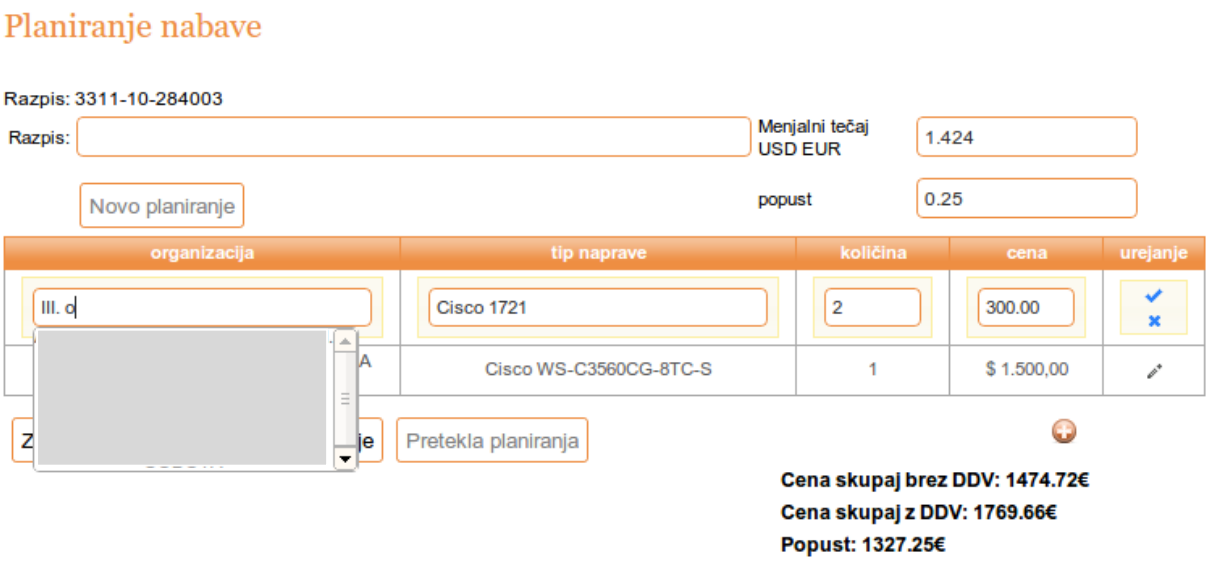
se pojavijo vse naprave, ki so bile poslane in še nimajo aktualne prevzemnice, vezane na njih.



Slika 8: Dodajanje prevzemnic k napravam

Šele ko prejmemo od prejemnika vrnjeno podpisano prevzemnico in je le ta pripeta napravi v sistemu, je možno napravo sprejeti nazaj na naše podjetje. V primeru, da prevzemnico založimo ali jo moramo iz kakršnega koli razloga ponovno natisniti, potrebujemo tudi zgodovino prevzemnic.

S tem zaključimo del aplikacije, ki je zadolžen za evidenco lokacije naprave. Naslednji del je planiranje nabave naprav (slika 9). Potrebujemo vmesnik, v katerega je možno vpisati razpis nabavljanja opreme, pretvornik med valutama evro in ameriški dolar, ker je cena naprave v ameriških dolarjih, naš proračun pa v evrih. Pri nakupu večje količine opreme pogosto dobimo popust, ki ga moramo tudi zabeležiti. Ko imamo vse te podatke, lahko začnemo vpisovati naprave na seznam. Element seznama mora vsebovati organizacijo na katero bomo poslali napravo, tip naprave, ki ga nabavljamo, količino naprav in ceno naprave v dolarjih. Ko smo z načrtovanjem končali, podatke izvozimo v XLS datoteko in ga zaključimo. V primeru napake ali premenbe naprav lahko iz seznama zaključenih načrtovanj izberemo načrtovanje za nadaljnje urejanje. Pod tabelo se nam mora izpisovati skupen znesek nabave naprav brez DDV v evrih, skupen znesek z DDV v evrih in skupen znesek z DDV v evrih od katerega je odštet popust.



Slika 9: Planiranje nabave

Sledi prikaz zaloge naprav pri nas (slika 10). Izpiše se tabela vseh tipov naprav, njihovo število, število poslanih, zaloga in rezervacije. Ob kliku na napravo se odpre nova stran z možnostjo rezervacije naprave za organizacijo in izpisom vseh rezervacij za to napravo (slika 11). Na tej strani je mogoče poleg dodajanja rezervacij, tudi rezervacije brisati in potrjevati. Rezervacija ima določeno veljavnost, če organizacija v določenem času ne prevzame rezervirane naprave, le-ta poteče. Ker so rezervacije vezane na organizacijo potrebujemo še iskalnik organizacij.

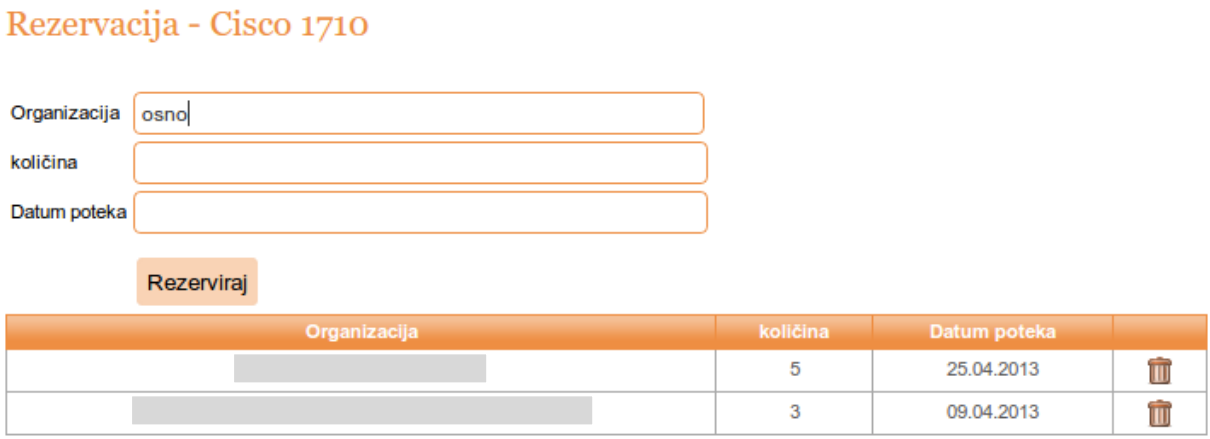
Ob izbiri organizacije iz seznama, iskalnik prikaže vse naprave določene organizacije, njene rezervacije in izposojene naprave. Te povezave od posamezne naprave nas privedejo do podrobnosti tudi izposojenih naprav. Rezervacije pa lahko potrdimo, da niso več aktualne, ali pa odpremo podrobnosti rezervacij.

Zaloga naprav

tip naprave	Skupaj	Poslano	rezervacije/zaloga	
Cisco 1710	78	49	8/29	
Cisco WS-C2950-12	38	33	0/5	
Cisco WS-C2950-24	6	2	0/4	
Cisco WS-C2950T-24	12	10	0/2	
Cisco 1721	17	5	0/12	
Cisco 1751	1	0	0/1	
Cisco 1841	173	170	2/3	
Cisco WS-C2960G-8TC-L	139	119	0/20	
Cisco WS-C3550-24	5	2	3/3	
Cisco WS-C3750-24TS-E	7	4	0/3	
Cisco WS-C3750G-12S	1	0	0/1	
Cisco WS-C3560G-24TS-E	9	6	0/3	
Cisco WS-C4503	6	4	0/2	
Cisco WS-C4503E	5	5	0/0	
GLC-LH-SM	108	102	0/6	
GLC-SX-MM	12	10	0/2	
GLC-T	28	23	0/5	
1000 BASE-LX GBIC	4	0	0/4	
WIC 1B S/T	8	3	0/5	
WIC 1ENET	6	2	0/4	

1 2 3

Slika 10: Prikaz zaloge naprav



Slika 11: Rezervacija naprav

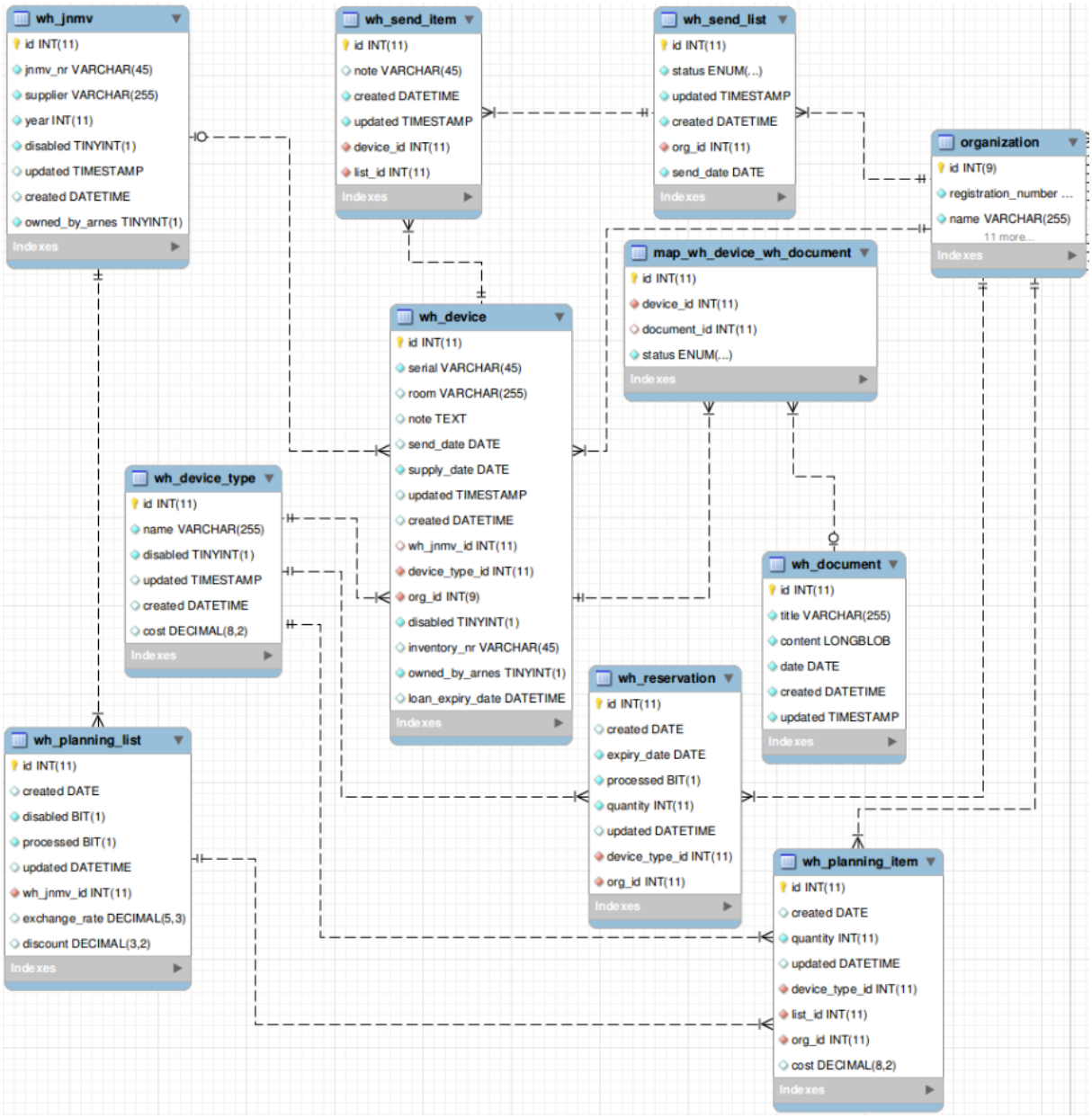
Zaradi pomembnosti, da aplikacija pokaže nujne akcije čim prej, smo na začetni strani dodali tabelo preteklih izposoj in rezervacij.

4 Podatkovna baza

Podatkovna baza, ki jo uporablja naša spletna aplikacija, ima skupaj preko 50 tabel, ker jo uporabljajo tudi druge aplikacije. Naša aplikacija uporablja 10 tabel, ki imajo zraven še tabele v katere se piše zgodovina sprememb. Poleg tega uporablja tudi tabelo, v kateri so zapisane organizacije in tabele preko, katerih preverjamo uporabnike ob prijavi v aplikacijo. Za ponudnika podatkovne baze smo izbrali MySQL, saj je zelo razširjen, brezplačen in ima dobro podporo uporabnikom. Vendar pa je čar aplikacije razvite z Java Enterprise Edition to, da sama aplikacija ne ve kdo je ponudnik baze, kje se le-ta nahaja in kakšna je njena točna zgradba. Lokacijo baze in njenega ponudnika zapišemo v konfiguracijo podatkovnega vira (ang. Data source) na aplikacijskem strežniku, ki naši aplikaciji ponudi bazen povezav (ang. Connection pool) pod določenim imenom. Točno zgradbo baze pa zamaskiramo z vmesnikom za objekten dostop do baze (ORM). V našem primeru bomo uporabili JPA. Obstaja več ponudnikov za JPA in mi bomo uporabili Hibernate. Hibernate bomo uporabili, ker moramo svoje razrede za dostop do podatkovne baze ustvariti znotraj projekta, ki že uporablja Hibernate.

4.1 Model

Najprej predstavimo tabele, ki so primarno namenjene naši aplikaciji (slika 12).



Slika 12: Entitetno-relacijski diagram podatkovne baze

Za lažje ločevanje tabel med raznimi aplikacijami, ima vsaka aplikacija na svojih tabelah predpono. Naše tabele imajo predpono wh_, ki je okrajšava za warehouse (skladišče). Glavna tabela aplikacije je wh_device (tabela 5) v kateri so zapisani podatki o napravah.

Atribut	**Opis**
id	Primarni ključ tabele.
serial	Serijska številka naprave.
room	Soba v kateri se nahaja naprava.

note	Opomba.
send_date	Datum pošiljanja naprave.
supply_date	Datum dobave naprave.
loan_expiry_date	Datum poteka izposoje.
updated	Čas zadnje spremembe nad vrstico.
created	Čas dodajanja vnosa.
disabled	Naprava je lahko označena kot izbrisana.
owned_by_arnes	Naprava je bila kupljena s sredstvi Arnesa.
inventory_nr	Inventarna številka.
wh_jnmv_id	Tuj ključ za povezavo s tabelo wh_jnmv.
device_type_id	Tuj ključ za povezavo s tabelo wh_device_type.
org_id	Tuj ključ za povezavo s tabelo organization.

Tabela 5: Tabela wh_device

Na to tabelo sta vezana šifranta razpisov (wh_jnmv) in tipov naprav (wh_device_type). Podrobnosti obeh šifrantov so vidne v tabelah 6 in 7.

Atribut	**Opis**
id	Primarni ključ tabele.
jnmv_nr	Številka razpisa.
supplier	Dobavitelj.
year	Leto razpisa.
disabled	Razpis je lahko označen kot izbrisan.
owned_by_arnes	Razpis je lahko interne narave.
updated	Čas zadnje spremembe nad vrstico.
created	Čas dodajanja vnosa.

Tabela 6: Tabela wh_jnmv

Atribut	Opis
id	Primarni ključ tabele.
name	Ime tipa naprave.
disabled	Tip naprave je lahko označen kot izbrisan.
cost	Cena.
updated	Čas zadnje spremembe nad vrstico.
created	Čas dodajanja vnosa.

Tabela 7: Tabela wh_device_type

Na njo je vezana tudi tuja tabela organizacije (organization), preko katere vemo, kje se nahaja naprava. Tabela organization (tabela 8) vsebuje več atributov, vendar naša aplikacija uporablja le ime organizacije.

Atribut	Opis
id	Primarni ključ tabele.
name	Ime organizacije.

Tabela 8: Tabela organization

Omenili smo, da naprave pošiljamo, zato imamo tabelo wh_send_list (seznam pošiljanja). Ta tabela je tudi vezana na organizacijo, v njej nastavimo status ali je to trenuten seznam za določeno organizacijo ali je seznam neaktualen (tabela 9).

Atribut	Opis
id	Primarni ključ tabele.
status	Nastavimo ali je seznam trenuten ali neaktualen.
updated	Čas zadnje spremembe nad vrstico.
created	Čas dodajanja vnosa.
send_date	Datum pošiljanja.
org_id	Tuj ključ za povezavo s tabelo organization.

Tabela 9: Tabela wh_send_list

Zaradi možnosti pojavljanja naprave na več seznamih in en seznam pri več napravah, potrebujemo vmesno tabelo wh_send_item (točka seznama). Podrobnosti tabele so vidne na tabeli 10.

Atribut	Opis
id	Primarni ključ tabele.
note	Opomba.
created	Čas dodajanja vnosa.
updated	Čas zadnje spremembe nad vrstico.
device_id	Tuj ključ za povezavo s tabelo wh_device.
list_id	Tuj ključ za povezavo s tabelo wh_send_list.

Tabela 10: Tabela wh_send_item

Iz teh dveh tabel, wh_send_list in wh_send_item, se generirajo prevzemnice, ki se skupaj z napravo pošljejo organizaciji. Nato potrebujemo tabelo, v katero shranjujemo dokumente za posamezno napravo. Ta tabela se imenuje wh_document in v njej so tako podpisane prevzemnice kot tudi morebitni dodatni dokumenti, ki bi jih lahko dodali napravi (tabela 11).

Atribut	Opis
id	Primarni ključ tabele.
title	Naslov dokumenta.
content	Vsebina.
date	Datum.
created	Čas dodajanja vnosa.
updated	Čas zadnje spremembe nad vrstico.

Tabela 11: Tabela wh_document

Ker je lahko več naprav na enem dokumentu in prav tako več dokumentov na eni napravi potrebujem vmesno tabelo map_wh_device_wh_document (tabela 12).

Atribut	Opis
id	Primarni ključ tabele.

document_id	Tuj ključ za povezavo s tabelo wh_document.
status	Nastavimo status dokumenta, ali je to prevzemnica ali navaden dokument.
device_id	Tuj ključ za povezavo s tabelo wh_device.

Tabela 12: Tabela map_wh_device_wh_document

Preostal nam je še del tabel, ki skrbijo za načrtovanje nabave naprav. Seznami načrtovanja se shranjujejo v wh_planning_list tabelo, na katero je vezana tabela organization (organizacija) in tabela wh_jnmv (razpisi). Tipi naprav in njihova količina pa so zapisani v tabeli wh_planning_item, na katero je vezana tabela wh_planning_list. Slednja združuje izbrane elemente načrtovanja za načrtovanje nabave naprav. Podrobnosti tabel so prikazane v tabelah 13 in 14.

Atribut	**Opis**
id	Primarni ključ tabele.
list_id	Tuj ključ za povezavo s tabelo wh_planning_list.
org_id	Tuj ključ za povezavo s tabelo organization.
device_type_id	Tuj ključ za povezavo s tabelo wh_device_type.
quantity	Količina naprav.
cost	Cena ene naprave.
created	Čas dodajanja vnosa.
updated	Čas zadnje spremembe nad vrstico.

Tabela 13: Tabela wh_planning_item

Atribut	**Opis**
id	Primarni ključ tabele.
processed	Načrtovanje je lahko končano.
disabled	Načrtovanje je lahko označeno kot izbrisano.
created	Čas dodajanja vnosa.
updated	Čas zadnje spremembe nad vrstico.

wh_jnmv_id	Tuj ključ za povezavo s tabelo wh_jnmv.
exchange rate	Pretvorni tečaj iz ameriških dolarjev v evre.
discount	Popust.

Tabela 14: Tabela wh_planning_list

V tabelo wh_reservation so zapisane rezervacije tipov naprav za organizacijo. Tabelo wh_reservation potrebujemo zato, da pravočasno ugotovimo, da je zmanjkalo naprav (tabela 15).

Atribut	**Opis**
id	Primarni ključ tabele.
device_type_id	Tuj ključ za povezavo s tabelo wh_device_type.
org_id	Tuj ključ za povezavo s tabelo organization.
quantity	Količina rezerviranih naprav.
processed	Nastavimo lahko, da so bile rezervirane naprave poslane.
expiry_date	Datum poteka rezervacije.
created	Čas dodajanja vnosa.
updated	Čas zadnje spremembe nad vrstico.

Tabela 15: Tabela wh_reservation

Poleg tega preko tabel user_arnes (uporabniki) in person (oseba) ugotovimo kdo je prijavljen v aplikacijo in lahko operiramo z njegovimi osebnimi podatki.

4.2 MySQL

MySQL [9] je sistem za upravljanje relacijskih podatkovnih baz (RDBMS – Relational Database Management System). Omogoča dostop več uporabnikov naenkrat do več podatkovnih baz. MySQL je najbolj uporabljen odprtokodni ponudnik RDBMS za podatkovne baze tipa SQL (Structured Query Language). Programska koda je dostopna pod licenco GNU GPL (General Public Licence).

4.3 Java Persistence API

Java Presistence API (JPA) je objektni vmesnik za SQL podatkovne baze. Deluje tako, da je vsaka tabela predstavljena z Java objektom, ki je opremljen z anotacijami s pomočjo katerih JPA ugotovi, kako izgleda tabela, ki jo objekt (entiteta) predstavlja. JPA se nahaja v paketu

javax.persistence. V primeru, da uporabljamo navaden kontejner za Java spletne aplikacije, kot je Apache Tomcat, moramo definirati EntityManagerFactory, s katerim povemo, kje se nahaja podatkovna baza, kdo je ponudnik in kakšni so avtentifikacijski podatki za dostop do baze. EntityManagerFactory zgradi objekt EntityManager, s katerim kličemo operacije nad bazo. V našem primeru pa za povezavo do baze skrbi aplikacijski strežnik GlassFish. Na njem definiramo podatkovni vir, do katerega dostopamo z anotacijo PersistenceContext, katere parameter unitName vsebuje ime podatkovnega vira. Ta anotacija je pomembna zato ker nam aplikacijski strežnik preko nje zgradi EntityManager-ja, preko njega kličemo operacije nad podatkovno bazo.

Za povezavo z bazo preko JPA potrebujemo datoteko z nastavitvami persistence.xml, kot je vidno na sliki 13, v kateri je zapisano ime podatkovnega vira, ki je definiran na aplikacijskem strežniku.

```
<persistence-unit name="aris" transaction-type="JTA">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>arisJtaDataSource23</jta-data-source>
    <properties>
        <property name="hibernate.show_sql" value="false"/>
        <property name="hibernate.format_sql" value="true"/>
```

Slika 13: Del datoteke presistence.xml

Name atribut značke persistence-unit predstavlja ime, ki se bo uporabljalo znotraj aplikacije v EJB-jih, kjer je potrebna povezava s podatkovno bazo. Značka provider nam pove, katerega ponudnika bomo izbrali za JPA in značka jta-data-source nam pove ime podatkovnega vira definiranega na aplikacijskem strežniku. Za JPA obstaja več ponudnikov, zaradi možnosti preprostega beleženja sprememb za bazi smo izbrali Hibernate [10], ki v svojem jedru vsebuje projekt Envers za beleženje zgodovine.

4.3.1.1 Beleženje zgodovine

V aplikaciji na več mestih želimo pregled nad spreminjanjem vnosov v bazi. Predvsem želimo videti kako so se pošiljale naprave iz organizacije na organizacijo in morebitne spremembe opomb, inventarnih številk in podobno. Za ta namen smo uporabili projekt Envers, ki je del jedra paketa Hibernate. V nastavitveni datoteki za JPA, persistence.xml poleg imena podatkovnega vira za povezavo s podatkovno bazo definiramo, kako želimo beležiti zgodovino. To nam prikazuje slika 14.

```
<property name="hibernate.ejb.event.post-insert" value="org.hib
<property name="hibernate.ejb.event.post-update" value="org.hib
<property name="hibernate.ejb.event.post-delete" value="org.hib
<property name="hibernate.ejb.event.pre-collection-update" valu
<property name="hibernate.ejb.event.pre-collection-remove" valu
<property name="hibernate.ejb.event.post-collection-recreate" v
```

Slika 14: Imena nastavitev za Hibernate Envers

Modul Envers je nastavljen tako, da se za vsako tabelo, katere entiteta je označena z anotacijami projekta Envers, ustvari še ena tabela, ki vsebuje vsa polja tabele in še dva dodatna, ki določata tip spremembe v bazi ter identifikacijsko številko revizije, v kateri se je sprememba zgodila. Tabele za beleženje zgodovine imajo končnico _AUD, za brskanje po zgodovini sprememb pa uporabimo objekta AuditReader in AuditQuery iz paketa org.hibernate.envers. Na sliki 15 je prikazan primer brskanja po zgodovini sprememb.

```
AuditReader reader = AuditReaderFactory.get(entityManager);

AuditQuery deviceQuery = reader.createQuery()
        .forRevisionsOfEntity(WhDeviceEntity.class, false, true)
        .add(AuditEntity.id().eq(entity.getId()));
List revDeviceList = deviceQuery.getResultList();
```

Slika 15: Primer iskanja zgodovine za eno napravo

4.4 DAO projekt

Zaradi uporabe enotne baze za več projektov imamo v ozadju en projekt, v katerem so napisane funkcionalnosti za pridobivanje podatkov iz podatkovne baze. V njem so definirane entitete (javax.persistence.Entity) katere zna JPA povezati z bazo. Akcije nad entitetami se

poznajo na vsebini baze. Projekt je sestavljen iz paketa entitet, DAO razredov, tovarn (ang. Factory) in vmesnikov za DAO razrede in DAO tovarne. V DAO razredu so definirane akcije nad eno entiteto, ki jo želimo spremeniti ali izbrisati. Primer klica metode v DAO razredu je prikazan na sliki 17. V DAO tovarni so definirane akcije iskanja entitet in dodajanja entitet. Primer klica metode v DAO tovarni je prikazan na sliki 16. V primeru, da rabimo večkrat funkcijo, ki na nek specifičen način išče po entitetah in ni smotrno pripravljati parametrov za splošno iskalno funkcijo znotraj kontrolerja na večih mestih v aplikaciji, napišemo specifično funkcijo, ki išče po točno določenem parametru. V tem projektu morajo biti funkcije kar se da kratke, v njih ni obdelave podatkov, ampak le posredovanje podatkov v kontroler. Vsaka funkcija mora v log zapisati svoj začetek in svoj konec. Iskalne funkcije, ki vračajo seznam entitet, morajo vračati java.util.Collection. Vsaka funkcija mora biti testirana glede na delovanje pri pravilnih vhodnih podatkih in pri praznih podatkih. V primeru, da damo za parameter funkcije null mora le-ta prožiti IllegalArgumentException.

Zato, da v DAO razredih nimamo statičnih metod za iskanje in kreiranje entitet, smo ustvarili DAO tovarne, preko katerih so dostopni DAO objekti za delo nad specifično entiteto (spreminjanje parametrov entitete in brisanje).

```
@Override
public Integer getDevicesSize(String searchString) {
    WhDeviceDAOFactory deviceDAOFactory =
            new WhDeviceDAOFactory(em, ctx.getCallerPrincipal().getName());
    return deviceDAOFactory
            .searchDevicesSize(searchString);
}
```

Slika 16: Primer klica iskalne funkcije za naprave

```
@Override
public void editDevice(WhDevice whDevice) throws AObjectNotExist, AObjectNotValid {
    WhDeviceDAOFactory deviceDAOFactory =
            new WhDeviceDAOFactory(em, ctx.getCallerPrincipal().getName());
    WhDeviceDAO whDeviceDAO =  deviceDAOFactory.getDao(whDevice.getId());
    whDeviceDAO.update(whDevice);
}
```

Slika 17: Primer klica DAO projekta za urajnje naprave

4.4.1 Testiranje

Za testiranje tega projekta uporabljamo jUnit. Testiramo posebej DAO razrede in DAO tovarne. Funkcije v testnih razredih anotiramo z:

- @BeforeClass: izvede se pred začetkom poganjanja testov,
- @Before: izvede se pred začetkom vsake testne funkcije,
- @After: izvede se po koncu vsake testne funkcije,

- @AfterClass: izvede se po koncu vseh testov ter
- @Test: definira testno funkcijo.

V testnih razredih (slika 18) anotiramo funkcije z @Before in @Test. Vsak testni razred razširja nadrazred, ki definira povezavo do testne baze, ki se nahaja na lokalnem sistemu. V tem nadrazredu se v funkciji @BeforeClass pobriše celotna testna baza in v funkciji @After se pobrišejo vsi podatki iz testne baze. V funkciji @Before nastavimo podatke, ki jih @Test funkcija potrebuje za pravilno delovanje. Za kreiranje potrebnih entitet uporabimo razred EntityCreator, napisan posebno za naše teste, zato da se koda kreiranja datotek ne ponavlja v vsakem testnem razredu. Testi se izvedejo ob vsakem pošiljanju na SVN sistem, ki projekt pošlje v pregled v aplikacijo, ki za nas aplikacijo prevede in požene teste ter nas kasneje obvesti ali se je projekt pravilno prevedel in testiral.

```
public class WhDeviceDaoAddDocumentTest extends WhDeviceDaoTest {

    private WhDocumentEntity documentEntity;

    @Before
    @Override
    public void setUp() throws ArnesException, IOException, SQLException {
        super.setUp();

        documentEntity = EntityCreator.createWhDocumentEntity(em);

        em.getTransaction().commit();
        ArisRevisionAudite.invalidateCurrentModifier();
        ArisRevisionAudite.registerCurrentModifier("WhDeviceDaoTest:" + funcCaller);
        em.getTransaction().begin();
    }

    @Test
    public void addDocumentNormalTest() throws Exception {
        whDeviceDAO.addDocument(documentEntity.getId());

        em.flush();

        String hql = "SELECT m FROM MapWhDeviceWhDocumentEntity AS m WHERE documentId = :docId";
        Query q = em.createQuery(hql);
        q.setParameter("docId", documentEntity);
        assertEquals(1, q.getResultList().size());
    }

    @Test(expected = IllegalArgumentException.class)
    public void addDocumentNullTest() throws Exception {
        whDeviceDAO.addDocument(null);
    }
}
```

Slika 18: Primer testnega razreda

Zgornji primer (slika 18) prikazuje testni razred za dodajanje dokumentov. Metoda setUp pripravi dokument in ga zapiše v testno bazo. Nato ga metoda addDocumentNormalTest doda k določeni napravi. Metoda addDocumentNullTest testira, kaj se zgodi v primeru, če damo

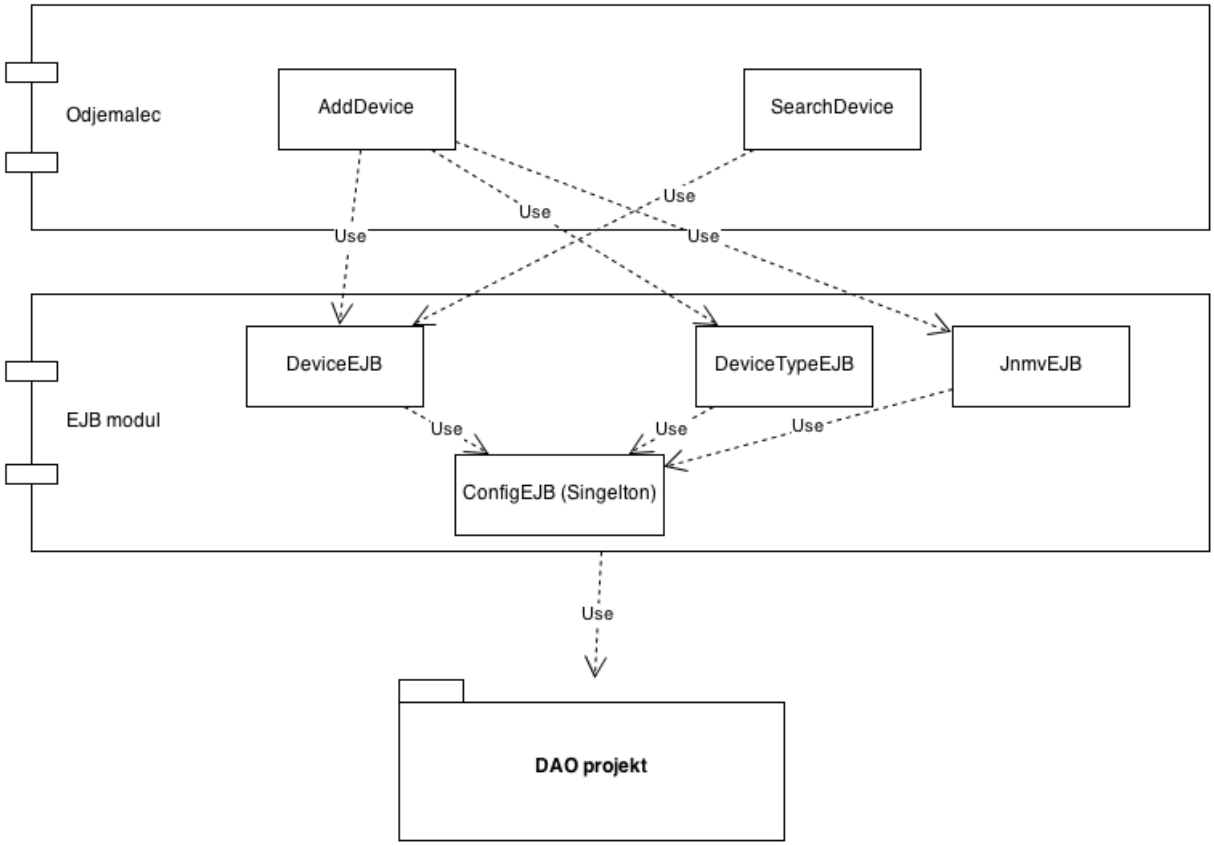
metodi prazen parameter. V tem primeru testna metoda pričakuje napako tipa IllegalArgumentException.

5 Enterprise JavaBeans

Ker smo se odločili razviti spletno aplikacijo v Javi Enterprise Edition velik del aplikacije predstavljajo Enterprise JavaBeans [1]. V tem poglavju jih bomo na kratko predstavili in prikazali njihovo uporabo na primeru naše aplikacije.

Enterpreise JavaBeans ali EJB so del Jave Enterprise Edition, katere implementacija se nahaja v aplikacijskih strežnikih. Tako je možno, da za objekte napisane v skladu z JEE skrbi aplikacijski strežnik in ne aplikacija sama. EJB-ji so navadni javanski objekti, v katere postavimo anotacije (slika 20), ki jih ob nalaganju aplikacije na aplikacijski strežnik, le-ta prepozna in pri sebi ustvari instance teh objektov. Anotacija je rezervirana beseda s predpono @ ki strežniku pove, kaj mora narediti. Obstajajo tri vrste EJB-jev. @Stateless EJB ne shranjuje svojega stanja, to pomeni, da ob vsakem klicu lahko dobimo drugo instanco EJB-ja. @Stateful EJB shrani svoje stanje in ob vsakem klicu dobimo isto instanco. V tem primeru si lahko kakšne podatke shranimo in jih kasneje uporabimo. Bolje je, če uporabimo @Stateless EJB-je saj nam ni potrebno zasedati EJB-ja za celotno trajanje seje. Tretja vrsta EJB-ja je @Singelton, ki se inicializira ob začetku delovanja aplikacije in se uniči ob ustavitvi aplikacije. Uporaben je za branje globalnih nastavitev. EJB potrebuje vmesnik, v katerem so deklaracije njegovih funkcij. EJB lahko kličemo znotraj iste aplikacije ali pa celo iz drugega srežnika. Ker ima verjetno tuj strežnik manj pravic kot naš strežnik, lahko definiramo dva različna vmesnika. Enega @Local (slika 21) za lokalni strežnik in enega @Remote za zunanji strežnik. Za potrebe razvoja aplikacije bodo na začetku vsi EJB-ji lokalni, zato da ni potrebno ob vsaki spremembi poganjati dveh aplikacij. Kasneje pa bo EJB modul prestavljen na drug strežnik.
V naši aplikaciji so vsi EJB-ji @Stateless. Zaščita je narejena z vlogami, ki jih za posameznega uporabnika dobimo od aplikacijskega strežnika. Povezavo z bazo pa dobimo z @PresistenceUnit, ki dobi objekt javax.persistence.EntityManager. Na začetku je bila vsa koda kontrolerja v enem EJB-ju, kar pomeni, da se je ob vsakem zagonu EJB-ja naložilo veliko preveč logike. Kasneje smo razdelili ta EJB na več manjših EJB-jev. Ločevali smo jih po entitetah, za katere skrbijo. Funkcije v EJB-jih kličejo funkcije iz DAO projekta in posredujejo rezultat upravljalnim zrnom ali CDI zrnom. V aplikaciji so vsi EJB-ji @Stateless razen EJB-ja, ki bere datoteko z nastavitvami, slednji je @Singelton. Na EJB-je je možno dodati REST anotacije tako, da razred predstavlja spletno storitev.

EJB razredi v našem primeru predstavljajo vmesnik med kontrolerji in DAO projektom, ki skrbi za komunikacijo s podatkovno bazo. Uporaba EJB-jev ponuja možnost razbitja aplikacije na tri module, odjemalca, EJB modul in DAO projekt, kot je razvidno na sliki 19. Poleg tega EJB-ji predvidijo uporabo načrtovalskega vzorca Multiton, saj priskrbijo globalno točko dostopa do svojih metod preko vmesnikov (ang. Interface). Eden izmed EJB-jev je tipa edinec in hrani vsebino konfiguracijske datoteke za čas izvajanja aplikacije.



Slika 19: Razredni diagram - sestava aplikacije

```
@Stateless(name="WhDeviceServerEJB")
@RolesAllowed("user")
@Interceptors(WhInterceptor.class)
public class WhDeviceServerImpl implements WhDeviceServer {

    @PersistenceContext(unitName = "aris")
    private EntityManager em;
    @Resource
    private SessionContext ctx;

    public WhDeviceServerImpl() {
    }

    @Override
    public Collection<WhDevice> getDevices(String searchString, Integer start, Integer pageSize) {
        WhDeviceDAOFactory deviceDAOFactory = new WhDeviceDAOFactory(em, ctx.getCallerPrincipal().getName());
        return deviceDAOFactory.searchDevices(searchString, start, pageSize);
    }
```

Slika 20: Primer EJB razreda

```
@Local
public interface WhDeviceServer {
    Collection<WhDevice> getDevices(String searchString, Integer start, Integer pageSize);
    Integer getDevicesSize(String searchString);
    WhDevice getDeviceById(Integer deviceId) throws AObjectNotExist;
    void addDevice(WhDevice whDevice) throws AObjectNotValid, AObjectAlreadyExists;
    void editDevice(WhDevice whDevice)  throws AObjectNotExist, AObjectNotValid;
    void deleteDevice(Integer id) throws AObjectNotExist;
    void sendDevice(Integer deviceId, Integer orgId, String room, String note, Date sendDate)
    Collection<WhDeviceHistory> getDeviceHistory(Integer deviceId) throws AObjectNotExist;
    void addDevices(ArrayList<WhDevice> devices) throws AObjectAlreadyExists, AObjectNotValid;
    Collection<WhDevice> getDevicesBySerial(String serial);
    Collection<WhDevice> getDevicesByType(Integer typeId);
    Collection<WhDevice> getDevicesByOrgId(Integer orgId);
    Collection<WhDevice> getLoanedDevicesByOrgId(Integer orgId);
}
```

Slika 21: Primer lokalnega vmesnika za EJB

5.1 Zaščita spletne aplikacije z JEE

Java EE omogoča zaščito spletnih aplikacij [8] s preverjanjem uporabnika na nivoju strežnika in določanju pravic na nivoju aplikacij, glede na vloge uporabnika.

Aplikacija je zaščitena z varnostnim (ang. Security) realmom, ki je že implementiran v aplikacijski strežnik GlassFish. Definicija varnostnega realma znotraj aplikacije je prikazana na slikah 22 in 23. Uporabili smo LDAP realm. Temu realmu podamo povezavo do LDAP strežnika, uporabniško ime in geslo za dostop do LDAP strežnika ter kasneje še avtentifikacijske podatke za uporabnika, ki se želi prijaviti v aplikacijo. GlassFish nam vrne skupino, kateri pripada uporabnik. V datoteki glassfish-web.xml naše aplikacije povemo, v katere vloge se ta skupina preslika. Ko si enkrat zagotovimo vloge v datoteki web.xml definiramo, kam lahko uporabnik z neko vlogo dostopa.

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name id="LDAPRealm">LdapRealm</realm-name>
    <form-login-config>
        <form-login-page>/login.xhtml</form-login-page>
        <form-error-page>/login.xhtml</form-error-page>
    </form-login-config>
</login-config>
```

Slika 22: Definiranje metode prijave v web.xml

```
<security-constraint>
    <display-name>USER</display-name>
    <web-resource-collection>
        <web-resource-name>USER</web-resource-name>
        <description/>
        <url-pattern>/*</url-pattern>
        <url-pattern>/templates/*</url-pattern>
        <url-pattern>/resources/*</url-pattern>
        <url-pattern>/resources/css/*</url-pattern>
        <url-pattern>/resources/img/*</url-pattern>
        <url-pattern>/resources/js/*</url-pattern>
        <url-pattern>/resources/img/icon/*</url-pattern>
        <url-pattern>/resources/img/logos/*</url-pattern>
        <url-pattern>/resources/js/cors/*</url-pattern>
        <url-pattern>/resources/js/vendor/*</url-pattern>
        <url-pattern>/javax.faces.resource/*</url-pattern>
        <http-method>POST</http-method>
        <http-method>PUT</http-method>
        <http-method>DELETE</http-method>
        <http-method>GET</http-method>
    </web-resource-collection>
    <auth-constraint>
        <description/>
        <role-name>user</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Slika 23: Definicija vloge in njenih pravic v web.xml

Nato pa še na EJB-jih z anotacijo @RolesAllowed nad celotnim EJB-jem ali samo nad določeno metodo povemo ali ima uporabnik s to vlogo pravico klicati metodo ali ne (slika 24).

```
@Stateless(name="WhDeviceServerEJB")
@RolesAllowed("user")
@Interceptors(WhInterceptor.class)
public class WhDeviceServerImpl implements WhDeviceServer {
```

Slika 24: Uporaba anotacije @RolesAllowed na EJB

Ker LDAP realm vrne le eno skupino in preverja samo ali je uporabnik veljaven ali ne, lahko imamo le eno vlogo. Naprednejši realmi ali naši lastni realni lahko vračajo več skupin, ki jih lahko preslikamo v več vlog. Ker se bo aplikacija izvajala na strežniku, ki ni viden zunanjim uporabnikom je ena vloga dovolj.

6 REST

REST [6] tehnologije smo morali uporabiti zaradi enega specifičnega problema. Spletna aplikacija mora omogočati prenos datotek na strežnik in iz njega. Ker so orodja za prenos datotek znotraj JSF ogrodja, ki smo ga uporabili za izdelavo uporabniškega vmesnika precej okorna, smo morali narediti JavaScript vmesnik, katerega edina možnost komunikacije s strežnikom je bila klic spletnih storitev preko AJAX-a.

REST je orodje za pripravo spletnih storitev. En EJB objekt smo anotirali z REST anotacijami, s katerimi smo definirali pot do storitve in deklarirali parametre funkcij, ki jih lahko kličemo. REST smo uporabili za nalaganje dokumentov, saj smo morali narediti svojo komponento za nalaganje dokumenta z enim potrditvenim klikom in za prenos dokumentov iz strežnika na uporabnikov računalnik. Dodati smo morali nekaj JavaScript-a, ki potem pokliče REST funkcijo za nalaganje. JAX-RS [7] je že del Jave EE, zato niso potrebne nobene dodatne konfiguracije za njegovo uporabo. Če Jave EE ne uporabljamo, je potrebno dodati krajšo nastavitev v web.xml datoteko aplikacije.

6.1 JAX-RS

JAX-RS lahko uporabimo na čisto navadnem objektu, vendar pa smo v našem primeru z JAX-RS anotacijami anotirali EJB (slika 25).

```
@Path(value = "/")
@Stateless
public class RestApi {
```

Slika 25: Anotiranje stateless EJB-ja za uporabo JAX-RS

Anotacija Path nam pove relativno pot do razreda. Metode definirane znotraj tega razreda potrebujejo nekaj več nastavitev. JAX-RS objekti omogočajo da delamo z navadnimi zahtevki in odgovori, kot so HttpServletRequest in HttpServletResponse, vendar pa je priporočljivo uporabiti njegove lastne zahteve in odgovore iz paketa javax.ws.rs.

```
@POST
@Produces(MediaType.APPLICATION_JSON)
@Path("/uploadDocument")
@Consumes(MediaType.MULTIPART_FORM_DATA)
public void uploadDocument(@Context HttpServletRequest request,
        @Context HttpServletResponse response,
        @QueryParam(value = "deviceId") String deviceId)
        throws FileUploadException, IOException,
                AObjectNotValid, AObjectNotExist {
```

Slika 26: Začetek metode v JAX-RS razredu

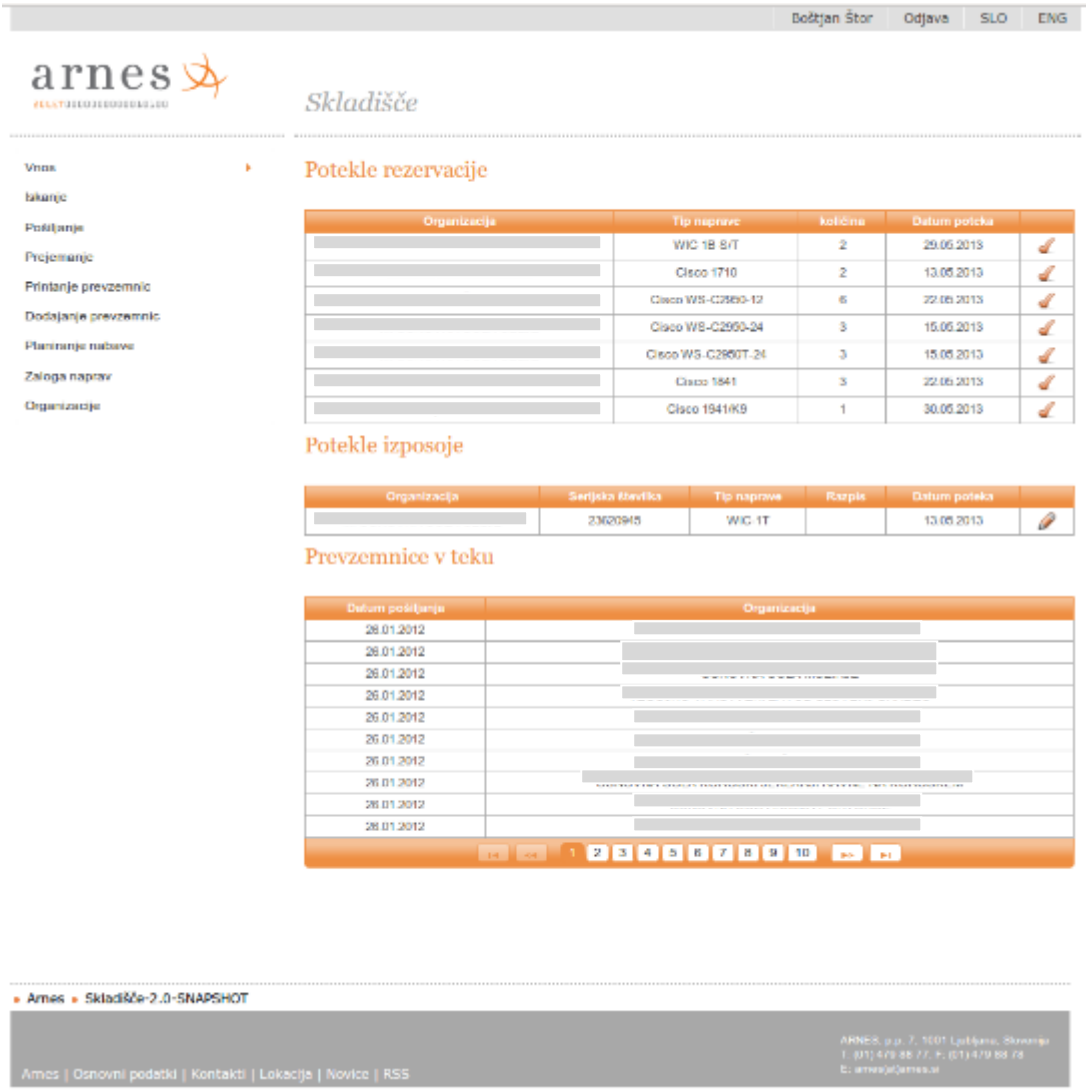
Pri pisanju metod moramo definirati tip metode dostopa(POST) definirane v glavi HTTP zahtevka, pot do metode, tip podatkov ki jih metoda sprejema in tip podatkov, ki jih vrača. @QueryParam anotacija nam priskrbi dodatne parametre HTTP zahteve. Deklaracija metode

na sliki 26 pripada metodi za nalaganje PDF dokumentov v podatkovno bazo. Ker je lahko dokument velik, ga zajema po delih. Za to skrbi jQuery-jev vtičnik za nalaganje datotek na strežnik, ki je napisan v JavaScript-u in deluje na strani odjemalca.

7 Uporabniški vmesnik

Za izdelavo uporabniškega vmesnika bomo uporabili Java Server Faces (JSF). Poleg osnovnih komponent, ki jih vsebuje JSF bom uporabili še komponente ogrodja ICEfaces. Uporabniški vmesnik sestavljajo XHTML datoteke, znotraj katerih so definirane povezave do dodatnih značk, ki so potrebne za JSF. Poleg tega, da se znebimo podvajanja programske kode na vsaki strani uporabimo predlogo, ki jo vključimo kot inicializacijski parameter strani in tako oblikujemo le vsebino, ne pa tudi glave, noge in stranskega menija. Za pravila prikaza strani skrbijo CSS datoteke, v katerih so poleg postavitve definirane barve, pisave, ozadja, oblike gumbov in podobno. Za JavaScript bi moralo sicer poskrbeti ogrodje, vendar je bil problem z nalaganjem datotek na strežnik, za katerega smo morali narediti nalagalnik s pomočjo jQuery knjižnice za JavaScript. Na slikah 27 in 28 je prikazan izgled uporabniškega vmesnika na aplikaciji Skladišče.

Slika 27: Spletna aplikacija - iskanje naprav



Slika 28: Spletna aplikacija - vstopna stran

7.1 Java Server Faces

Java Server Faces je eden izmed orodij za izdelavo uporabniškega vmesnika spletne aplikacije. V osnovi je zelo podoben Java Server Pages, le da je vsa logika v Java razredih, upravljalnih zrnih ali CDI zrnih. Funkcionalnosti so podobne, glavna razlika je, da upravljalno zrno spada pod JSF in CDI zrno pod EJB. Nepisano pravilo velja, če je le možnost uporabimo CDI zrno, saj ga obravnava aplikacijski strežnik kot EJB, upravljalno zrno pa FacesServlet, ki se nahaja v aplikaciji. upravljalno zrno uporabimo v primeru, da potrebujemo večjo fleksibilnost in da je zaradi tega aplikacija hitrejša, ali da je rešitev veliko preglednejša in enostavnejša. Za razliko od nekaterih drugih ogrodij, ki recimo delovanje protokola HTTP popolnoma skrijejo programerju in celotno pošiljanje podatkov poteka izključno preko AJAX-a, pri JSF-jih ni tako. Če uporabljamo samo JSF, so tukaj še zmeraj obrazci, ki pošiljajo zahteve preko POST ali GET metode. Ogrodje do neke mere to zamaskira in samo poskrbi za del delnih pošiljanj na strežnik preko AJAX-a, vendar je

delovanje HTTP protokola še zmeraj jasno vidno. Velika prednost upravljalnih zrn in CDI zrn, je določanje časa veljavnosti s tako imenovanim ciklom. Tako, da ne obstajajo le v času zahteve, ampak ostanejo dalje, po možnosti je en objekt lahko veljaven celotno izvajanje aplikacije. Vendar je treba biti previden, saj so ta upravljalna zrna požrešnejša od navadnega servleta. Njihova prednost je v tem, da jih ni potrebno vsakič inicializirati, ampak nam jih aplikacijski strežnik da že narejene in zato ne tratimo časa z vzpostavljanjem povezave do baze in kreiranjem objektov. Iz končne XHTML strani dostopamo do pomožnega objekta, tako da dostopamo direktno do spremenljivke, ki mora imeti definirane funkcije za branje in nastavljanje vrednosti. Poleg tega določene JSF značke omogočajo, da na njih pripnemo poslušalce (listener ali actionListener), ki pokličejo metodo znotraj upravljalnega zrna in ko le ta spremeni vrednosti spremenljivk, ki se nahajajo v kontekstu (FacesContext), se vidijo spremembe na spletni strani. To je začetek specifike ogrodja, ki ga bomo uporabljali za lažje delo in lepši ter bolj uporaben vmesnik.

7.2 Icefaces

Icefaces je eno izmed ogrodij za izdelavo spletnih aplikacij na podlagi JSF. Pri odločanju, katero ogrodje bomo uporabili, je bilo potrebno dobro pregledati še ostala ogrodja ter pretehtati njihove prednosti in slabosti. Na izbiro imamo nekako tri večje ponudnike, RichFaces, ICEfaces [3] in PrimeFaces. Poleg teh ponudnikov obstajata še MyFaces in OmniFaces. Osredotočili smo se na RichFaces, ICEfaces in PrimeFaces, ker so v času izbora ogrodja bili največji. Po izvedbi krajše primerjalne analize, smo ugotovili naslednje, da ima RichFaces najbolj dodelan logični del in algoritme, ki skrbijo za hitro in stabilno delovanje, vendar ima najmanj komponent. Pri PrimeFaces je situacija ravno nasprotna, ogromno komponent ampak zelo lahek sistem v ozadju. ICEfaces je zlata sredina med količino komponent in dovršenostjo sistema v ozadju. Te ugotovitve smo povzeli po članku [11] in se na podlagi le teh odločili za ICEfaces 3.2.0. Tekom razvoja smo ugotovili, da to ni bila najboljša izbira, saj ima ICEfaces veliko hroščev v kodi, za katere popravki niso planirani za bližnjo prihodnost. Zaradi te pomanjkljivosti in manjše pomembnosti sistema v ozadju, se programerji vse bolj množično odločajo za PrimeFaces.

Ogrodje ICEfaces je razdeljeno na dva tipa komponent uporabniškega vmesnika. Obstajajo ICE in ACE komponente. ICE komponente so nekoliko bolj direktno vezane na običajne HTML komponente in imajo enak ali zelo podoben set atributov. ACE komponente so bolj dodelane in omogočajo več stvari.

7.2.1 Komponente

V paketu ICEfaces [2] sta dva kompleta komponent, ICE in ACE. Nekatere se direktno prevedejo v html komponente z nekaj dodanega javascripta in AJAX klicev. V nadaljevanju je opisanih nekaj kompleksnih komponent, ki precej olajšajo pisanje uporabniškega vmesnika in povsem odpravijo pisanje kode v JavaScript-u.

7.2.1.1 ace:dataTable

Ace:dataTable je tabela za prikaz podatkov (slika 30). Z atributi komponente določimo, katere podatke želimo prikazati, kako naj bo urejeno ostranjevanje, ali je mogoče vrstice urejati, ali želimo izbirati posamezne vrstice tabele za kasnejšo obravnavo. Atribut value sprejme objekt tipa java.util.List, v katerem so objekti, ki jih želimo prikazati v vrsticah, listVar predstavlja en objekt iz prejšnega seznama. Za ostranjevanje skrbijo atributi paginator (true ali false vrednost, ali sploh želimo uporabiti ostranjevanje), pageSize (velikost strani) in paginatorPosition (pozicija ostranjevanja). Atributi selectable in selectionListener obravnavajo izbiranje vrstic (slika 29). Zanimiv je še atribut lazyLoading, ki omogoča, da se za vsako stran posebej dobi podatke, tako da ne generiramo prevelike tabele, če je veliko rezultatov. Nekaj dodatnih uporabnih funkcionalnosti je še urejanje vrstic, iskanje po stolpcih in sortiranje vrstic.

```
<ace:dataTable id="deviceTable"
               value="#{deviceSearchManagedBean.lazyDevices}"
               var="device"
               lazy="true"
               paginator="true"
               paginatorPosition="bottom"
               rows="10">
```

Slika 29: Primer programske kode za ace:datatable

organizacija	tip naprave	količina	cena	urejanje
	Cisco WS-C2950-24	2	$ 1.000,00	
	Cisco 1721	3	$ 250,00	

Slika 30: Izgled ace:datatable

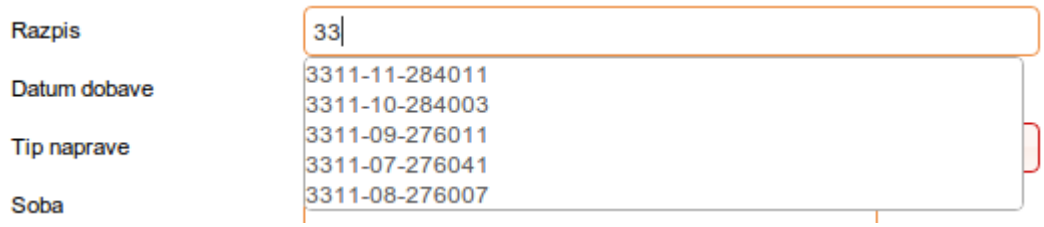
7.2.1.2 ace:autoCompleteEntry

Komponenta ace:autoCompleteEntry (slika 32) pri vpisovanju teksta v vnosno polje ponuja rezultate iskanja, ki jih morda želimo uporabiti, da nam ni treba vpisovati celotne vsebine. Ta komponenta je uporabna v situaciji, če želimo izbrati nek vpis v tabelo v bazi, vendar je njena slabost to, da dobimo iz nje samo tekst, ki smo ga vpisali in ne tudi indeksa vrstice v bazi. Zato potrebujemo dodatno preverjanje ali je res izbran vnos ali je v vnosno polje vpisan nek naključen tekst. Pri tej komponenti smo ugotovili dve veliki napaki, katere rešitev ni načrtovana v bližnji prihodnosti. Če uporabimo atribut validator, v katerem je definirana funkcija, ki skrbi za preverjanje, se seznam možnih izbir ne prikaže (slika 31), tako je edino možno preverjanje šele pri pošiljanju podatkov na strežnik (submit). Druga napaka je v uporabi ace:autoCompleteEntry znotraj neke druge ACE komponente, kot je ace:dialog ali ace:tabSet. Seznam možnih izbir prikaže približno pol strani v desno in pol strani navzdol od samega vnosnega polja. To težavo smo rešili tako, da v namesto ACE komponentah ace:autoCompleteEntry gnezdimo v ICE komponentah. Glede na to, da je alternativa uporaba

navadnih HTML komponent in pisanje JavaScripta ter AJAX klicev, je še zmeraj smiselno uporabiti to komponento.

```
<ace:autoCompleteEntry id="jnmvAutoComplite"
                       value="#{deviceAddManagedBean.selectedJnmv}"
                       rows="10" width="277"
                       filterMatchMode="startsWith"
                       listVar="jnmv"
                       listValue="#{deviceAddManagedBean.jnmvs}"
                       filterBy="#{jnmv.jnmvNr}"
                       valueChangeListener="#{deviceAddManagedBean.jnmvChangeListener}">
    <f:facet name="row">
        <h:outputText value="#{jnmv.jnmvNr}" />
    </f:facet>
</ace:autoCompleteEntry>
```

Slika 31: Programska koda za ace:autoCompleteEntry



Slika 32: Izgled ace:autoCompleteEntry

7.2.1.3 ace:dateTimeEntry

Ace:dateTimeEntry je polje za vnos datuma (slika 34). Deluje tako, da datum izberemo iz koledarja. Ta koledar je lahko fiksno prikazan na strani ali pa se odpre v novem pogovornem oknu tik pod vnosnim poljem. Sprejme atribut kot je pattern, v kateremu povemo, v kakšni obliki želimo videti izpis izbranega datuma (slika 33).

```
<ace:dateTimeEntry id="calIcon" showOn="both"
                   value="#{deviceAddManagedBean.supplyDate}"
                   pattern="dd.MM.yyyy"
                   renderAsPopup="true"
                   popupIcon="/resources/img/icon/clock.png"
                   popupIconOnly="true"
                   rendered="true"
                   size="50"
                   required="true" />
```

Slika 33: Programska koda za ace:dateTimeEntry

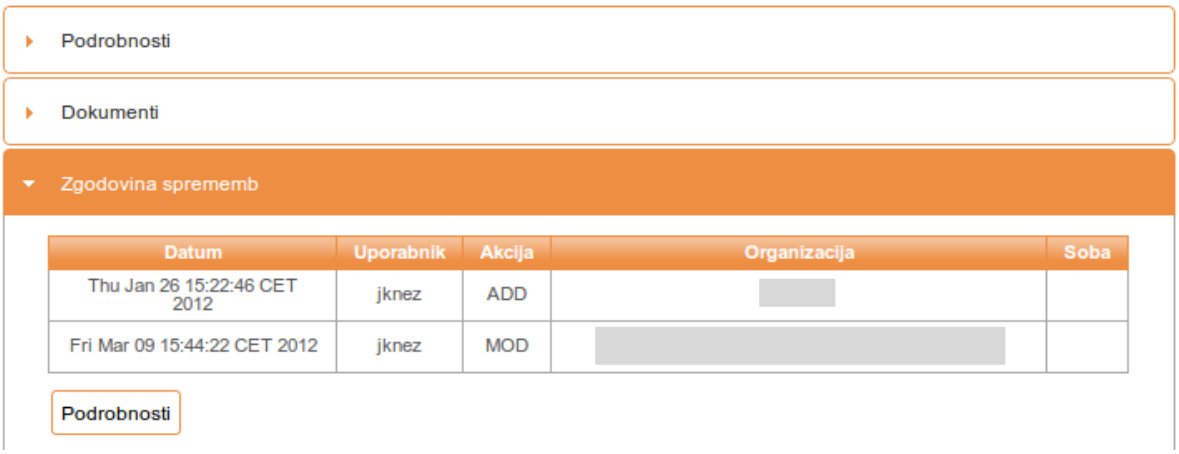
Slika 34: Izgled ace:dateTimeEntry

7.2.1.4 ace:accordion

Ace:accordion je komponenta, ki ima več pogledov, ki so postavljeni eden nad drugega, med katerimi je samo en odprt ostali pa zaprti (slika 36). Ko izberemo drug zavihek, se trenutni pogled skrije in izbrani odpre. Poglede znotraj ace:accordion definiramo z ace:accordionPane (slika 35). Komponenta je primerna za strani na katerih potrebujemo veliko podatkov, ampak ni nujno, da so vsi podatki vidni naenkrat.

```
<ace:accordion>
    <ace:accordionPane title="#{msg['device.details.title']}">
        <h:panelGrid columns="2" width="400px" style="height: 450px;">
            <h:outputText value="#{msg['device.details.id']}:" /><h:out
            <h:outputText value="#{msg['device.add.serial']}:" /><h:out
            <h:outputText value="#{msg['device.add.inventorynr']}:" /><
```

Slika 35: Programska koda za ace:accordion



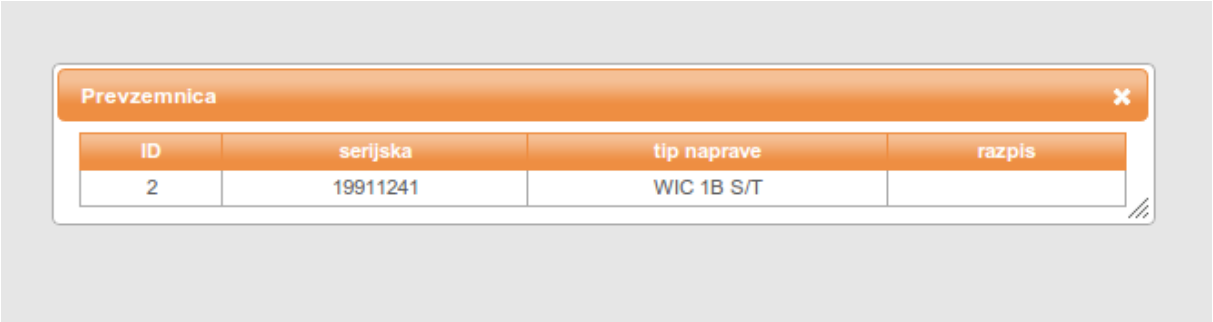
Slika 36: Izgled ace:accordion

7.2.1.5 ace:dialog

Ace:Dialog komponenta nam odpre novo pogovorno okno (slika 38). To okno je lahko modalno (za čas odprtja okna onemogoči preostanek strani in ga obarva v prosojno sivo). Na njem je možnih več efektov odprtja in zaprtja okna (slika 37). V njem prikazujemo razne podrobnosti, za katere na glavni strani ni prostora, ali obrazce za dodajanje ali urejanje podatkov.

```
<ace:dialog id="addDialog"
            header="#{msg['jnmv.add.title']}"
            modal="true"
            widgetVar="addModal"
            draggable="true"
            showEffect="fade"
            hideEffect="explode"
            width="400"
            closable="true">
```

Slika 37: Programska koda ace:dialog



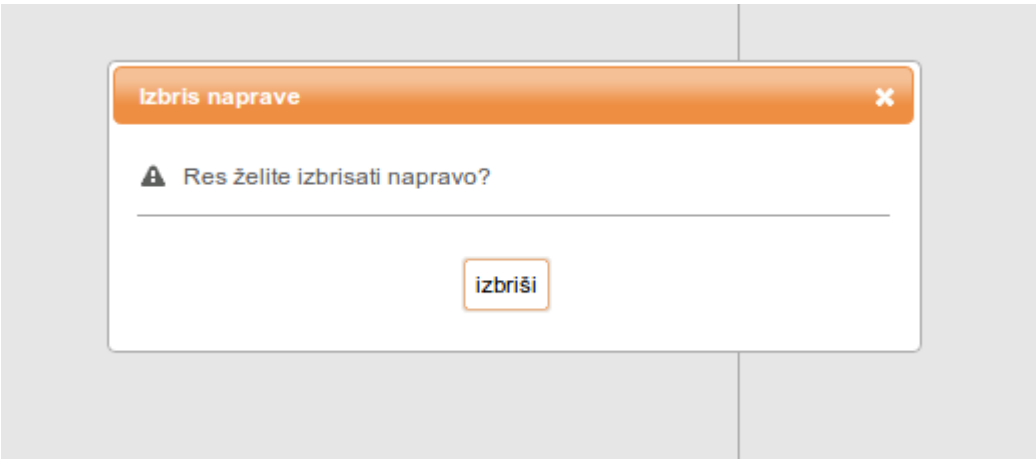
Slika 38: Izgled ace:dialog

7.2.1.6 ace:confirmationDialog

Ace:confirmationDialog je različica ace:dialog-a, ki je namenjen le za potrjevanje (slika 40). V primeru brisanja podatkov navadno želimo ponuditi uporabniku možnost, da si premisli v primeru, da je kliknil napačno. Sprejema atribute za nastavitev naslova in sporočila (slika 39). V telesu komponente je prostor za gumb za potrditev in za preklic akcije.

```
<ace:confirmationDialog id="deleteDialog"
                        header="#{msg['devicetype.delete.title']}"
                        message="#{msg['devicetype.delete.message']}"
                        widgetVar="deleteModal"
                        modal="true"
                        draggable="true"
                        showEffect="fade"
                        hideEffect="explode"
                        width="400"
                        closable="true"
                        closeOnEscape="true">
    <h:panelGrid width="100%" style="text-align: center;">
        <h:commandButton class="button" value="#{msg['toolbarbutton.hint.delete']}"
    </h:panelGrid>
</ace:confirmationDialog>
```

Slika 39: Programska koda ace:confirmationDialog



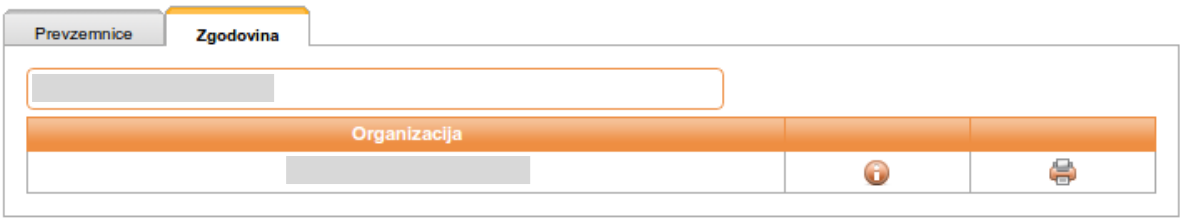
Slika 40: Izgled ace:confirmationDialog

7.2.1.7 ice:panelTabSet

Ice:tabSet je komponenta, ki je po funkcionalnosti zelo podobna ace:accordion, le da ima zavihke postavljene vodoravno na vrhu vsebine komponente (slika 42). Komponento smo uporabili zato, ker smo želeli v njeni vsebini uporabiti komponento ace:autoCompleteEntry, ki pa znotraj ace komponent ne deluje pravilno. Primer uporabe komponente ice:panelTabSet je prikazan na sliki 41.

```
<ice:panelTabSet>
    <ice:panelTab label="#{msg['device.recievepaper.title']}">
        <h:panelGrid width="100%">
```

Slika 41: Programska koda ice:panelTabSet



Slika 42: Izgled ace:panelTabSet

8 Rezultati in diskusija

Rezultat diplomskega dela je spletna aplikacija razvita s tehnologijami Jave EE. Iz specifikacije JEE so bili uporabljeni EJB-ji, JPA, REST anotacije za EJB objekte, CDI zrna in JSF. DAO modul aplikacije je testiran s sistemom jUnit. Za končni prikaz uporabniku pa so bili uporabljeni JSF-ji. Za JSF smo uporabili ogrodje ICEfaces, tako da smo lahko uporabljali lepše, bolj kompleksne komponente brez pretiranega pisanja Javascript kode in AJAX klicev. Delo na aplikaciji pa so že od vsega začetka olajševali načrtovalski vzorci brez katerih bi naleteli tudi na kakšen nerešljiv problem. Od vseh načrtovalskih vzorcev so k preglednosti kode in boljšemu delovanju aplikacije najbolj pripomogli metoda tovarne, lena inicializacija, bazen objektov, edinec, prednji krmilnik, modul in specifikacija. Z metodo tovarne smo se znebili statičnih metod v DAO razredih. Sedaj razred tovarna vrne enega ali več DAO razredov, odvisno od tipa metode. Leno inicializacijo smo uporabili zato, da pri prikazu večjega števila podatkov ne prenašamo vseh podatkov. Prikažemo jih manj in če potrebujem več podatkov, naredimo še eno poizvedbo. Bazen objektov si naredi aplikacijski strežnik sam. V primeru, da potrebujemo nek objekt, nam strežnik pošlje objekt, ki je že narejen in čaka na uporabo. Edinca smo uporabili za branje nastavitvenih datotek. Ker so dostopi do njih samo bralni, lahko imamo en objekt, ki vsakemu uporabniku ponudi isto vsebino. Prednji krmilnik je del MVC vzorca. Prejema podatke iz podatkovnega modela in pripravi vsebino za prikaz uporabniku. Predstavlja enega izmed modulov, ki smo ga naredili zaradi vzorca modul. Če imamo aplikacijo razdeljeno na module, imamo manj dela če se kakšen informacijski vir zamenja. V tem primeru zamenjamo le modul. Specifikacija prestavlja pravila programiranja po katerih je potrebno pisati aplikacije. V primeru daljše odsotnosti enega razvijalca v skupini, lahko nekdo drug brez večjih težav prevzame njegovo delo.

Tekom razvoja aplikacije smo naleteli na nekaj problemov, ki so bili rezultat ne najboljše izbire ogrodij. Glavni krivec težav je bil nedvomno ogrodje za JSF, ICEfaces, za katerega smo šele po nadaljnjih raziskavah ugotovil, da ima veliko količino programskih hroščev, katerih odprava pa ni planirana za bližnjo prihodnost. Polega tega so tekom razvoja aplikacije prenehale delovati določene funkcionalnosti zaradi spremembe verzije ICEfaces. Ob pogledu nazaj bi bila boljša odločitev uporabiti ogrodje PrimeFaces. JSF smo uporabili zato, ker JSF-ji delujejo v ServletContext-u, kar pomeni da lahko do EJB objektov in ostalih orodij Jave EE dostopamo preko anotacij in nam za njihovo Inicializacijo skrbi aplikacijski strežnik. Na začetku razvoja aplikacije smo uporabljali ogrodje ZK, katerega objekti se ne izvajajo v ServletContext-u in je bilo zaredi tega potrebno do EJB-jev dostopati preko JNDI poizvedovanja.

Poleg tega je še problem izbire kontrolerjev za JSF. Lahko uporabimo CDI objekte, ki so del Jave EE ali pa upravljalne objekte, za katere skrbi JSF servlet. Splošno prepričanje je, da so CDI zrna boljša vendar pa imajo JSF upravljalna zrna dosti bolj fleksibilne življenjske cikle. Iz tega razloga so kontrolerji, katerih življenjski cikli zavzemajo zahtevo in se potem pozabijo, tipa CDI in vsi ostali, ki morajo držati svoje stanje več časa upravljalna zrna.

9 Sklep

Cilj diplomske naloge je bil preučiti načrtovalske vzorce in prikazati njihovo uporabo na primeru razvoja spletne aplikacije. Za izdelavo spletne aplikacije smo uporabili tehnologije Jave EE. Po teoretičnem opisu načrtovalskih vzorcev smo jih poskusili čim več uporabiti v spletni aplikaciji. Nekatere načrtovalske vzorce nam je narekovala že sama tehnologija. Z uporabo EJB-jev dobimo več različnih modulov aplikacije, ki jih lahko sami, po potrebi še dodatno razbijemo. Uporaba Java Server Faces pa zahteva uporabo kontrolerjev za spletne strani. Uporabili smo še nekaj dodatnih načrtovalskih vzorcev zaradi katerih aplikacija deluje hitreje, je bolj pregledna in nam lajša vzdrževanje. Uporabili smo nekaj več kot polovico vseh opisanih načrtovalskih vzorcev. Ker so nekateri vzorci precej namenski, ni smiselno ali mogoče vseh implementirati.

Glede tehnologij, smo načrtovali uporabo EJB-jev in Java Server Faces. Oba sta dela Jave EE. EJB-je smo uporabili kot vmesnik med DAO projektom in kontrolerji za JSF strani. Za takšno strukturo smo se odločili, ker želimo v nadaljevanju razdeliti spletno aplikacijo na dva dela, na strežnik in odjemalca. Strežnik bo vseboval DAO projekt in EJB modul, ki bo dostopen preko oddaljenega vmesnika (ang. Remote interface), odjemalec pa JSF strani in njihove kontrolerje. Za to rešitev smo se odločili zato, da lahko strežnik postavimo v privatno omrežje in da v primeru vdora v odjemalca ne razkrijemo pomembnejših virov, kot je recimo podatkovna baza. Java Server Faces smo uporabili za izdelavo uporabniškega vmesnika. Ker je JSF del Jave EE podpira dostop do virov aplikacijskega strežnika in nam za dostop do le-teh ni treba izvajati JNDI poizvedovanj. Za dostop do podatkovne baze smo uporabili Java Persistence API. Zaradi problemov z obstoječim nalagalnikom datotek znotraj ICEfaces smo morali uporabiti JavaScript nalagalnik, ki je za dostop do strežnika potreboval spletne storitve, ki smo jih implementirali z JAX-RS (REST).

Trenutna verzija spletne aplikacije Skladišče je 2.0. Iz 1.0 na 2.0 se je zgodil prehod na Javo EE. V nadaljnjih verzijah sledi ločitev aplikacije na strežnik in odjemalca. Zaradi prehitre izbire ogrodja za JSF, bi rad le-tega kmalu zamenjal. Verjetno se bomo odločili za zamenjavo ogrodja ICEfaces za PrimeFaces. Zaradi vedno hitrejšega širjenja programiranja v oblaku bi bilo pametno razmisliti o premiku aplikacije na enega od IaaS sistemov. Dodatne spremembe v prihodnosti bi še bolj pripomogle k skalabilnosti spletne aplikacije.

10 Literatura

[1] L. Rubinger, W. J. Burke, Enterprise JavaBeans 3.1, Sixth edition, Sebastopol: O'Reilly Media, Inc., 2010

[2] (2013) ICEfaces Showcase. Dostopno na: http://icefaces-showcase.icesoft.org/showcase.jsf ,

[3] (2013) ICEfaces Wiki. Dostopno na: http://www.icesoft.org/wiki/display/ICE/ICEfaces+Wiki

[4] E. Freeman, E. Freeman, K. Sierra, B Bates, Head First Design Patterns, Sebastopol: O'Reilly Media, Inc., 2004

[5] (2013) Software design pattern. Dostopno na: http://en.wikipedia.org/wiki/Software_design_pattern

[6] (2013) RESTful Web Services. Dostopno na: http://www.oracle.com/technetwork/articles/javase/index-137171.html

[7] (2013) Java API for RESTful Services (JAX-RS). Dostopno na: https://jax-rs-spec.java.net/

[8] (2013) The Java EE 6 Tutorial. Dostopno na: http://docs.oracle.com/javaee/6/tutorial/doc/

[9] (2013) MySQL Community Edition. Dostopno na: http://www.mysql.com/products/community/

[10] (2013) Hibernate Developer Guide. Dostopno na: http://docs.jboss.org/hibernate/orm/4.2/devguide/en-US/html/

[11] (2012) PrimeFaces vs RichFaces vs IceFaces. Dostopno na: http://www.mastertheboss.com/richfaces/primefaces-vs-richfaces-vs-icefaces