Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jernej Jerin

Integracija informacijskega sistema z napravami na primeru meteorološke postaje

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: dr. prof. Matjaž Branko Jurič

Ljubljana, 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

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: 00026/2012
Datum: 11.04.2012

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

Kandidat: **JERNEJ JERIN**

Naslov: **INTEGRACIJA INFORMACIJSKEGA SISTEMA Z NAPRAVAMI NA PRIMERU METEOROLOŠKE POSTAJE**

INFORMATION SYSTEM INTEGRATION WITH DEVICES ON CASE OF METEOROLOGICAL STATION

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Proučite in primerjajte internet storitev in internet naprav oz. stvari ter opišite ključne tehnologije. Analizirajte tehnologije Java EE platforme, primerne za integracijo sistemov in naprav. Zasnujte informacijski sistem za integracijo z meteorološko postajo po posameznih nivojih in prikažite primere uporabe.

Mentor:

prof. dr. Matjaž B. Jurič

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jernej Jerin, z vpisno številko **63080038**, sem avtor diplomskega dela z naslovom:

Integracija informacijskega sistema z napravami na primeru meteorološke postaje

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom dr. prof. Matjaža Branka Juriča,
- 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 26. avgusta 2012 Podpis avtorja:

Kazalo

Slike

Tabele

Seznam uporabljenih kratic

Kratica	Angleški pomen	Slovenski pomen
API	Application Programming Interface	vmesnik uporabniškega programa
AS	Application Server	aplikacijski strežnik
CDI	Contexts and Dependency Injection	vstavljanje v okviru odvisnosti in konteksta
CRUD	Create, Read, Update, Delete	ustvari, preberi, posodobi, zbriši
DNS	Domain Name System	sistem za upravljanje z imeni domen
DOM	Document Object Model	objektni model dokumenta
EAR	Enterprise Application Project	poslovni aplikacijski projekt
EE	Enterprise Edition	poslovna izdaja
EJB	Enterpise Java beans	poslovna Java zrna
EL	Expression language	izrazni jezik
HTML	HyperText Markup Language	jezik za označevanje nadbesedila
HTTP	Hypertext Transfer Protocol	protokol za prenos HTML strani
IOT	Internet of Things	internet stvari
IT	Information technology	informacijska tehnologija

Kratica	**Angleški pomen**	**Slovenski pomen**
JNDI	Java Naming and Directory Interface	Java imenski in imeniški vmesnik
JSF	Java Server Faces	Java strežniški obrazi
JSON	JavaScript Object Notation	JavaScript objektna notacija
JVM	Java virtual machine	Java virtualni stroj
REST	REpresentational State Transfer	arhitekturni stil gradnje porazdeljenih sistemov
SOA	Service-oriented architecture	storitveno orientirana arhitektura
SOAP	Simple Object Access Protocol	protokol za prenašanje strukturiranih informacij
SQL	Structured Query Language	strukturirani povpraševalni jezik za delo s podatkovnimi bazami
SUPB	database management system	Sistem za upravljanje podatkovne baze
THW	Temperature, Humidity, Wind	Temperatura, vlaga, veter
TXT	Text file	Tekstovna datoteka
UI	User interface	Uporabniški vmesnik
URI	Uniform resource identifier	Enolična identifikacija vira
VPS	Virtual private server	virtualni privatni strežnik
XML	Extensible Markup Language	razširljiv označevalni jezik

Povzetek

Načrtovanje, izdelava in integracija informacijskega sistema je proces, ki zahteva dobro pripravljen načrt realizacije. Potrebno je namreč identificirati dele informacijskega sistema, ki so kritični za pravilno delovanje. Prav tako morajo biti odporni na vnose napačnih, oziroma nedefiniranih podatkov. S temi načeli v mislih je bil cilj diplomske naloge izdelati informacijski sistem za podporo delovanja meteorološke postaje in na podlagi integriranega informacijskega sistema teoretično opredeliti internet storitev in internet stvari. Opredelili smo naslednje dele informacijskega sistema, ki so imeli prioriteto pri realizaciji. Del sistema, ki omogoča zapisovanje izmerjenih meteoroloških podatkov v podatkovno bazo in njihovo obdelavo. Ter del sistema, ki realizira spletne storitve in prikaz trenutnih/preteklih podatkov na spletni strani na uporabniku prijazen način. Za vse naštete potrebe smo razvili informacijski sistem v tehnologiji Java EE 6. Informacijski sistem je sestavljen iz štirih nivojev. Prvi nivo je nivo odjemalca, ki uporablja tehnologije spletnega brskalnika za dostop do storitev informacijskega sistema. Drugi nivo je spletni nivo, ki uporablja tehnologijo JSF za generiranje spletnih strani. Tretji nivo je poslovni nivo, ki uporablja tehnologijo EJB za poslovno logiko in tehnologijo JPA za objektno relacijsko preslikavo. Četrti oziroma zadnji nivo je nivo podatkovne baze, ki uporablja tehnologijo MySQL za trajno shranjevanje podatkov.

Ključne besede: informacijski sistem, internet storitev, internet stvari, Java Enterprise Edition, meteorološka postaja, integracija z napravami

Abstract

Planning, developing and integration of information system is a process which takes a good plan for realization. It is necessary to identify parts of information system, which are criticall for the proper functioning. They must also be resistent to inputs of false or undefined data. With these principles in mind the aim of the thesis was to create information system to support the operation of meteorological station and on the basis of an integrated IT system theoretically define Internet of Services and Internet of Things. We have identified the following parts of the information system which had priority in the realization. Part of the system, which allows the recording of measured meteorological data in the database and processing of it. And part of a system that realizes the web service and display current / historical data in the web application in a user-friendly way. For all these needs, we have developed an information system in Java EE 6. Information system consists of four tiers. The first tier is client tier that uses web browser technology to access the information system services. The second tier is web tier that uses JSF technology for generating web pages. The third tier is business tier that uses EJB for the business logic and JPA technology for object-relational mapping. The fourth and final tier is the database tier that uses MySQL to store data permanently.

Keywords: Information system, Internet of Services, Internet of Things, Java Enterprise Edition, Meteorological station, Integration with Devices

Poglavje 1

Uvod

Hiter razvoj računalništva je imel in še vedno ima velik vpliv na razvoj ostalih panog. Ena izmed panog, na katero je razvoj računalništva imel še posebej velik vpliv je meteorologija. Zaradi ogromnih količin podatkov, ki jih meteorologi pridobivajo iz stacionarnih meteoroloških postaj, meteoroloških postaj na ladjah, bojah, letalih in iz satelitov v orbiti je potreba po hitri obdelavi zbranih podatkov velika. Tu nastane potreba po informacijskem sistemu, ki omogoča realizacijo sistematičnega zbiranja, obdelave, interpretacije in prikaza zbranih podatkov.

Tako smo v okviru diplomske naloge razvili informacijski sistem, ki rešuje zgoraj naštete probleme. Informacijski sistem smo razvili na platformi Java EE 6, ki omogoča razvoj kompleksnih poslovno informacijskih sistemov. Izbira platforme je zelo primerna, saj omogoča visoko skalabilnost, zanesljivost in odzivnost sistema. Poleg tega nam Java ponuja veliko knjižnic, ki olajšajo razvoj sistema. Naši glavni prispevki so:

- Aplikacija za branje podatkov iz datoteke,
- podatkovni model in logika znotraj podatkovnega modela,
- poslovna logika za obdelavo in zapisovanje podatkov na aplikacijskem strežniku,
- uporabniški vmesnik za prikaz trenutnih/preteklih podatkov,

- sistem, ki nudi spletne storitve.

Del informacijskega sistema, ki zagotavlja spletne storitve smo povezali z teoretičnim poglavjem internet storitev. V tem poglavju smo opredelili kaj so to storitve in kako so povezane z praktično implementacijo. Del sistema, ki zagotavlja povezavo programske opreme z napravami pa smo povezali z teoretičnim poglavjem internet stvari. Tu smo prikazali, na kakšen način so senzorji meteorološke postaje povezani z programsko opremo.

Poglavje 2

Internet storitev in internet stvari

V okviru diplomske naloge smo opisali dva pomembna vidika v svetu interneta in sicer internet storitev in internet stvari. Pri opisu smo se poskusili navezovati na praktično realizacijo informacijskega sistema in tako predstaviti storitve in stvari ne samo skozi teoretični pogled ampak tudi praktični.

Razliko med storitvami in stvari lahko opišemo na naslednji način. Storitve so v nekaterih primerih programska abstrakcija stvari, kjer so stvari fizični objekti oziroma senzorji povezani v Internet. Torej stvari nam dejansko z zbiranjem podatkov iz okolice omogočajo storitve, le da se pri uporabi storitev tega mnogokrat ne zavedamo. Obstajajo pa tudi takšne storitve, ki niso vezane na stvari. Uporaba takšnih storitev ima neposredne učinke samo v virtualnem svetu.

2.1 Internet storitev

2.1.1 Uvod

Internet storitev (Internet of Services - IoS) je spletno osnovani storitveni ekosistem podprt z informacijsko tehnologijo, ki omogoča storitveno usmerjeno delovanje[15]. Kot vsaka druga vrsta storitev potrebuje tudi Internet storitev

za delovanje tako tehnično kot tudi poslovno infrastrukturo. Tehnični del Internetnih storitev opisuje infrastrukturo, ki uporablja Internet kot medij za dostop do storitev. Še posebej je tu potrebno izpostaviti tržišče storitev (service marketplace), kjer lahko ponudniki in porabniki trgujejo z storitvami. Na takšna tržišča se poleg Interneta gleda kot na glavno vizijo Internetnih storitev.

Čeprav ima Internet kot medij veliko težo in se velikokrat na Internetne storitve gleda kot na klasične storitve, ki so implementirane z drugačno tehnologijo lahko na Internet storitev gledamo kot na poslovni model, ki lahko popolnoma spremeni naše dojemanje uporabljanja in odkrivanja storitev. Torej na Internet storitev lahko gledamo iz dveh različnih perspektiv. Iz perspektive informacijske tehnologije in iz poslovne perspektive. Perspektiva informacijske tehnologije določa globalni opis standardov, orodij, aplikacij in arhitekture, ki so na voljo poslovni perspektivi. Medtem ko poslovna perspektiva določa različne tipe storitev, ki so lahko vključene v kompleksnejše poslovne procese oziroma njihova implementacija omogoča operiranje z drugimi storitvami.

2.1.2 Kaj so storitve?

Storitve predstavljajo avtonomno programsko komponento, ki jo lahko enolično definiramo preko URI naslova in do katere dostopamo preko standardnih internetnih protokolov, kot so XML, SOAP ali HTTP[2]. Obstajajo pa tudi druge vrste storitev, ki ne temeljijo na internetnih protokolih (npr. EJB storitve). Poznamo tri različne izraze, ki predstavljajo podobne koncepte, toda iz različnih domen in sicer poslovne storitve, e-storitve in spletne storitve. V naslednjih odstavkih smo opredelili prej naštete izraze in domene, katerim pripadajo.

Poslovne storitve

V poslovni in ekonomski domeni se storitve obravnava kot neopredmetene aktivnosti. Takšne aktivnosti oziroma storitve, ki jih odjemalcu oziroma po-

trošniku ponuja trg naj bi odjemalcu prinašale dodano vrednost. Odkrivanje in izvrševanje poslovnih storitev je lahko ročno ali avtomatsko. Zaradi neopredmetene narave storitev je njihov opis glede na lastnosti nemogoč. Tako je potrebno storitve definirati na način, ki opisuje posredne učinke uporabe takšnih storitev na odjemalca (potrošnika). V naši implementaciji bi lahko definirali učinke uporabe storitev kot informacijo, ki daje uporabniku prednost v smislu informiranosti pred nevarnimi trenutnimi in prihodnjim vremenskim razmeram. Iz tega sledi, da je opisovanje posrednih učinkov na odjemalca ena izmed najpomembnejših dejavnosti v poslovni domeni.

E-storitve

E-storitve je množica programskih storitev, ki se nahajajo na omrežju in so dostopne preko standardiziranih protokolov. Funkcionalnost takšnih storitev lahko odkrijemo avtomatsko in integriramo v aplikacijo oziroma v množico, ki tvori kompleksnejšo storitev. Zaradi njihove uporabe v poslovne namene se e-storitve obravnava kot podmnožice poslovnih storitev. Vsako storitev katere funkcionalnost je lahko dostopna preko podatkovnega omrežja lahko transformiramo v e-storitev. E-storitve so tudi neodvisne od jezika, ki definira njihovo funkcionalnost ali vmesnik. Samo kadar uporabimo Internet kot medij za dostop do e-storitev lahko imenujemo takšno storitev Internetne storitve.

Spletne storitve

Spletne storitve so ohlapno povezane programske komponente, ki delujejo prek Interneta in uporabljajo standardne internetne tehnologije[4]. Sicer spadajo v podmnožico e-storitev in so na voljo odjemalcu z uporabo spletno orientiranih protokolov oziroma programov. Ob ločitvi logične in tehnične plasti lahko implementiramo spletne storitve v dveh različnih načinih:

- SOAP spletne storitve
- REST spletne storitve

V naslednjih odstavkih bomo na kratko opisali SOAP spletno storitev, medtem ko bomo REST spletnim storitvam namenili več pozornosti in prikazali implementacijo tudi na praktičnem primeru.

SOAP spletne storitve so osnovane na SOAP protokolu, ki je namenjen izmenjavi podatkov (v obliki XML) v porazdeljenem okolju. Osnovna enota komunikacije je sporočilo, za katerega SOAP protokol predvideva standarden format, osnovan na XML.

2.1.3 REST spletne storitve

REST spletne storitve so osnovane na REST arhitekturi, ki je ena izmed arhitekturnih stilov gradnje porazdeljenih sistemov[4]. Razvita je bila kot alternativa SOA arhitekturi. Ključna razlika med REST in SOA arhitekturo je, da je REST arhitektura orientirana glede na vire, med tem ko je SOA arhitektura storitveno orientirana. Ključni princip REST arhitekture je uporaba protokola HTTP in njegovih metod za dostop do enolično definiranega vira. Iz tega sledi, da mora imeti vsak vir svoj URI.

Naslavljanje virov

Pri definiranju virov z URI naslovi je pomembno, da so naslovi opisni, smiselni in konsistentni. REST arhitektura omogoča tudi več različnih URI-jev, ki kažejo na isti vir. Spletna aplikacija naj bi izpostavila ključne podatke kot vire, torej z naslovi - URI-ji. V našem primeru integracije informacijskega sistema smo izpostavili trenutne vremenske podatke kot tudi ostale statistične podatke. Recimo uporabnik dostopa do vira z trenutnimi podatki preko naslednjega URL naslova: `http://www.vp-zalec.net/rest/trenutni/trenutniXML`. Ob tem je potrebno poudariti, da se pri dostopu na strani strežnika ne ohranja stanje. Odjemalcu tudi ni potrebno pri dostopu do vira strežnik peljati v določeno stanje, da dobi trenutne podatke. Lahko govorimo o tem, da stanje aplikacije živi na odjemalcu, medtem ko stanje vira živi na strežniku.

Predstavitev virov

Isti vir lahko predstavimo na različne načine oziroma obliko. Recimo isti vir je lahko predstavljen v XML obliki, v TXT obliki, itd. V naši implementaciji nudimo predstavitev vira v štiri različnih oblikah:

- TXT
- HTML
- XML
- JSON

Pri pridobivanju različnih predstavitev za isti vir mora odjemalec strežniku nekako sporočiti zahtevano predstavitev. Imamo dve možnosti[4]:

- Vsaka predstavitev ima svoj URL naslov. Takšno rešitev uporabljamo v naši implementaciji.
- Pogajanje o vsebini (Content negotiation), kjer odjemalec nakaže predstavitev z parametri v HTTP glavi, npr. Accept-Language ali drugimi metapodatki (čas pošiljanja, IP naslov).

Dostop do virov

Do virov oziroma njihovih predstavitev se pogosto dostopa preko hiperpovezave. Odjemalec ponavadi dostopa do vira preko klika na povezavo, ne pa z vpisovanjem URI naslova. Takšno zaporedje klikov je stanje aplikacije, katerega si strežnik privzeto ne zapomni. V naši implementaciji imamo na strani `http://www.vp-zalec.net/storitve` seznam vseh virov v obliki hiperpovezav. Odjemalec izbere željeno predstavitev vira in strežnik pošlje odjemalcu dokument oziroma izbrani vir v željeni predstavitvi.

Operacija	SQL ukaz	HTTP metoda
Ustvari vir	INSERT	POST
Beri vir	SELECT	GET
Spremeni vir	UPDATE	PUT(POST)
Zbriši vir	DELETE	DELETE
Poizvedba po metapodatkih	(Systables)	HEAD

Tabela 2.1: HTTP metode[4].

Metode HTTP v RESTful storitvah

Protokol HTTP vsebuje standardne metode GET, POST, PUT, DELETE in HEAD. Tabela 2.1 prikazuje primerljive SQL ukaze z HTTP metodami.

RESTful spletne storitve uporabljajo HTTP metode za upravljanje, kreiranje in dostop do virov. V okviru naše implementacije uporablja odjemalec samo metodo GET, kjer strežnik pošlje v telesu odgovora predstavitev vira. GET metoda se obravnava kot varna in idempotentna metoda. Varna zato, ker se odjemalcu ob njeni uporabi ne more zgoditi nič. Idempotentna pa zato, ker če večkrat izvedemo isto HTTP zahtevo, ne bo nič drugače, kot če jo le enkrat.

2.1.4 Življenjski cikel storitev

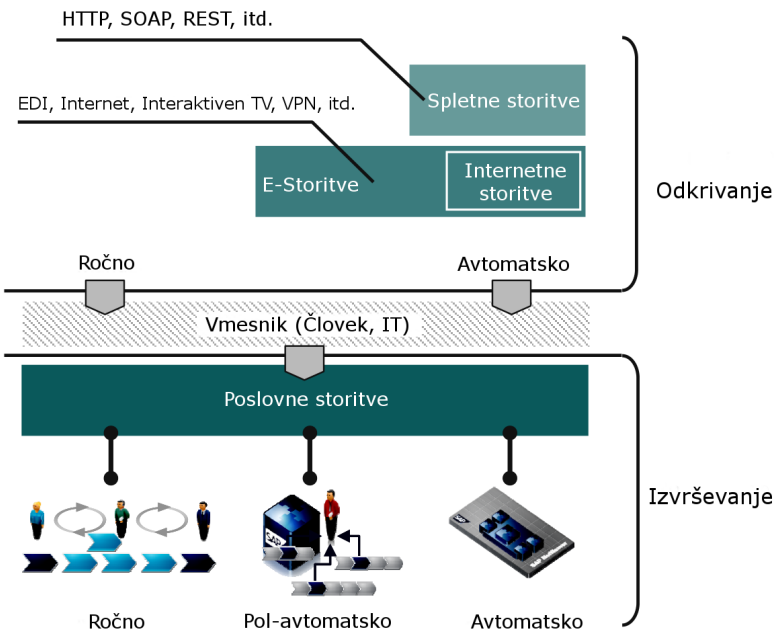
Življenjski cikel storitev vsebuje tri glavna stanja[2]:

- Razvoj storitev
- Odkrivanje storitev
- Izvrševanje storitev

Odkrivanje storitev se nanaša na medij in tehnologijo za iskanje in pošiljanje zahtev določeni storitvi. Izvrševanje storitev pa opisuje, kako se storitev izvrši, kjer je poudarek izvršitve glede na interakcijo človeka pri izvršitvi. Oba

življenjska cikla sta prikazana na sliki 2.1. Imamo tri možne scenarije (vse tri možnosti so prikazane na sliki 2.1):

- Človek - človek
- Človek - računalnik
- Računalnik - računalnik



Slika 2.1: Življenjski cikel storitev[2].

V naši implementaciji je odkrivanje storitev osnovano na IT, kot velja za vse Internetne storitve. Medtem ko imamo pri izvrševanju storitve dve možnosti in sicer človek - računalnik ali pa računalnik - računalnik. V prvem primeru človek pošlje zahtevo za izvršitev (klik na povezavo do storitve), računalnik (strežnik) obdela zahtevo in vrne rezultat, ki ga človek interpretira. V drugem primeru računalnik pošlje zahtevo za izvršitev, računalnik obdela zahtevo in vrne rezultat, ki ga računalnik interpretira oziroma nadalje obdela.

2.1.5 Razvijanje storitev

Razvijanje storitev vključuje množico aktivnosti, ki so namenjene za razvoj storitveno usmerjene rešitve. Pomembno je poudariti, da morajo imeti razvijalci dober pregled tako nad poslovno kot tudi nad tehnično plast storitve. Pred samo tehnično implementacijo je potrebno dobro analizirati poslovno okolje in procese, ter identificirati poslovne funkcije, ki jih lahko realiziramo kot storitve. Takšen pristop k razvijanju storitev se imenuje storitveno inženirstvo.

Definicija storitvenega inženirstva 1 *Storitveno inženirstvo je strukturiran pristop, ki omogoča uporabo naprednih konceptov kot so modeli ter nam omogoča lažje razumevanje strukture, implementacije, dokumentacije, vzdrževanja in spreminjanja elektronskih storitev[2].*

Z uporabo storitvenega inženirstva lahko prevedemo opis storitve iz naravnega jezika v množico modelov, ki predstavljajo tehnično implementacijo storitve. Tako imamo storitev opisano v različnih plasteh, kar omogoča razumevanje delovanja in implementacije same storitve iz različnih perspektiv. V našem primeru realizacije storitev se nismo posluževali naprednih konceptov zaradi majhne zahtevnosti realizacije. V primeru kompleksnejšega sistema storitev pa je dobro, da se držimo strukturiranega pristopa kot je določen v Zachman oviru.

2.2 Internet stvari

2.2.1 Uvod

Internet stvari (Internet of Things - IoT) predstavlja enolično identificirane objekte (stvari) in njihovo virtualno predstavitev na omrežju internet[16]. Imamo torej prepletanje fizičnega sveta z virtualnim svetom. Ne smemo pa na IOT gledati kot samo na naprave, ampak kot na celoten ekosistem. Torej

od dejanskih naprav, ki zbirajo podatke in jih pošiljajo čez medij kot je Internet, ter do strežnikov in programske opreme, na katerih se obdelujejo zbrani podatki. Če lahko Internet opišemo kot omrežje omrežij, lahko Internet stvari opišemo kot omrežje omrežij in stvari[9]. V svetu Internetnih stvari so vsi objekti realnega sveta povezljivi in dostopni digitalno, kot tudi analogno[11]. Objekti oziroma stvari komunicirajo med sabo in si izmenjujejo podatke. V sodobnem svetu obstaja velik interes po deljenju podatkov, to pomeni, da so podatki namenjeni komurkoli, oziroma so na voljo večjemu številu aplikacij, ne pa samo točno določeni. Če se osredotočimo na povezavo stvari lahko Internet stvari obravnavamo tudi kot digitalno povezovalno plast, ki povezuje obstoječo infrastrukturo in stvari[11]. Zaradi vedno večje razširjenosti se jih velikokrat omenja tudi kot naslednjo stopnjo razvoja Interneta, oziroma evolucija, ki bo spremenila naša življenja na mnogih področjih. Namreč smo že presegli točko, kjer je na Internet priključenih več naprav kot pa je ljudi na Zemlji[9].

2.2.2 Stvari

Stvari oziroma objekti v realnem svetu predstavljajo senzorje, ki se zavedajo svojega okolja in pretvarjajo karakteristike okolja v podatke. Zmožne so komuniciranja druga z drugo ter izvrševanja potrebnih akcij brez vmešavanja človeka, koncept komunikacije, ki ga poznamo pod kratico M2M[9]. Pomembno je poudariti koncept komunikacije, ki ne vsebuje človeka. Namreč ljudje so nezanesljivi pri podajanju podatkov o njihovem okolju. Meritve, ki jih opravljajo ljudje so velikokrat subjektivne in ne objektivne narave. Prav tako imamo ljudje omejen čas, hitrost in natančnost. Ravno to so pa glavne prednosti koncepta M2M.

Internet stvari združuje osebne elektronske naprave kot so PC, prenosni računalnik, pametni telefon, tablični računalnik, igralne konzole, televizija, itd[9]. Toda to je samo vrh ledene gor, zraven lahko dodamo še druge vsakdanje stvari kot so kuhinjske naprave, avtomati z hrano in pijačo in prevozna sredstva kot so avtomobili, vlaki, ladje, letala itd. Dejansko ne obstaja fizični

objekt, kateremu ne bi morali dodati čipa in ga spremeniti v povezani objekt.

2.2.3 Primer povezovanja stvari in programske opreme

Na praktičnem primeru bomo predstavili, kako smo integrirali stvari oziroma naprave z programsko opremo oziroma storitvami. Imamo glavno enoto, ki jo poganjajo sončne celice oziroma rezervni akumulator, ki se uporablja kot vir energije ob pomanjkanju zadostne količine sončnega sevanja. Na glavno enoto so priključeni trije senzorji in sicer:

- Senzor temperature in vlage (termometer in higrometer) (prikaz senzorja na sliki 2.2).

Slika 2.2: Senzor temperature in vlage (termometer in higrometer).

- Senzor za hitrost in smer vetra (anemometer) (prikaz senzorja na sliki 2.3).
- Senzor za merjenje količine padavin (prikaz senzorja na sliki 2.4).

Vsi trije senzorji so povezani na glavno enoto preko UTP dvo-žilnega kabla (povezava je prikazana na sliki 2.6). Senzorji zbirajo podatke o svoji okolici, torej merijo prej navedene parametre in jih pošiljajo v glavno enoto. V tem primeru imamo torej centralno vozlišče, kateremu pošiljajo ostala

Slika 2.3: Senzor za hitrost in smer vetra (anemometer).

vozlišča izmerjene podatke. Nadalje se podatki iz glave enote brezžično pošljejo do konzole. Brezžično pošiljanje uporablja frekvenco od 868.0 - 868.6 MHz z načinom FHSS. Konzola pridobiva podatke, ki jih glava enota pošlje brezžično ter jih zapiše v notranji pomnilnik. Konzolo lahko obravnavamo kot vmesni sloj, ki omogoča komunikacijo programske opreme z senzorji. Program nato prebere te podatke in jih zapiše v podatkovno bazo. Sedaj lahko s pomočjo informacijskega sistema izpostavimo podatke iz podatkovne baze kot storitve, torej uporabnik dostopa do podatkov, ki so jih pravkar izmerili senzorji.

2.2.4 Tehnologija

Internet stvari lahko realiziramo z najrazličnejšo tehnologijo. Toda preden bo lahko IOT polno zaživel je potrebno rešiti naslednje probleme[9]:

- Način za enolično identifikacijo vseh naprav.
- Zanesljivost omrežja
- Brezžični prenos podatkov
- Pametni senzorji z nizko porabo in ceno.

Slika 2.4: Senzor za merjenje kolicine padavin.

- Vgrajena inteligenca.
- Avtomatizacija in integracija.
- Varnost

V naslednjih odstavkih bomo na kratko opisali naštete probleme in morebitne tehnologije, ki jih rešujejo.

Način za enolično identifikacijo naprav

Vse te naprave bo potrebno nekako identificirati, oziroma nasloviti. Namreč za komunikacijo kot je M2M, mora imeti vsaka naprava svoj naslov. Odgovor se ponuja v uporabi IPv6 protokola. Ta protokol nam omogoča naslavljanje veliko večjega števila naprav kot pa njegov predhodnik IPv4. To je tudi razumljivo, saj je bil IPv4 razvit za povezovanje ljudi v Internet, medtem ko je nadgradnja v IPv6 bila narejena zaradi vedno večjega števila naprav, ki

Slika 2.5: Glavna enota z anteno in sončnimi celicami.

so povezani v Internet. Namreč IPv4 uporablja 32-bitni naslovni prostor, ki omogoča 4,294,967,296 (2^{32}) naslovov. Medtem ko IPv6 uporablja 128-bitni naslovni prostor kar omogoča 2^{128} oziroma $3.4 * 10^{38}$ naslovov. Pomembno je tudi omeniti enolično identifikacijo naprav preko URI naslova, torej prepletanje z RESTful spletnimi storitvami.

Zanesljivost omrežja

Omrežje mora biti za delovanje kritičnih senzorjev zanesljivo. Uporaba Interneta kot medija za prenos podatkov iz oblaka k odjemalcu, medtem ko za lokalno omrežje senzorjev uporaba intraneta. Pomembno je tudi raz-

Slika 2.6: Notranjost glave enote.

viti komunikacijske protokole, ki bodo preprečevali preveliko obremenjevanje omrežja[5]. Recimo uporaba učinkovitejših kod za izmenjavo podatkov kot pa recimo XML, ki je zelo potraten za prenos, saj je prilagojen človeku[5]. Glede na to, da imamo tu M2M komunikacijo, ni potrebe po formatih za prenos podatkov, ki so berljivi človeku, ampak se je potrebno osredotočiti na učinkovitost.

Brezžični prenos podatkov

Obstajajo štiri glavne vrste brezžičnega povezovanja več naprav. Razlikujejo se predvsem v številu povezanih naprav, hitrosti prenosa podatkov in dosegu:

- Wi-Fi
- 3G/4G
- Bluetooth
- ZigBee

Najvišje hitrosti prenosa so pri Wi-Fi, najnižje pa pri ZigBee. Toda kar je pomembno je število naenkrat povezanih naprav, oziroma vozlišč. Tukaj ima ZigBee veliko prednost, saj podpira kar do 64000 vozlišč na omrežje, kjer lahko dodamo omrežne koordinatorje in sestavimo omrežja z še večjim številom vozlišč. Namreč senzorji ne potrebujejo pasovne širine, ki jo ponujata Wi-Fi in 3G/4G, ampak potrebujejo majhne hitrosti, srednji doseg in možnost povezovanja velikega števila vozlišč.

Pametni senzorji z nizko porabo in ceno

Osnova za nadaljnji razvoj IOT je poceni elektronika in nizka poraba električne energije. Nizka cena elektronike nam omogoča integracijo v najbolj vsakdanje naprave[10]. Medtem ko nam nizka poraba električne energije omogoča, da ni več potrebno da bi bile naprave priključene na električno omrežje, oziroma da bi bilo potrebno menjavati baterije na vsakih nekaj mesecev. Razvoj gre v smer, kjer senzor pridobiva električno energijo iz okolice (sončne celice). Takšna rešitev je uporabljena tudi v našem sistemu, kjer se centralna enota in senzorji, ki so priključeni na njo napajajo z sončnimi celicami je prikazana na sliki 2.5.

Vgrajena inteligenca

Velik poudarek na arhitekturi, ki je dogodkovno-poganjanja (event-driven). Senzor naj bi pomen dogodka določil ne samo na deterministični način, ampak tudi v kakšnem kontekstu je prišlo do tega dogodka.

Avtomatizacija in integracija

Osnova za avtomatizacijo naprav so M2M protokoli, ki omogočajo žično in brezžično komunikacijo z ostalimi napravami. Eden od pomembnejših delov celotnega sistema pa bodo tudi strežniki oziroma superračunalniki v oblaku, ki bodo iz zbranih podatkov senzorjev izluščili pomembne informacije, oziroma interpretirali pridobljene informacije za nadaljnje odločitve.

Varnost

Varnost lahko zagotovimo tako, da so podatki nekaterih senzorjev na voljo samo v okviru lokalnega omrežja, oziroma če so na voljo celotnem Internetu samo uporabnikom z priviligiranim dostopom.

Poglavje 3

Uporabljeni koncepti pri razvoju IS

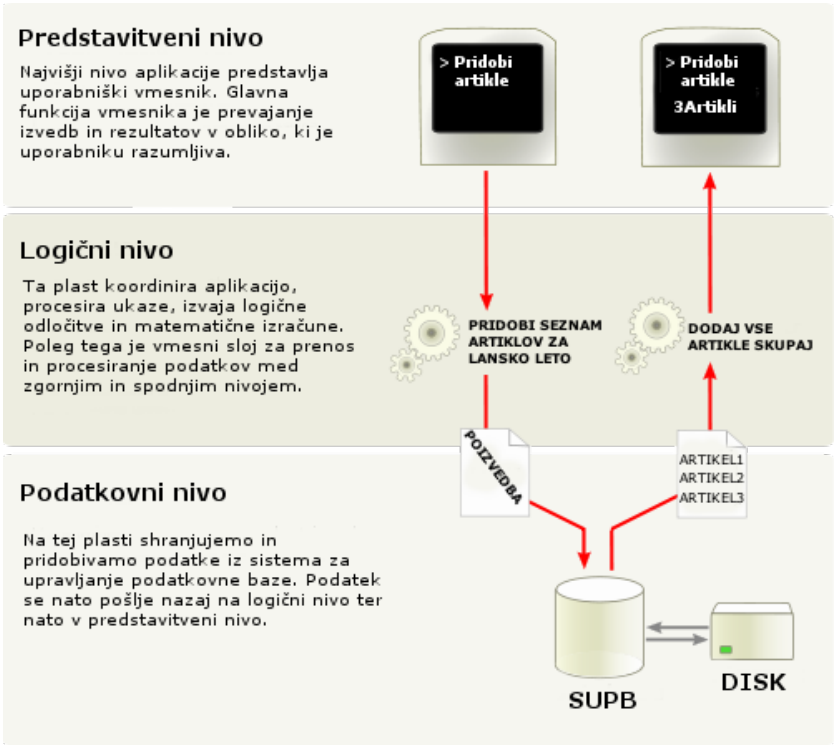
Opis, razlaga delovanja in primeri tipične uporabe konceptov. Razlogi za uporabo določenih konceptov ter njihove prednosti pred ostalimi koncepti.

3.1 Arhitektura in načrtovalski vzorec

Uporabljamo tri-nivojsko arhitekturo in načrtovalski vzorec MVC, ki olajšata in skrajšata čas razvoja informacijskega sistema.

3.1.1 Tri-nivojska arhitektura

Pri integraciji informacijskega sistema sledimo tri-nivojski arhitekturi. Tri-nivojska arhitektura je odjemalec-strežnik arhitektura, kjer so predstavitveni nivo, nivo logike in nivo podatkov ločeni procesi[25]. Uporaba takšne arhitekture nam omogoča razvoj fleksibilnega in ponovno uporabljivega sistema. Več nivojev nam tudi omogoča ločen razvoj posameznih nivojev, kjer lahko različni razvijalci istočasno razvijajo sistem na različnih nivojih. Poleg tega moramo ob dodajanju novih funkcionalnosti, oziroma spremembi obstoječih spremeniti samo določene dele oziroma nivoje, ne pa celotnega sistema.



Slika 3.1: Tri-nivojska arhitektura[25].

Opis nivojev v tri-nivojski arhitekturi (slika 3.1 prikazuje delovanje tri-nivojske arhitekture):

- Predstavitveni nivo: Je namenjen predstavitvi podatkov uporabniku preko uporabniškega vmesnika. Uporabniški vmesnik je lahko v tem primeru konzola, grafični vmesnik aplikacije ali spletni brskalnik.

- Nivo logike: Vsebuje logiko za procesiranje zahtev iz predstavitvenega nivoja in pridobivanje podatkov iz podatkovnega nivoja.

- Podatkovni nivo: Namenjen za trajno shranjevanje podatkov. Ob poizvedbi iz nivoja logike se podatki pridobijo iz podatkovne baze oziroma podatkovnega sistema ter se pošljejo naprej proti nivoju logike, kjer se obdelajo.

3.1.2 MVC

Pri razvoju informacijskega sistema se držimo načrtovalskega vzorca MVC. Najprej definirajmo, kaj je to načrtovalski vzorec.

Definicija načrtovalskega vzorca 1 *Načrtovalski vzorec opisuje problem, ki se večkrat pojavlja v našem okolju, podaja jedro njegove rešitve na tak način, da lahko idejo rešitve uporabimo v milijon različnih primerih, ne da bi pot od ideje do rešitve prehodili na enak način[6].*

Načrtovalski vzorec uporabljamo zaradi naslednjih točk[6]:

- Učinkovito reševanje problemov ni povezano le z znanjem uporabe modernih tehnologij, ampak tudi z izkušnjami.
- Ne-podvajanje truda.
- Učimo se na napakah drugih - kako obvarovati novince v načrtovanju pred ponavljanjem napak?
- Kako ohranjati ekspertizo v podjetju tudi po odhodu ekspertov?
- Omogočiti razširljivost, fleksibilnost.
- Napačne odločitve v zasnovi pokvarijo uporabnost in privlačnost.
- Kako zapisovati izkušnje?

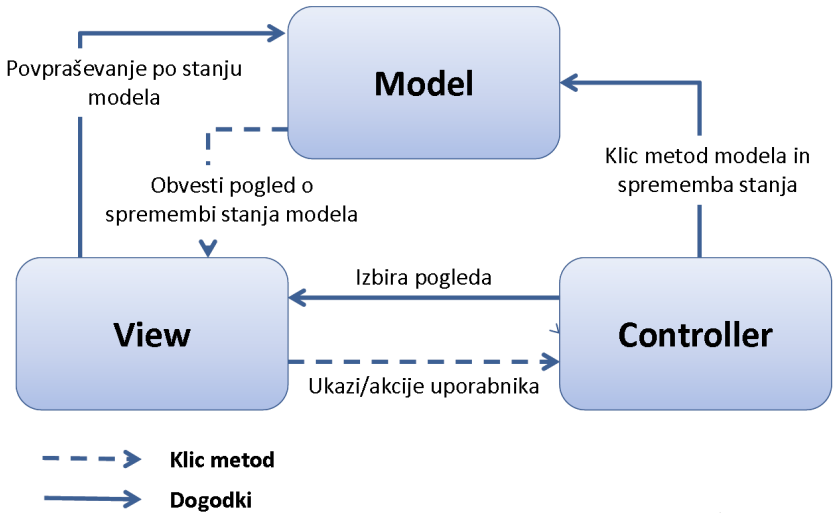
Načrtovalski vzorec MVC nam omogoča naslednje stvari[6]:

- Ločevanje med podatkovnim slojem in uporabniškim vmesnikom.
- Ponovna uporaba poslovne logike.
- Razvoj več uporabniških vmesnikov, ne da bi pri tem spreminjali kodo poslovne logike.
- Manj ponavljanja iste kode.

- Lažje vzdrževanje (popravljanje) in nadgrajevanje kode.

MVC vzorec je razdeljen na naslednje sklope[6] (slika 3.2 prikazuje povezavo in interakcijo med posameznimi sklopi):

- Model:
  - Predstavlja podatke aplikacije in vsebuje logiko za dostop in manipulacijo teh podatkov.
  - Vsebuje metode za dostop in posodobitev stanja modela ter metode za izvajanje kompleksnih procesov.
  - Neodvisen od nivoja podatkovnega dostopa.
- Pogled (View):
  - Pogled je zadolžen za prikazovanje stanja modela.
  - Lahko imamo več različnih pogledov (recimo ločen pogled za mobilne naprave).
  - Pogled se posodobi, ko je bil spremenjen model in ga je o tem tudi obvestil (razširjen MVC).
  - Pogled posreduje uporabnikove vhode k krmilniku (Controller).
  - V nekaterih primerih lahko dostopa do podatkov modela (razširjen MVC).
- Krmilnik (Controller):
  - Zadolžen za prestrezanje in prevod uporabnikovih vnosov v akcije, ki jih mora izvesti model.
  - Zadolžen za izbiro naslednjega pogleda glede na uporabnikove zahteve in rezultat operacij modela.



Slika 3.2: MVC vzorec[6].

3.2 Platforma, orodja in tehnologije

Platforme, orodja in tehnologije so v pomoč pri razvoju informacijskega sistema. Z njihovo uporabo se izognemo razvijanju sistemov, ki so že bili razviti prav za enake namene. Njihova uporaba je močno priporočljiva, saj so pogosto dobro testirana in uporabljena v mnogih sistemih.

3.2.1 Java EE 6 razvojna platforma

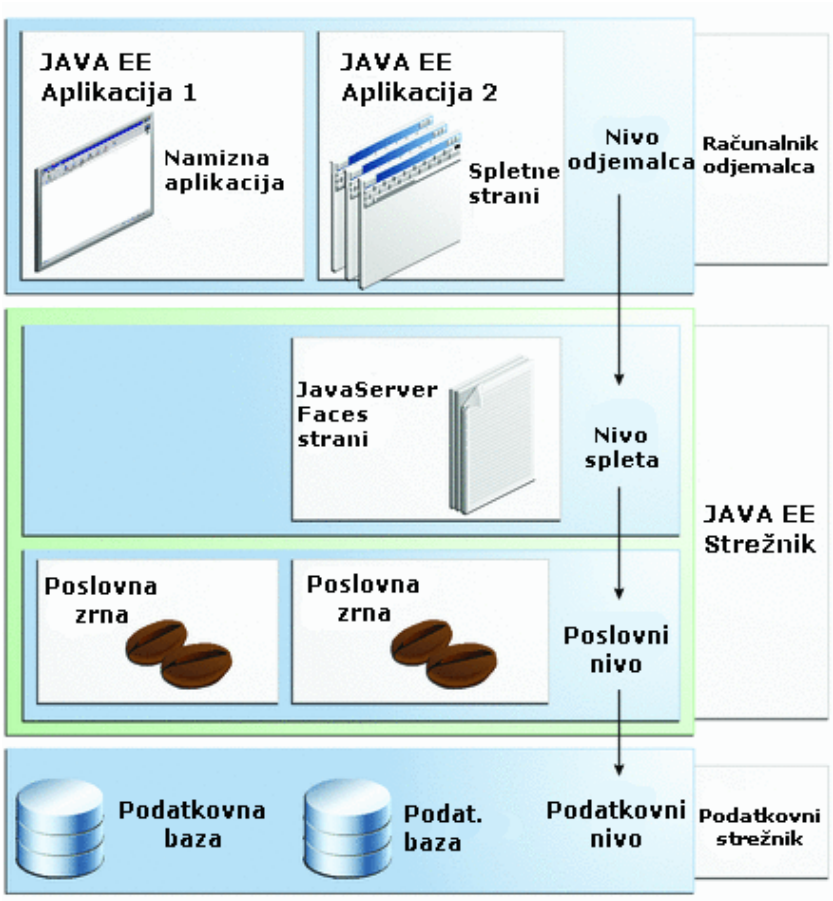
Java EE 6 platforma je osnovana na programskem jeziku Java in virtualnem stroju Java (JVM). Cilji platforme so naslednji[1]:

- Poenostavljen razvoj kompleksnih informacijskih sistemov.
- Integracija distribuirane več-nivojske arhitekture.
- Modularno razvijanje informacijskega sistema.
- Enostavno pakiranje v končni izdelek.

Kot smo že omenili definira Java EE aplikacijski model distribuirano več-nivojsko arhitekturo. Aplikacijska logika je deljena v komponente glede na

funkcije in te komponente so naložene na različne nivoje, glede na to kateremu nivoja pripada določena funkcionalnost. Aplikacijski model je razdeljen na naslednje nivoje[1] (slika 3.3 prikazuje arhitekturo po nivojih):

- Nivo odjemalca (Client Tier).
- Nivo spleta (Web Tier).
- Poslovni nivo (Bussines Tier).
- Podatkovni nivo (Database Tier).



Slika 3.3: Prikaz več-nivojske arhitekture v Java EE modelu[1].

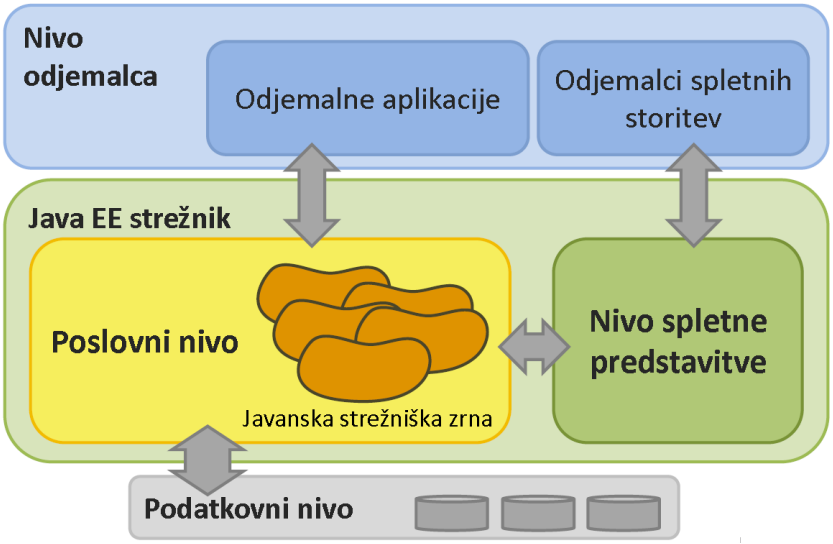
Čeprav imamo štiri nivoje, kot je prikazano na sliki **??**, se Java EE večnivojske aplikacije ponavadi obravnavajo kot tri-nivojske aplikacije zato ker so razdeljene preko treh različnih lokacij:

- Računalnik odjemalca.
- Java EE aplikacijski strežnik.
- Podatkovni strežnik.

Sledi opis Java EE komponent, ki smo jih uporabili za realizacijo informacijskega sistema.

EJB

EJB - Javanska strežniška zrna (Enterprise Java Beans). So strežniške komponente za modularno izgradnjo poslovno informacijskih aplikacij (Enterprise Software)[8] (slika 3.4).



Slika 3.4: Umestitev Javanskih strežniških zrn[8].

Java EE definira naslednje tipe javanskih strežniških zrn[8]:

- Sejna zrna (Session beans) opravljajo poslovne operacije, orkestrirajo transakcije ter upravljajo z dostopi.
- Sporočilna zrna (Message-driven beans) so asinhrona in odgovarjajo na zunanje dogodke.

Lastnosti EJB[8]:

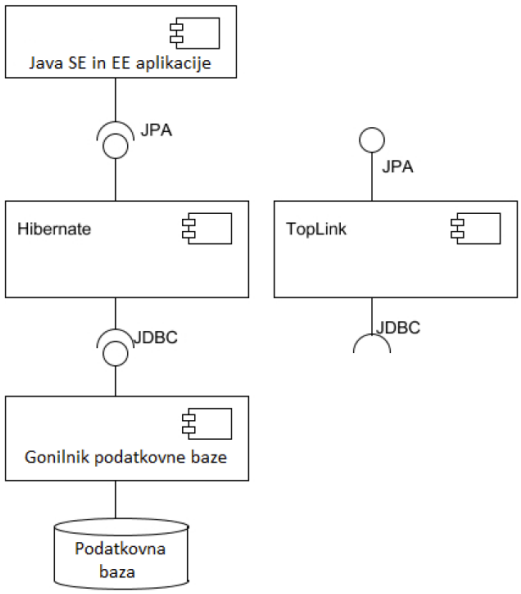
- Pomembna lastnost javanskih zrn je opisno določevanje vedenja z uporabo deklarativnih metapodatkov. To omogoča prilagodljivost vedenja zrna brez implementiranja logike.
- Metapodatke podamo v XML obliki ali z anotacijami.
- Javanska zrna imajo samodejno določeno privzeto vedenje, zato razvijalci določajo le izjemno (ne privzeto) vedenje (Configuration by Exception).
- Aplikacije večjih razsežnosti zahtevajo preprosto razširljivost. EJB strežnik podpira vzdrževanje bazenov virov (Resource Pooling) za čim večjo ponovno uporabo objektov in virov.

JPA

JPA (Java Persistence API) je specifikacija zahtev za objektno-relacijsko preslikavo. Je tehnika za premoščanje med objektno-orientiranim modelom in relacijsko podatkovno bazo[7] (slika 3.5).

Lastnosti JPA[7]:

- Kreiranje entitet je preprosto tako kot kreiranje razredov za serializacijo.
- Podpira velike nabore podatkov.
- Zagotavlja konsistentnost podatkov.
- Omogoča hkratno uporabo.

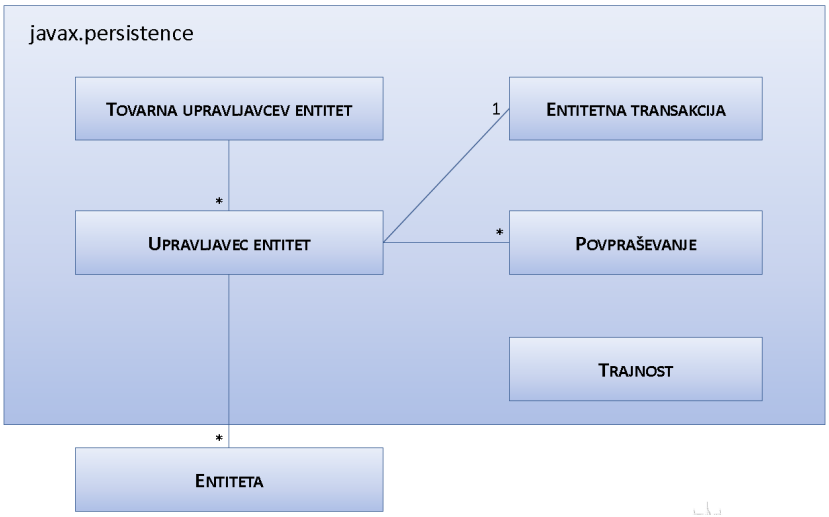


Slika 3.5: Način JPA preslikave[7].

- Ponuja povpraševalne sposobnosti JDBC.
- Omogoča uporabo naprednih objektnih konceptov.
- Temelji na standardni specifikaciji in se izogiba odvisnosti od ponudnika.
- Osredotoča se na relacijske podatkovne baze.
- Je preprost za uporabo.

JPA zajema naslednja področja[7]:

- Sam API za izvajanje osnovnih (CRUD) operacij, ki omogoča shranjevanje, posodabljanje, pridobivanje in odstranjevanje objektov iz podatkovne baze.
- Deklarativen način za izvedbo preslikave med objekti in relacijskimi podatkovnimi bazami.
- Povpraševalni jezik za pridobivanje objektov iz podatkovne baze na ustrezen način (optimizacija), brez potrebe po pisanju SQL stavkov.



Slika 3.6: Struktura JPA[7].

JBoss Hibernate Ker je JPA samo specifikacija zahtev, ki jih morajo implementacije upoštevati in sama po sebi ne ponuja nobenih funkcionalnosti smo uporabili JBoss Hibernate[19] JPA implementacijo, natančno verzijo 4.0.1.

JSF

JSF tehnologija je specifikacija ogrodja za gradnjo spletnih strani. Specifikacija predvideva naslednje glavne komponente[1]:

- Komponentno ogrodje za uporabniški grafični vmesnik.
- Fleksibilen model za izris komponent v različnih HTML formatih oziroma različnih tehnologijah.
- Standardni RenderKit za generiranje HTML/4.01.

Komponente, ki so sestavni del ogrodja za uporabniški grafični vmesnik podpirajo naslednje funkcionalnosti[1]:

- Preverjanje vhodnih podatkov.

- Upravljanje z dogodki.
- Pretvarjanje podatkov med objekti modela in komponentami.
- Konfiguracijo navigacije strani.
- Izrazni jezik EL.

Oracle Mojarra Za JSF implementacijo smo uporabili Oracle Mojarra[28] implementacijo.

JNDI

JNDI API omogoča aplikacijam dostop do imenskih storitev, kot so recimo LDAP, NDAP in DNS. Tako lahko omogočimo aplikaciji asociranje atributov z objekti in iskanje ter pridobivanje lokalnih ali oddaljenih objektov glede na atribute objektov[1].

CDI

CDI definira množico storitev v okviru konteksta, ki omogočajo razvijalcem vstavljanje odvisnosti na EJB zrna [3.2.1] kot tudi na ostale vire (npr. Data Source). S pomočjo te tehnologije je zelo enostavno vstaviti vire v JSF [3.2.1] spletni aplikaciji. CDI je bil načrtovan v mislih za objekte, ki ohranjajo stanje (stateful), ampak omogoča širšo uporabo kot je integracija različnih komponent na tipsko varen način[1].

3.2.2 MySQL

Za realizacijo podatkovnega nivoja smo uporabili odprto kodni sistem za upravljanje relacijske podatkovne baze (SUPB), ki teče kot strežni proces in nudi sočasni dostop do podatkovne baze večjemu številu uporabnikov[26]. Pri implementaciji podatkovnega modela uporabljamo naslednje dele sistema za upravljanje relacijske podatkovne baze.

Shranjene procedure

Definicija shranjene procedure:

Definicija shranjene procedure 1 *Shranjena procedura je podprogram, ki je na voljo aplikacijam, ki dostopajo do relacijskega podatkovnega sistema. Procedura je shranjena v podatkovni bazi[3].*

Shranjene procedure se uporabljajo zaradi naslednjih razlogov:

- Preverjanje podatkov.
- Konsolidacija in centralizacija logike.
- Izvrševanje velikega števila SQL poizvedb.

Prožilci

Definicija prožilca podatkovne baze:

Definicija prožilca 1 *Prožilec podatkovne baze je procedura, ki se avtomatsko izvrši kot odziv na dogodek nad določeno tabelo ali pogledom v podatkovni bazi[3].*

Prožilci se večinoma uporabljajo za:

- Vzdrževanje integritete informacije v podatkovni bazi.
- Zapisovanje sprememb nad tabelami.
- Izvrševanje poslovnih pravil.
- Izboljšanje zmogljivosti.

Pogledi

Definicija pogleda podatkovne baze:

Definicija pogleda 1 *Pogled je navidezna relacija, ki ne obstaja v relacijski bazi, temveč se dinamično kreira takrat, ko nekdo po njej povprašuje[3].*

Namen pogledov je naslednji[3]:

- Predstavljajo odličen mehanizem za zagotavljanje varnosti saj skrivajo posamezne dele podatkovne baze pred določenimi uporabniki.
- Uporabnikom dajejo možnost, da do podatkov dostopajo na prilagojen način, kar omogoča da so lahko isti podatki s strani različnih uporabnikov v istem času vidni na različne načine.
- Poenostavljajo kompleksne operacije nad osnovnimi relacijami.

3.2.3 MySQL Workbench

MySQL Workbench[27] je orodje za modeliranje podatkovne baze s pomočjo grafičnega vmesnika (slika 3.7 predstavlja orodje). Poleg modeliranja podatkovne baze podpira še naslednje stvari:

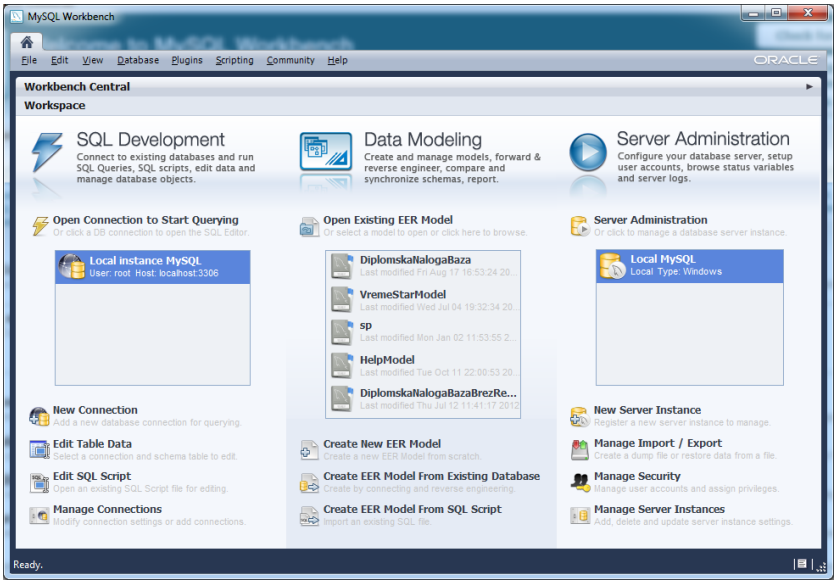
- Administracija obstoječih podatkovnih baz.
- Poizvedovanje, brisanje in spreminjanje podatkov nad obstoječimi podatkovnimi bazami.

3.2.4 Eclipse

Eclipse je razvojno okolje[12] za razvoj aplikacij v različnih programskih jezikih kot so Java, Ada, C, C++, COBOL, Haskell, Perl, PHP, Python, itd. (slika 3.8 predstavlja orodje). Podpira tudi namestitev dodatnih vtičnikov za razširitev funkcionalnosti.

JBoss Tools

JBoss Tools je množica vtičnikov za razvojno okolje Eclipse, ki vsebuje dodatne funkcionalnosti za razvoj Java EE aplikacij, kjer se za aplikacijski strežnik uporablja JBoss aplikacijski strežnik.



Slika 3.7: Zaslonski posnetek domače strani orodja MySQL Workbench.

3.2.5 JBoss AS 7.1.1

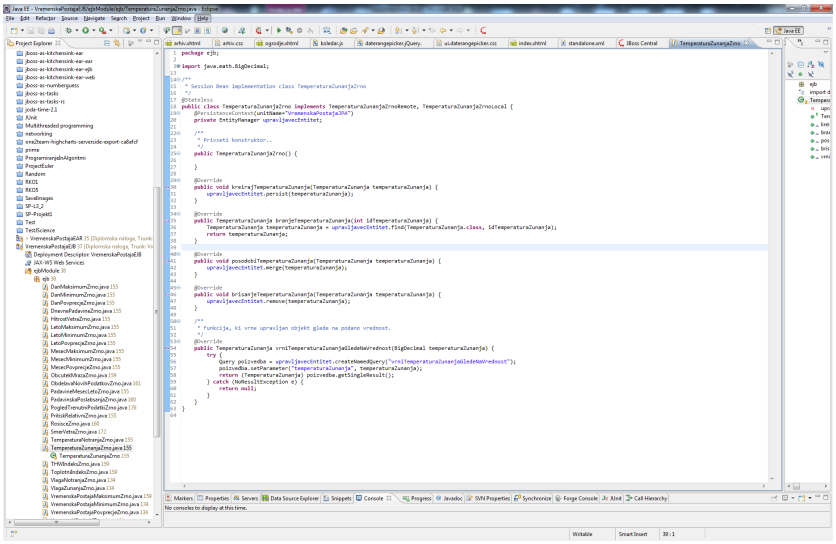
JBoss AS 7.1.1[18] je aplikacijski strežnik, ki implementira celotni sklad platforme Java EE 6. Implementiran informacijski sistem se izvaja na takšnem strežniku. Strežnik se uporablja tako v produkciji kot v razvoju.

3.2.6 Podporne knjižnice

Pri razvoju informacijskega sistema uporabljamo veliko podpornih knjižnic, ki jih razvijajo tuje organizacije. Te knjižnice so nam v pomoč pri razvoju, saj olajšajo in skrajšajo razvoj informacijskega sistema. V naslednjih odsekih so opisane najbolj pomembne knjižnice in njihova uporaba.

Primefaces

Primefaces[29] je JSF [3.2.1] komponenta knjižnica. Vsebuje bogati nabor komponent, ki so implementirane s pomočjo razširjene Javascript knjižnice JQuery [3.2.7] in JQuery UI [3.2.7]. Podpira tudi JQuery UI ThemeRoller CSS ogrodje. Na voljo je tudi nabor komponent posebej za mobilne naprave.



Slika 3.8: Zaslonski posnetek razvojnega okolja Eclipse.

Gson

Gson[13] je odprto kodna Java knjižnica, katere namen je omogočiti serializacijo in deserializacijo Java objektov v oziroma iz formata JSON.

Joda Time

Joda Time[21] je odprto kodna Java knjižnica, ki se lahko uporabi kot nadomestilo za Java razrede, ki obravnavajo datum in čas. Njena glavna prednost je veliko večji nabor metod nad datumom in časom ter boljšo zmogljivost glede na uradno Java knjižnico.

JScience

JScience[22] je Java knjižnica, ki se večinoma uporablja v znanstveni sferi. Vsebuje vnaprej definirane matematične in fizikalne strukture, kot tudi učinkovite implementacije celih, racionalnih, realnih in kompleksnih števil.

Javolution

Javolution[17] je Java knjižnica, ki ponuja rešitve za realno časovne, vgrajene in visoko odzivne aplikacije.

3.2.7 Podporne knjižnice za nivo odjemalca

Na nivoju odjemalca uporabljamo dodatne Javascript knjižnice.

JQuery

JQuery[23] je Javascript knjižnica narejena z namenom poenostaviti skriptiranje na odjemalčevi strani. Ponuja abstrakcijo pogostih nalog na odjemalčevi strani kot so:

- Prečkanje DOM.
- Upravljanje z dogodki.
- Animacija.
- AJAX podpora.

JQuery-UI

JQuery-UI[24] je Javascript knjižnica, ki razširja funkcionalnost Javascript knjižnice JQuery. Razširjene so naslednje stvari:

- Interakcija.
- Animacija.
- Pripomočki (widgets).

Highcharts

Highcharts[14] je Javascript knjižnica, ki je namenjena izrisu interaktivnih in grafično bogatih grafov v spletnih aplikacijah oziroma spletnih straneh. Knjižnica omogoča izris različnih tipov grafov kot tudi izvoz v različne formate kot so PNG, JPG, PDF ali SVG.

Poglavje 4

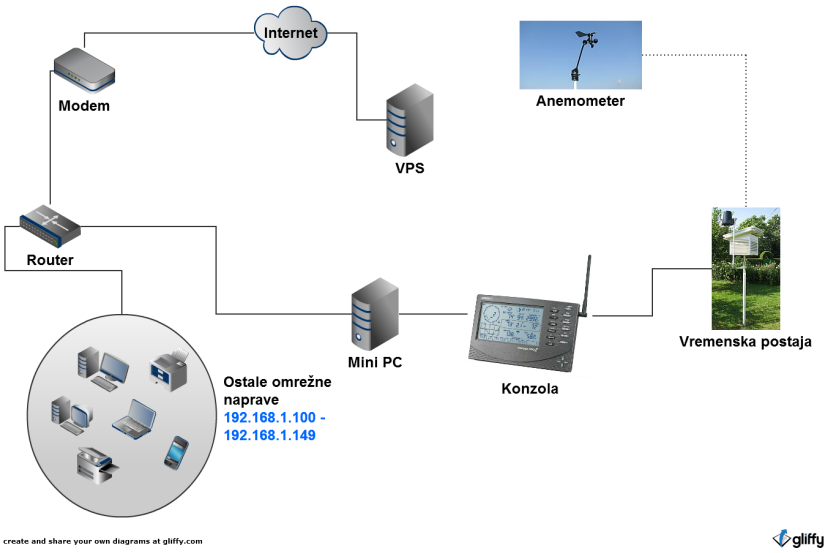
Arhitektura celotnega sistema

Sistem za katerega smo razvili informacijski sistem je sestavljen iz naslednjih fizičnih komponent:

- Meteorološka hiška.
- Meteorološka postaja Davis Vantage Pro2:
  - Senzor temperature, vlažnosti.
  - Dežemer.
  - Anemometer.
  - Oddajnik.
  - Konzola (sprejemnik).
- Mini računalnik za branje podatkov iz konzole in pošiljanje v oblak. Na računalniku teče OS Windows 7, WeatherLink in lastni program spisan v programskem jeziku Java.
- VPS kjer teče JBoss AS 7.1.1 in MySQL SUPB.

Shema celotnega sistema je prikazana na sliki 4.1. Koraki delovanja celotnega sistema:

(a) Meteorološka postaja zbira podatke in jih brezžično pošilja na konzolo.



Slika 4.1: Shema celotnega sistema.

(b) Konzola pošilja podatke na mini PC preko serijskega vmesnika RS232.

(c) WeatherLink bere podatke iz konzole in jih zapisuje v datoteke.

(d) Program v programskem jeziku Java bere datoteke in zapisuje nove podatke v bazo, ki se nahaja na VPS.

(e) Odjemalec dostopa do spletne strani oziroma strežnika, ki se nahaja na VPS.

Poglavje 5

Opis realizacije informacijskega sistema

Realizacijo informacijskega sistema smo razdelili na več sklopov. Najprej bomo začeli z opisom zasnove podatkovnega modela, nadaljevali z opisom zasnove programske opreme za polnjenje podatkovne baze ter na koncu z zasnovo informacijskega sistema na aplikacijskem strežniku.

5.1 Zasnova podatkovnega modela

Tabele relacijske podatkovne baze smo za potrebe opisa razdelili v sklope, kateremu pripadajo na podlagi podatkov, ki jih vsebujejo:

- Glavna tabela.
- Tabele z zalogo vrednosti.
- Tabele statističnih podatkov.
- Tabele padavin.
- Pomožne tabele.

5.1.1 Glavna tabela VremenskiPodatki

Vsebuje zapise (glej dodatek A, podpoglavje A.2) izmerjenih meteoroloških podatkov v uporabniško določenem intervalu (v našem primeru je interval 1 minuta). Tabela ne vsebuje dejanskih izmerkov za posamezni meteorološki parameter ampak sklice na pomožne tabele, kjer posamezna tabela vsebuje zalogo vrednosti za določeni meteorološki parameter (npr. za zunanjo temperaturo je interval vrednosti od -40°C do +65°C). Poleg tega vsebuje še nekaj metapodatkov, kot so npr. datum in čas izmerjene vrednosti, opombe in nekaj dejanskih podatkov, ki jih zaradi njihove narave ni mogoče zapisati v pomožne tabele kot zalogo vrednosti in potem uporabiti sklic na te tabele.

5.1.2 Tabele z zalogo vrednosti

Vsebujejo zalogo vrednosti za določen meteorološki parameter. Vsaka tabela vsebuje dva atributa:

- Umetni primarni ključ.
- Meteorološki parameter, ki predstavlja zalogo vrednosti.

Na te tabele se sklicujemo z uporabo tujega ključa iz glavne tabele VremenskiPodatki (5.1.1). Relacije so tipa ena proti mnogo (1:n). Vsaka povezava je neobvezujoča iz obeh strani, namreč:

- Vsaka n-terica iz pomožne tabele pripada nič, eni ali več n-tericam iz glavne tabele. Namreč če bi bila obveznost iz strani pomožne tabele, potem bi morala vsaka n-terica iz pomožne tabele pripadati vsaj eni ali več n-teric iz glavne tabele.
- Vsaka n-terica iz strani glavne tabele se sklicuje na nič ali eno n-terico iz pomožne tabele. Če bi bila obveznost iz strani glavne tabele, potem bi se morala vsaka n-terica iz glavne tabele sklicevati na točno eno n-terico iz pomožne tabele. To pa ne bi bilo najbolje, saj takšna obveznost zahteva sklic (torej vrednost tujega ključa ne sme biti NULL).

Namreč lahko se zgodi da zaradi različnih vzrokov manjka podatek za določen meteorološki parameter in tako je potrebno zapisati v bazo NULL vrednost. To bi sicer lahko rešili tako, da bi dodali v vsako pomožno tabelo n-terico z vrednostjo NULL.

Prednosti realizacije z uporabo sklicev so naslednje:

- Veliko hitrejše sortiranje podatkov po posameznih atributih. Namreč zavedati se je potrebno, da število n-teric s časom linearno narašča. Namreč vsako eno minuto se doda nova n-terica, to znese 60 n-teric na uro, 1440 n-teric na dan oziroma 525,600 n-teric na leto. Po več letih zbiranja podatkov bo tako že več milijonov n-teric v bazi. Če bi v n-terico glavne tabele zapisovali dejanske podatke, potem imamo pri sortiranju po naraščajočih/padajočih vrednostih časovno zahtevno operacijo. Namreč potrebno je sortirati podatke, ki vsebujejo decimalne vrednosti. Če pa imamo cela števila, torej v našem primeru sklice na dejanske vrednosti, pa je sortiranje veliko hitrejše. Namreč večja kot je številka sklica, večja je tudi dejanska vrednost meteorološkega parametra.

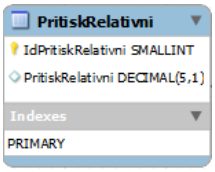
- Omejena zaloga vrednosti. Namreč s tem je nemogoče, da bi shranili v bazo vrednost, ki je glede na specifikacije senzorjev meteorološke postaje nemogoča. Torej s tem omejimo zalogo vrednosti, ki je lahko zapisana v bazo.

Tabela PritiskRelativni

Tabela vsebuje zalogo vrednosti relativnega pritiska (slika 5.1) iz intervala [540.0 - 1100.0] z natančnostjo 0.1. Zapisi predstavljajo podatek o relativnem pritisku v enotah hPa.

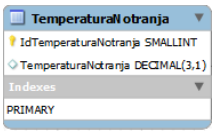
Tabela TemperaturaNotranja

Tabela vsebuje zalogo vrednosti notranje temperature (slika 5.2) iz intervala [0.0 - 60.0] z natančnostjo 0.1. Zapisi predstavljajo podatek o notranji



Slika 5.1: Struktura tabele PritiskRelativni.

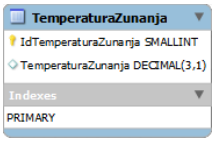
temperaturi v enotah °C.



Slika 5.2: Struktura tabele TemperaturaNotranja.

Tabela TemperaturaZunanja

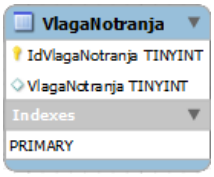
Tabela vsebuje zalogo vrednosti zunanje temperature (slika 5.3) iz intervala [-40.0 - 65.0] z natančnostjo 0.1. Zapisi predstavljajo podatek o zunanju temperaturi v enotah °C.



Slika 5.3: Struktura tabele TemperaturaZunanja.

Tabela VlagaNotranja

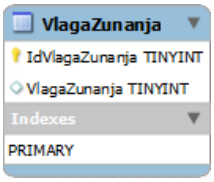
Tabela vsebuje zalogo vrednosti notranje vlage (slika 5.4) iz intervala [1 - 100] z natančnostjo 1. Zapisi predstavljajo podatek o notranji relativni vlagi v enotah %.



Slika 5.4: Struktura tabele VlagaNotranja.

Tabela VlagaZunanja

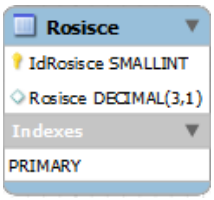
Tabela vsebuje zalogo vrednosti zunanje vlage (slika 5.5) iz intervala [1 - 100] z natančnostjo 1. Zapisi predstavljajo podatek o zunanju relativni vlagi v enotah %.



Slika 5.5: Struktura tabele VlagaZunanja.

Tabela Rosisce

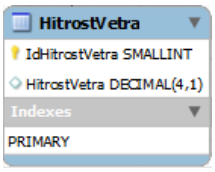
Tabela vsebuje zalogo vrednosti rosišča (slika 5.6) iz intervala [-76.0 - 54.0] z natančnostjo 0.1. Zapisi predstavljajo podatek o zunanjem rosišču v enotah °C.



Slika 5.6: Struktura tabele Rosisce.

Tabela HitrostVetra

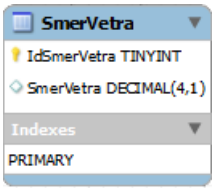
Tabela vsebuje zalogo vrednosti hitrosti vetra (slika 5.7) iz intervala [0.0 - 290.0] z natančnostjo 0.1. Zapisi predstavljajo podatek o hitrosti vetra v enotah km/h.



Slika 5.7: Struktura tabele HitrostVetra.

Tabela SmerVetra

Tabela vsebuje zalogo vrednosti smeri vetra (slika 5.8) iz intervala [0.0 - 337.5] z korakom 22.5. Zapisi predstavljajo podatek o smeri vetra v enotah °.



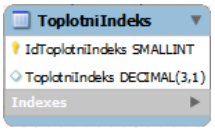
Slika 5.8: Struktura tabele SmerVetra.

Tabela ToplotniIndeks

Tabela vsebuje zalogo toplotnega indeksa (slika 5.9) iz intervala [-40.0 - 74.0] z natančnostjo 0.1. Zapisi predstavljajo podatek o notranjem in zunanjem toplotnem indeksu v enotah °C.

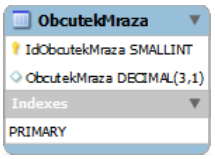
Tabela ObcutekMraza

Tabela vsebuje zalogo občutka mraza (slika 5.10) iz intervala [-79.0 - 57.0] z natančnostjo 0.1. Zapisi predstavljajo podatek o občutku mraza v enotah



Slika 5.9: Struktura tabele ToplotniIndeks.

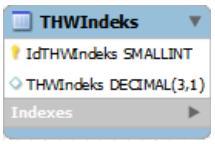
°C.



Slika 5.10: Struktura tabele ObcutekMraza.

Tabela THWIndeks

Tabela vsebuje zalogo THW indeks (slika 5.11) iz intervala [-68.0 - 74.0] z natančnostjo 0.1. Zapisi predstavljajo podatek o THW indeksu v enotah °C.



Slika 5.11: Struktura tabele THWIndeks.

5.1.3 Tabele z statističnimi podatki

Podatkovna baza vsebuje tri različne množice tabel z statističnimi podatki:

- Tabele za minimume.
- Tabele za maksimume.
- Tabele za povprečja.

Vsaka množica vsebuje tabele za naslednja časovna obdobja:

- Dan
- Mesec
- Leto
- Celotno obdobje od začetka meritev

Struktura tabel za maksimume, minimume in povprečja (glej dodatek A, podpoglavje A.1) je naslednja:

- Umetni primarni ključ tabele.
- Datum zapisa, ki ga potrebujemo za lažje iskanje zapisov glede na datum.
- Množica sklicev na n-terice glavne tabele za vsak vremenski parameter, za katerega beležimo statistično vrednost. Tip vseh sklicev je MEDIUMINT, torej enak tip kot ga ima primarni ključ v glavni tabeli.

Seveda je potrebno poudariti, da v tabelah za minimum nimamo sklicev za meteorološke parametre kot je hitrost vetra in sunek vetra, saj pač ta dva parametra nista relevantna v tabeli minimumov. Tabele so si po strukturi popolnoma enake, razen v dveh primerih:

- LetoMaksimum in LetoMinimum imata namesto tipa Date (datum) zamenjan z tipom YEAR (Leto).
- VremenskaPostajaMaksimum in VremenskaPostajaMinimum ne vsebujeta dodatnega tipa za datum ali leto. V tabeli se vedno nahaja samo ena n-terica, ki predstavlja maksimum oziroma minimum odkar se merijo podatki na meteorološki postaji.

Torej kot imamo v glavni tabeli sklice na pomožne tabele dejanskih vrednosti, imamo tukaj sedaj sklice na n-terice glavne tabele . Na tem mestu se postavlja vprašanje, zakaj ne shranjujemo kar dejanskih vrednosti v tabelo. Razlogi so sledeči:

- S tem ko imamo sklic na n-terico glavne tabele dobimo poleg sklica na n-terico v pomožni tabeli, ki je maksimum/minimum za določeno časovno obdobje še podatke o ostalih vremenskih parametrih ob tem času.

- Manjše število atributov. Namreč sedaj dobimo implicitno čas ob katerem je bil izmerjen minimum/maksimum iz n-terice, na katero se sklicujemo. Če bi imeli v tabeli dejanske vrednosti minimumov/maksimumov, bi morali shraniti še za vsak podatek datum in čas, torej kdaj je bil ekstrem dosežen.

- Referenčna integriteta statističnega podatka. Ne more obstajati nek ekstrem v podatkovni bazi, ne da bi bil vezan na dejanski zapis v glavni tabeli.

5.1.4 Tabele padavin

Vsak zapis v glavni tabeli vsebuje podatek o relativnih padavinah v izmerjenem intervalu. Želimo pa imeti dodatno izračunane tudi ostale statistične podatke, ki pa jih zaradi specifičnih lastnosti ne moremo vključiti v obstoječe statistične tabele ekstremov in povprečja. Govorimo o naslednjih podatkih (v oklepajih so navedena imena pripadajočih tabel):

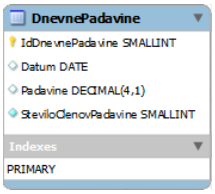
- Dnevne padavin (DnevnePadavine).

- Padavinska poslabšanja (PadavinskaPoslabsanja).

- Mesečne in letne padavine (PadavineMesecLeto).

Tabela DnevnePadavine

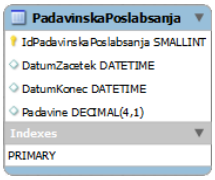
Vsebuje vsoto padavin za vsak dan (slika 5.12). Vsota se računa iz atributa PadavineRelativno iz glavne tabele. Poleg tega imamo še meta podatek o številu členov (število n-teric), ki so bili uporabljeni pri izračunu dnevne vsote količine padavin.



Slika 5.12: Struktura tabele DnevnePadavine.

Tabela PadavinskaPoslabsanja

Vsebuje padavinska poslabšanja (slika 5.13). Padavinsko poslabšanje je definirano kot vsota količine padavin, ki padejo v manj kot 24h zamiku.



Slika 5.13: Struktura tabele PadavinskaPoslabsanja.

Tabela PadavineMesecLeto

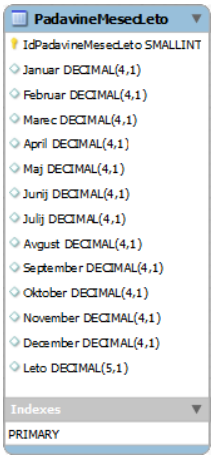
Vsebuje zapise o mesečni in letni količini padavin (slika 5.14). Vsak zapis vsebuje letno količino padavin in količino padavin po mesecih.

5.1.5 Pomožne tabele

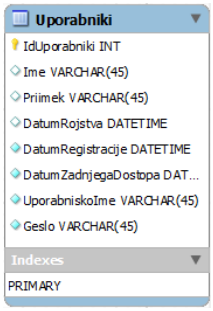
Tabele, ki so namenjene podpori delovanja informacijskega sistema.

Tabela Uporabniki

Vsebuje zapise o uporabnikih (slika 5.15), ki imajo administracijski dostop do spletne strani.



Slika 5.14: Struktura tabele PadavineMesecLeto.



Slika 5.15: Struktura tabele Uporabniki.

5.1.6 Celotna shema podatkovne baze

Shema podatkovne baze vsebuje vse prej naštete tabele predstavljene v kontekstu, torej kakšne so povezave med posameznimi tabelami. Shemo si lahko ogledate v dodatku A, podpoglavje A.3.

5.1.7 Uporaba shranjenih procedur

Shranjene procedure(3.2.2) smo uporabili zaradi naslednjih točk:

- Prestavimo logiko za shranjevanje in posodabljanje v bazo. S tem nismo več vezani na specifičen programski jezik.
- Zmanjšanje režije (overhead) pri izvrševanju velikega števila SQL poi-

zvedb, saj ni potrebno pošiljati SQL ukazov iz aplikacije v SUPB ampak dostopamo do logike, ki je shranjena kot procedure v bazi.

Vsaki tabeli z statističnimi podatki pripada ena procedura, ki je namenjena izračunu ekstremov. Procedure smo razdelili v naslednje sklope:

- Procedure za statistiko.
- Proceduri za padavine.
- Procedura za vzpostavljanje baze.

V shranjenih procedurah za statistiko (glej dodatek A, podpoglavje A.4) se maksimumi računajo enostavno z primerjanjem trenutne vrednosti z trenutnim maksimumom. Minimumi se računajo en enak način, medtem ko povprečje računamo inkrementalno kot je prikazano v enačbi 5.1.

$$\begin{aligned} A_{n+1} = \frac{\sum_{i=1}^{n+1} v_i}{n+1} = \frac{\sum_{i=1}^{n} v_i + v_{n+1}}{n+1} = \frac{nA_n + v_{n+1}}{n+1} \\ = \frac{v_{n+1} + nA_n + A_n - A_n}{n+1} = \frac{v_{n+1} + (n+1)A_n - A_n}{n+1} \\ = A_n + \frac{v_{n+1} - A_n}{n+1} \end{aligned} \tag{5.1}$$

Obstaja še drugačna implementacija. Lahko bi imeli shranjene procedure vezane na vremenske parametre in ne npr. na tabelo minimumov. Toda prednosti shranjene procedure vezane na posamezno tabelo in ne na meteorološki parameter so naslednje:

- Za vsak meteorološki parameter je potrebno imeti zadnjo n-terico iz tabele ekstrema. Tako bi pri proceduri, ki bi bila vezana na določen parameter večkrat poizvedovali po zadnji n-terici iz teh tabel ekstremov, potem pri naslednjem parametru spet iste poizvedbe za zadnje n-terice itd. Če imamo proceduro vezano na tabelo, samo enkrat naredimo poizvedbo na zadnjo n-terico v tej tabeli ekstremov in potem preko te n-terice pridobivam in posodabljam ekstreme. Torej pri n-parametrih prihranimo $n - 1$ poizvedovanj za zadnjo n-terico v tabeli ekstremov.

5.1.8 Uporaba prožilcev podatkovne baze

V naši implementaciji informacijskega sistema se uporablja prožilec podatkovne baze (3.2.2) nad glavno tabelo VremenskiPodatki. Zadolžen je za naslednje stvari:

- Klicanje procedure za vzpostavljanje baze ob praznih tabelah.
- Klicanje procedur za minimume.
- Klicanje procedur za maksimume.
- Klicanje procedur za povprečja.
- Klicanje procedure za dnevne padavine.
- Klicanje procedure za mesečne in letne padavine.

Prožilec se izvrši po vstavljanju novega zapisa v glavno tabelo VremenskiPodatki. Glavna naloga tega prožilca je posodabljanje statistike (izvrševanje poslovnih pravil). Za glavo in izsek kode prožilca za posodabljanje statistike glejte dodatek A, podpoglavje A.5.

5.1.9 Uporaba pogledov podatkovne baze

V naši podatkovni bazi se pogledi (3.2.2) uporabljajo nad tabelami, ki vsebujejo sklice na pomožne tabele. To so naslednje tabele:

- Glavna tabela.
- Tabele statističnih podatkov.

Za glavo in izsek kode pogleda za glavno tabelo glejte dodatek A, podpoglavje A.6.

5.2 Zasnova programske opreme za polnjenje podatkovne baze

Opis zasnove programske opreme smo razdelili na dva dela in sicer branje podatkov iz datoteke ter pisanje prebranih podatkov v bazo.

5.2.1 Branje podatkov

Podatke vremenske postaje zapisuje aplikacija WeatherLink v datoteke tipa .wlk. To je uradna aplikacija vremenske postaje Davis Vantage Pro2. Toda za potrebe integracije informacijskega sistema potrebujemo lastno aplikacijo za zapis podatkov v podatkovno bazo. Na srečo je bila poleg aplikacije WeatherLink priložena tudi shema datotek tipa .wlk, kamor se zapisujejo novi podatki. S pomočjo te sheme lahko razberemo celotno strukturo .wlk datoteke in kako jo učinkovito ter hitro prebrati.

Struktura .wlk datotek

Datoteke .wlk imajo strukturo struct iz programskega jezika C in C++. Tehnične specifikacije natančno določajo strukturo posameznega podatka. Pomembno je zapisati naslednji ugotovitvi za pravilno interpretacijo zapisov:

- Pri zapisu v datoteke se uporablja način zapisa tanki konec (LITTLE ENDIAN), kar pomeni da je teža bajta na zadnjem naslovu najbolj pomembna.
- Uporablja se predznačen dvojiški komplement, kot je prikazan v enačbi 5.2.

$$x = -b_0 2^{N-1} + \sum_{i=1}^{N-2} b_i 2^{N-1-i}, x = [b_0, b_1, \ldots, b_{N-1}], \quad \text{če je } b_0 = 0 \text{ je enako kot nepredznačeno} \tag{5.2}$$

Sledi interpretacija specifikacije:

(a) Glava datoteke (HeaderBlock) (glejte dodatek B, podpoglavje B.1): Glava se nahaja na začetku vsake datoteke. Vsebuje tri različne zapise (v oklepajih so zapisane velikosti zapisov v bajtih):

- idCode: Id kode (16 * 1B = 16B).
- totalRecords: Skupno število vseh zapisov v datoteki (4B). Ta vrednost je enaka vsoti dnevnih zapisov v posameznem dnevnem indeksu (glej formulo 5.3).

$$\sum_{i+1}^{n} DayIndex[i].recordsInDay \tag{5.3}$$

- dayIndex: Indeksni zapis za vsak dan (32 * 6B = 192B).

Skupna velikost glave je tako: $16B + 4B + 192B = 212B$. Podatek o velikosti glave bomo potrebovali za realizacijo programa za branje.

(b) Struktura dnevnega indeksa (DayIndex) (glejte dodatek B, podpoglavje B.2): Na začetku zapisa za vsak dan se nahaja struktura DayIndex. Ta struktura vsebuje dva ključna zapisa za realizacijo programa za branje:

- recordsInDay: Skupno število zapisov v dnevu (2B). Torej če imamo nastavljen interval zapisovanja na 1min je to $0x05a2 = 1442$. Namreč 1440 minut (24h), kar pomeni 1440 zapisov + DnevniPovzetek1 + DnevniPovzetek2.
- startPos: Začetna pozicija prvega zapisa v Dnevnem povzetku (4B). Dejansko je startPos seštevek vseh prejšnjih zapisov recordsInDay (glej enačbo 5.4).

$$DayIndex[n].startPos = \sum_{i}^{n-1} DayIndex[i].recordsInDay \tag{5.4}$$

S pomočjo začetne pozicije lahko sedaj izračunamo absolutni odmik, s katerim postavimo kazalec na DnevniPovzetek1 (glej enačbo 5.5).

$$ABSOLUTNIODMIK = DayIndex[n].startPos * 88 + 212 \tag{5.5}$$

- 88 zato, ker je vsak zapis (DailySummary1, DailySummary2 in WeatherDataRecord) velik 88B.
- 212 zato, ker zavzame HeaderBlock na začetku 212B.

Vsak dan se prične z dvema dnevnima povzetkoma:

(c) Struktura dnevnega povzetka 1 (DailySummary1) (glejte dodatek B, podpoglavje B.3): Vsebuje različne statistične podatke za posamezni dan (minimumi, maksimumi in povprečja). Večina podatkov je predstavljena tako, da so desetinke zapisane tudi kot celo število. Shranjeni so v imperialnih enotah. Velikost zapisa je 88B.

(d) Struktura dnevnega povzetka 2 (DailySummary2) (glejte dodatek B, podpoglavje B.4): Nadaljevanje statističnih podatkov za posamezni dan. Velikost zapisa je 88B.

(e) Struktura dnevnega zapisa (WeatherDataRecord) (glejte dodatek B, podpoglavje B.5): Vsebuje podatke za vsak interval. Velikost celotnega zapisa je 88B.

Implementacija strukture .wlk datoteke

Zaradi potreb po hitrem in enostavnem branju datoteke bajt za bajtom uporabljamo programski vmesnik Javolution (3.2.6). V okviru tega API-ja bomo potrebovali natančneje:

- javolution.io
  - razred Struct

Implementacija strukture je definirana v posebnem projektu z imenom WlkStruktura. Namreč na ta projekt se sklicujem iz več drugih projektov, tako se izognemo podvajanju kode. Vsak razred vsebuje del kode, ki je enak v vseh razredih in določa tanki konec in poravnanost pomnilniških naslovov (glejte dodatek C, podpoglavje C.1). Projekt WlkStruktura vsebuje naslednje Java razrede:

(a) GlavaBlok, ki implementira razred HeaderBlock (glejte dodatek C, podpoglavje C.2).

(b) DanIndeks, ki implementira strukturo DayIndex (glejte dodatek C, podpoglavje C.3).

(c) DnevniPovzetek1, ki implementira strukturo DailySummary1 (glejte dodatek C, podpoglavje C.4).

(d) DnevniPovzetek2, ki implementira DailySummary2 (glejte dodatek C, podpoglavje C.5).

(e) VremenskiPodatkiZapis, ki implementira strukturo WeatherDataRecord (glejte dodatek C, podpoglavje C.6).

Program za branje datoteke

Implemenatacija program se nahaja v projektu WlkBralec. Uporablja prej definirane strukture in tri pomožne razrede:

- Razred PretvorbaPodatkov, ki je namenjen pretvarjanju podatkov iz imperialnega celoštevilskega zapisa v metrični decimalni zapis (glejte dodatek D, podpoglavje D.1).
- Razred VmesnikiZaZrna, ki je namenjen pridobivanju oddaljenih vmesnikov iz aplikacijskega strežnika(glejte dodatek D, podpoglavje D.2). Za pridobivanje oddaljenih vmesnikov uporabimo JNDI poizvedbo (3.2.1).
- Razred PrenosPodatkov, ki je namenjen prenosu podatkov na aplikacijski strežnik v obliki objektov (glejte dodatek D, podpoglavje D.3).

Program za branje .wlk datotek deluje na naslednji način:

(a) Pridobi oddaljene vmesnike iz aplikacijskega strežnika.

(b) S pomočjo oddaljenega vmesnika pridobi instanco zadnjega zapisa iz baze.

(c) Zaporedoma bere datoteke z mesečnimi podatki. Za branje datotek uporablja medpomnilnik za bajte in zgoraj definirane strukture. Ko pride do podatka, katerega datum je večji od datuma zadnjega zapisa v podatkovni bazi, takrat zapiše podatke v instanco razreda za prenos podatkov (D.3) na aplikacijski strežnik.

Ob končanju program zaspi za 10 minut. Nato spet prične brati datoteke za novimi podatki. Torej celoten program je ovit v neskončno zanko.

5.2.2 Zapisovanje podatkov

Ob vsakem novem zapisu se instanca tega zapisa pošlje na aplikacijski strežnik za nadaljnjo obdelavo. Na aplikacijskem strežniku imamo zrno ObdelavaNovihPodatkovZrno (glejte dodatek E, podpoglavje E.1), ki skrbi za zapisovanje novih podatkov v podatkovno bazo. Tu imamo tudi logiko za dodajanje novih zapisov ob novem dnevu/mesecu/letu v tabele statistik in računanje podatkov, katerih logike ni mogoče učinkovito prestaviti na SUPB kot je računanje novega padavinskega poslabšanja, 1h padavinski maksimum ter 24h padavinski maksimum. Tako se ob vsakem zapisu nove n-terice v glavno tabelo VremenskiPodatki s pomočjo prožilca sprožijo shranjene procedure, ki za ekstreme preverijo ali je bil dosežen nov ekstrem, oziroma izračunajo inkrementalno novo povprečje.

5.3 Zasnova informacijskega sistema na aplikacijskem strežniku

Celoten informacijski sistem je zapakiran v projekt tipa EAR, ki ima naslednjo strukturo/module:

- VremenskaPostajaEJB
- VremenskaPostajaEJBClient
- VremenskaPostajaJPA

- VremenskaPostajaJSF
- WlkStruktura

Deljenje sistema na module nam omogoča ponovno uporabo napisane logike, torej izognemo se podvajanju kode. Večja je tudi preglednost implementacije sistema kar pomeni, da je lažje vzdrževanje in dodajanje novih funkcionalnosti. V naslednjih odstavkih bomo opisali posamezne module in njihov namen.

5.3.1 Modul VremenskaPostajaEJB

Vsebuje implementacijo lokalnih in oddaljenih vmesnikov EJB zrn (3.2.1). Vsako zrno implementira logiko za določeno entiteto oziroma tabelo. Vsa zrna v naši implementaciji ne ohranjajo stanja, torej so tipa Stateless.

Kot smo že omenili je potrebno zrno označiti kot brez stanja. To naredimo z anotacijo @Stateless. V glavi razreda implementiramo lokalne in oddaljene vmesnike. V telesu razreda najprej vstavimo trajnostni kontekst za upravljavca entitet. Z upravljavcem entitet dostopamo do trajnostnega konteksta, ki vsebuje množico upravljanih entitet. Nato ima vsako zrno implementirane standardne CRUD operacije (glejte dodatek F, podpoglavje F.1). Ostale metode, ki jih implementira posamezno zrno so vezane na specifično funkcijo zrna, npr. RosisceZrno implementira metode za izračun rosišča iz podatka temperature zraka in relativne vlage (glejte dodatek F, podpoglavje F.1).

Če na kratko povzamemo, prvi del zrna vsebuje standardne metode, drugi del zrna pa vsebuje metode, ki so vezane na namen zrna.

5.3.2 Modul VremenskaPostajaEJBClient

Vsebuje lokalne in oddaljene vmesnike za opisana zrna v prejšnjem modulu. Z njimi predpišemo metode, ki jih mora implementirati nek razred. Lokalne in oddaljene vmesnike uporabljamo, ker s tem omogočimo dostop do metod v

zrnu preko lokalnega ali oddaljenega nivoja. Oddaljene vmesnike uporabimo takrat, kadar želimo dostopati do metod zrna izven JVM, v kateri se izvaja aplikacijski strežnik.

Implementacija vmesnikov je zelo enostavna. Če je vmesnik lokalni, ga anotiramo z @Local, če je oddaljeni pa z @Remote. S tem opisno povemo, za kakšen tip vmesnika gre. Nato v telesu napišemo glave metod, ki jih mora zrno implementirati. V našem primeru imajo lokalni vmesniki veliko večji nabor metod, medtem ko oddaljeni vmesniki definirajo majhno število metod. Razlog je v tem, da želimo izpostaviti za oddaljeni dostop samo nekatere metode. Recimo v našem primeru uporabimo oddaljeni vmesnik pri branju novih podatkov iz datoteke. V programu potrebujemo oddaljeni vmesnik za pridobivanje zadnje n-terice v glavni tabeli VremenskiPodatki. Za primer implementacije glejte dodatek F, podpoglavje F.2 in F.3. Medtem ko lokalne vmesnike uporabljamo uporabljamo kjerkoli potrebujemo dostop do metod v zrnu v okviru aplikacijskega strežnika.

5.3.3 Modul VremenskaPostajaJPA

Vsebuje entitete (Javanski razredi), ki so preslikave relacijskih tabel. Tako lahko vsako n-terico v relacijski tabeli preslikamo v objektno predstavitev oziroma vsaki relacijski tabeli pripada svoja entiteta. Za preslikavo uporabljamo programski vmesnik JPA (3.2.1).

Uporaba objektno-relacijske preslikave nam v našem primeru olajša pridobivanje dejanskih podatkov. Namreč zaradi pogoste uporabe tujih ključev bi bila uporaba običajnega JDBC veliko težje opravilo. Poleg tega je vsak objekt upravljan v trajnostnem kontekstu, kar pomeni da imamo natančen pregled življenjskih faz objekta. To nam omogoča enostavno upravljanje in nam veliko pripomore pri razumevanju stanja informacijskega sistema.

Implementacija entitet je suhoparna, saj je potrebno definirati precej stvari za pravilno preslikavo. Najprej je potrebno anotirati razred z @Entity, s čimer opisno določimo, da gre za entiteto. Nato določimo statične JPQL poizvedbe, ki omogočajo hitrejše izvrševanje poizvedb. V naši implementaciji

jih uporabljamo za naslednje poizvedbe:

- Pridobivanje zadnjega zapisa iz statističnih tabel.
- Pridobivanje objekta glede na vrednost iz tabel z zalogo vrednosti.
- Pridobivanje seznama objektov za zadnjih 24h.

Zgoraj našteti primeri statičnih poizvedb so uporabni predvsem pri pridobivanju podatkov za izpis uporabniku, kjer ne želimo, da bi moral uporabnik predolgo čakati. Poleg anotacije @Entity je potrebno podati tudi anotacijo @Table z atributom name, s katerim povemo, katera je primarna tabela entitete. Vsaka razred mora tudi implementirati vmesnik Serializable. Namreč če želimo instanco entitete poslati po vrednosti kot odklopljen objekt od trajnostnega konteksta, potem je potrebno da razred implementira vmesnik Serializable. In nazadnje pride definiranje naslednjih delov, ki so bistvo preslikave:

- Atributi in pripadajoči podatkovni tipi, kamor se preslikajo atributi tabel.
- Metode, ki nastavljajo vrednosti atributov (setter).
- Metode, ki vračajo vrednosti atributov (getter).

Opcijsko smo dodali v naše entitete @XML anotacije za preslikavo instanc v XML zapis. To nam pride prav pri spletnih storitvah, kjer je potrebno za XML format nekako preslikati objekt. Za primer implementacije entitete na primeru entitete VlagaZunanja glejte dodatek F, podpoglavje F.4.

5.3.4 Modul VremenskaPostajaJSF

Za izgradnjo spletnih aplikacij in spletnih strani uporabljamo tehnologijo JSF (3.2.1). V nadaljnjem opisu bomo predstavili sklope modula JSF, ki podpirajo delovanje spletnih strani.

Konverterji

Konverter razrede uporabimo za lastno serializacijo objektov v JSON notacijo/JavaScript polje z uporabo Gson knjižnice (3.2.6). Primer uporabe konverterja za kreiranje JavaScript polja, ki vsebuje podatke zunanje temperature zraka si oglejte pod dodatek F, podpoglavje F.5. Večina konverterjev je v našem primeru namenjena kreiranju Javascript polja, ki se uporabi kot vhod za izris grafa. Ostali se uporabljajo za potrebe spletnih storitev.

Kontrolerji

Paket kontrolerji vsebuje podporna zrna za spletne strani. Podporna zrna vsebujejo logiko za pridobivanje podatkov s pomočjo lokalnih vmesnikov EJB zrn. Na metode in lastnosti posameznega kontrolerja se lahko sklicujemo iz XHTML spletnih strani s pomočjo EL jezika. Paket vsebuje naslednje kontrolerje:

- ArhivKontroler
- TrenutniPodatkiKontroler

TrenutniPodatkiKontroler TrenutniPodatkiKontroler je namenjen pridobivanju potrebnih podatkov za spletno stran trenutni podatki. Lokalne vmesnike EJB zrn dobimo z vstavljanjem odvisnosti na lokalne vmesnike (glejte dodatek F, podpoglavje F.6). Vstavljanje odvisnosti je podprto z tehnologijo CDI (3.2.1). Ob kreiranju strani (incializacija) pridobimo instance za različne meteorološke podatke v zadnjih 24h. Podatki za izris grafov se za potrebe prenosa k odjemalcu serializirajo iz List<VremenskiPodatki>v Javascript polje. Polje se doda v povratni parameter konteksta zahtevka. Namreč podatke za izris grafa pridobivamo z AJAX tehnologijo. Ostale metode so namenjene za podporo delovanja prikaza trenutnih podatkov.

Spletne storitve

Informacijski sistem nudi spletne storitve za naslednje podatke:

1. Trenutni podatki
2. Dnevni maksimumi
3. Dnevni minimumi
4. Dnevna povprečja
5. Mesečni maksimumi
6. Mesečni minimumi
7. Mesečna povprečja
8. Letni maksimumi
9. Letni minimumi
10. Letna povprečja
11. Vremenska postaja maksimumi
12. Vremenska postaja minimumi
13. Vremenska postaja povprečja

V naslednjih oblikah:

- TXT
- HTML
- XML
- JSON

Spletne storitve so tipa RESTful. Recimo za dostop do spletnih storitev za trenutne podatke v XML obliki uporabnik dostopa preko naslova `http://www.vp-zalec.net/storitve/trenutniPodatkiXML`.

Trenutni podatki Najprej je potrebno z anotacijo deklarirati pot do storitve ter vstaviti odvisnosti na lokalna zrna za meteorološke podatke (glejte dodatek F, podpoglavje F.7). V naslednjih odstavkih so navedene formati spletnih storitev in njihova implementacija:

(a) Tekstovna implementacija (glejte dodatek F, podpoglavje F.7). Z anotacijo definiramo pot, kjer se nahaja ta storitev in kakšen tip proizvede.

(b) HTML implementacija (glejte dodatek F, podpoglavje F.7).

(c) XML implementacija (glejte dodatek F, podpoglavje F.7). Kot lahko vidimo je XML implementacija enostavnejša, saj uporabljamo za kreiranje XML datoteke kar entitete. Kot smo lahko videli pri entitetah imamo anotacije za kreiranje XML.

(d) JSON implementacija (glejte dodatek F, podpoglavje F.7). Za JSON implementacijo uporabljamo knjižnico Gson (3.2.6) in lastni konverter, kjer implementiramo način serializacije.

5.3.5 Spletne strani

Implementacije spletnih strani, pripadajočih ogrodij, stilov in javascript knjižnic se nahaja v imeniku WebContent. Struktura WebContent je naslednja:

- Pages: vsebuje glavne spletne strani
- Resources: viri spletni strani
  - Css: stili spletnih strani
  - Fonts: fonti spletnih strani
  - Images: Slike spletnih strani
  - Javascript: javascript knjižnice spletnih strani
- WEB-INF:
  - IncludeFiles: komponente spletni strani

- Lib: jar knjižnice
- Templates: ogrodje spletne strani

Ogrodje spletne strani

Ogrodje ima definirane privzete knjižnice in stile, ki se uporabljajo v vseh spletnih straneh. Definiramo tudi glavo, meni, vsebino in nogo (glejte dodatek F, podpoglavje F.8) Vsebino posameznih strani bomo vključevali v vsebovalnik z imenom vsebina.

Uporaba ogrodja

Ogrodje uporabimo z vstavljanjem <ui:composition template=>(glejte dodatek F, podpoglavje F.8).

Poglavje 6

Primeri uporabe informacijskega sistema

Del informacijskega sistema je namenjen tudi navadnim uporabnikom oziroma odjemalcem. Ti uporabniki imajo dostop do naslednjih podatkov:

- Trenutni vremenski podatki.
- Ekstremi in povprečja za posamezna časovna obdobja.
- Vremenski podatki zadnjih 24h.
- Spletne storitve.

V naslednjih sklopih bomo predstavili primere uporabe za zgoraj naštete možnosti, kakšno dodano vrednost prinesejo odjemalcu in na kakšen način lahko dostopa do teh podatkov.

6.1 Trenutni vremenski podatki

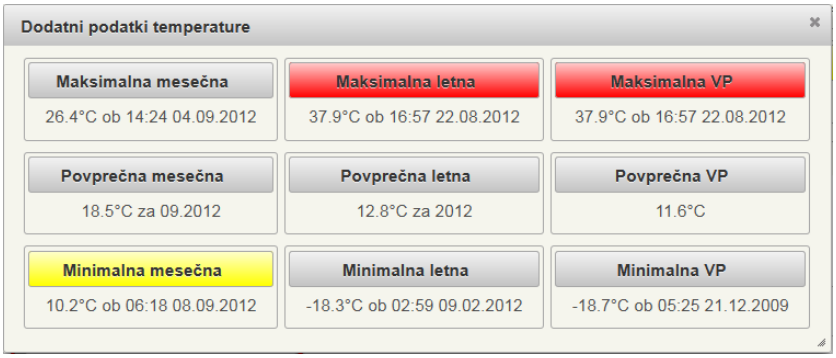
Odjemalec dostopa do trenutnih vremenskih podatkov preko spletnega brskalnika. Dostopni so na naslednjem spletnem naslovu: `http://www.vp-zalec.net/podatki`. Predstavljeni so v tekstovni obliki in njihovo ozadje se obarva glede na enakost kateremu od doseženih ekstremov za posamezno časovno

obdobje. Odjemalec lahko z njihovo uporabo interpretira trenutne vremenske razmere. Na primer če je temperatura $< 0°C$ in je relativna vlažnost visoka potem lahko uporabnik sklepa, da je velika verjetnost da je na cesti poledica in bo zaradi tega moral biti previden. Prav tako mu temperatura zraka sporoča, kako se naj obleče (slika 6.1 prikazuje predstavitev zunanje temperature). Iz parametra temperatura rosišča lahko izkušenejši uporabnik ugotovi, kakšne so pogoji za nastanek neviht. Višja rosišča velikokrat pomenijo boljše pogoje za nastanek močnejših neviht. Parameter občutek mraza, ki je izračunan na podlagi temperature zraka in hitrosti vetra pove uporabniku, da se je potrebno ob občutni razliki med temperaturo zraka in občutkom mraza bolje obleči. Iz trenutne količine relativnih padavin lahko vidi, kako močno dejansko dežuje.

Slika 6.1: Prikaz trenutnih podatkov.

6.2 Ekstremi in povprečja za posamezna časovna obdobja

Zraven trenutnih podatkov so zapisani ekstremi in povprečja za posamezna časovna obdobja. Uporabniku so sprva vidni samo dnevni maksimum, dnevni minimum in dnevno povprečje. S temi podatki lahko uporabnik na primer ugotovi, ali je zjutraj že bila temperatura pod 0°C, ali dnevni maksimum že presega 30°C, kakšni so bili najmočnejši sunki vetra. Ekstremi za ostala časovna obdobja, torej mesec, leto in vremenske postaje pa so dostopni preko klika na meni ->dodatni podatki. Kako so ekstremi predstavljeni lahko vidite na sliki 6.2.



Slika 6.2: Prikaz dodatnih ekstremov temperature zraka.

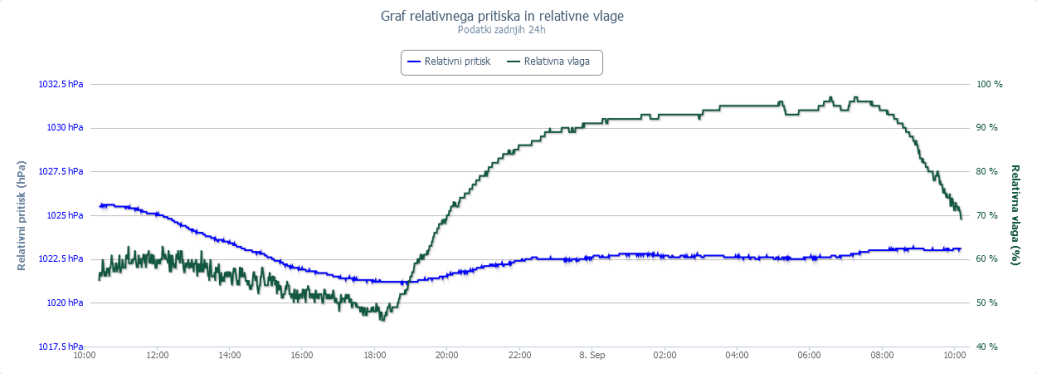
6.3 Vremenski podatki zadnjih 24h

Odjemalcu se pri dostopu do trenutnih vremenskih podatkov izrišejo tudi grafi z podatki za zadnjih 24h. Imamo skupaj 6 grafov in sicer:

- Graf temperature, rosišča in občutka mraza.
- Graf relativnega pritiska in relativne vlage.
- Graf količine padavin.

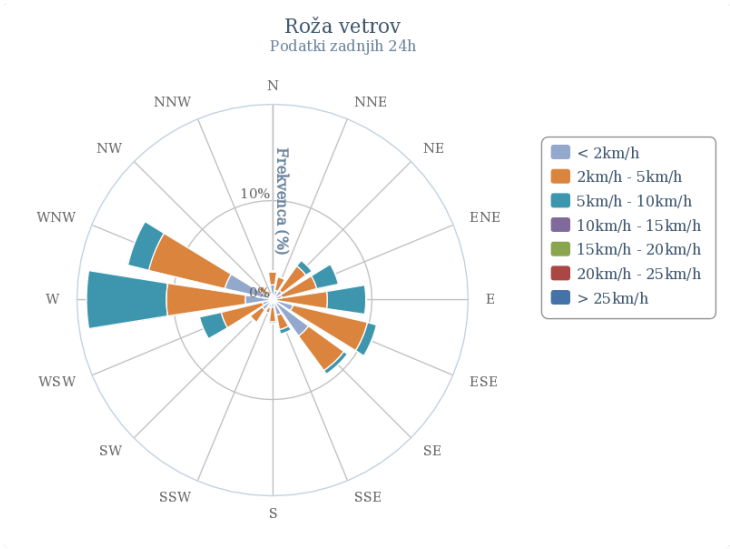
- Graf hitrosti vetra in sunka vetra.
- Graf smeri vetra.
- Graf rože vetrov.

Vsi grafi so interaktivni. Ob prehodu miške skozi posamezna časovna obdobja se nam prikaže podatek kakšno vrednost je imel izbrani meteorološki parameter ob določenem času. Poleg tega ima uporabnik možnost z miško večkrat povečati posamezne odseke grafa. Uporabnik lahko iz grafov prebere, kako se obnaša posamezni vremenski parameter, torej iz grafa lahko sklepa ali bo relativni pritisk še naprej padal, s kakšno intenziteto oziroma ali bo začel naraščati (glej sliko 6.3). To pomeni, da lahko iz grafa relativnega pritiska razberemo ali se nam bliža oziroma nas prehaja ciklon.



Slika 6.3: Graf relativnega pritiska in relativne vlage zadnjih 24h.

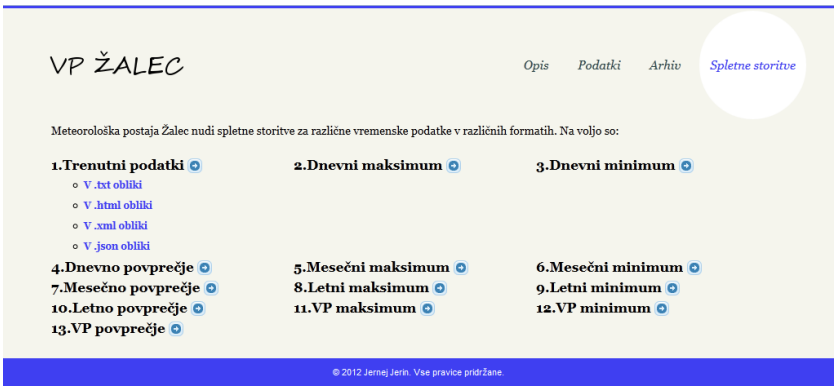
Eden izmed bolj zanimivih grafov, ki jih ponuja spletna stran je graf rože vetrov (glej sliko 6.4), ki predstavlja porazdelitev vetra glede na hitrost in smer. Imamo sedem različnih intervalov hitrosti in 16 različnih smeri. Podatke grupiramo po intervalu hitrosti, ter nato še po smeri neba. Graf predstavlja frekvenco grup in ima veliko dodano vrednost pri analiziranju razmer za morebitno postavitev vetrnih elektrarn.



Slika 6.4: Graf roža vetrov zadnjih 24h.

6.4 Spletne storitve

Spletne storitve so namenjene človeškemu kot tudi programskemu dostopu do podatkov. Nahajajo se na spletnem naslovu `http://www.vp-zalec.net/storitve`. Uporabnik ima na voljo spletne storitve za trenutne podatke in statistične podatke za posamezna časovna obdobja. Na voljo so kot interaktivni seznam, ki odpira seznam različnih formatov izbrane storitve in zapre ostale storitve (prikaz seznama in formatov na sliki 6.5). Uporaba storitev veščemu uporabniku omogoča enostaven programsko prebiranje podatkov.



Slika 6.5: Seznam formatov za spletno storitev trenutni podatki.

Poglavje 7

Sklepne ugotovitve

V diplomski nalogi smo integrirali informacijski sistem z napravami oziroma senzorji za merjenje meteoroloških parametrov. Naš prispevek je obsegal izdelavo podatkovnega modela, program za branje podatkov in logiko na AS za pisanje, obdelavo in prikaz podatkov.

Brez realizacije zgoraj naštetih stvari informacijski sistem ne bi deloval. Vsak del je enako pomemben za njegovo delovanje, torej od branja podatkov, zapisovanja, obdelave in prikaza. Sistem omogoča zaradi modularne izgradnje enostavno dodajanje novih funkcionalnosti in vzdrževanja obstoječih. Prav tako je veliko možnosti uporabe takšnega sistema, nekaj smo jih že našteli tudi v poglavju primeri uporabe.

Ostaja pa veliko prostora pri izboljšanju realizacije informacijskega sistema. Predvsem bi poudaril napačno odločitev za uporabo podatkovnega tipa BigDecimal namesto primitivnega tipa int, kjer bi prihranili kar nekaj prostora. Poleg tega bi precej pohitrili del sistema za pridobivanje podatkov. Naslednja stvar je manjše drobljenje pri podatkovnem modelu, oziroma manjša uporaba tujih ključev, saj si tako prihranimo kar nekaj dela in pri tem precej poenostavimo podatkovni model kot tudi logiko za dostop do podatkov na AS.

Na koncu naj povemo, da se razviti sistem trenutno že uporablja za privatno vremensko postajo Žalec, oziroma uporabniki lahko dostopajo do storitev

sistema preko spletne strani `http://www.vp-zalec.net`.

Literatura

[1] E. Jendrock, I. Evans, D. Gollapudi, K. Haase, C. Srivathsa, *The Java EE 6 Tutorial: Basic Concepts*. Upper Saddle River: Addison-Wesley, 2011.

[2] J. Cardoso, K. Voigt, M. Winkler, "Service Engineering for the Internet of Services", v *Enterprise Information Systems*, št. 1, zv. 19, str. 15-27, 2009. Dostopno na: `http://eden.dei.uc.pt/~jcardoso/Research/Papers/LNBI-ICEIS-2009-ServiceEngineering-for-the-IoS.pdf`

[3] M. Bajec, "Relacijski podatkovni model", 2012. Dostopno na: `http://bajecm.fri.uni-lj.si/pg_opb%20BOLOGNA.htm`

[4] M. Ciglarič, "Spletne storitve", 2012. Dostopno na: `http://ucilnica.fri.uni-lj.si/mod/resource/view.php?id=12624`

[5] R. C. Johnson, "The intangible assets of the Internet of Things", 2012. Dostopno na: `http://www.eetimes.com/electronics-news/4371064/The-intangible-assets-of-the-Internet-of-Things`

[6] M. B. Jurič, "Servleti", 2012. Dostopno na: `http://ucilnica.fri.uni-lj.si/mod/resource/view.php?id=12331`

[7] M. B. Jurič, "Java Persistence API", 2012. Dostopno na: `http://ucilnica.fri.uni-lj.si/mod/resource/view.php?id=12619`

[8] M. B. Jurič, "EJB - JAVANSKA STREŽNIŠKA ZRNA", 2012. Dostopno na: http://ucilnica.fri.uni-lj.si/mod/resource/view.php?id=13500

[9] M. McConnell, "The Rise of the Internet of Things", 2012. Dostopno na: http://blog.xo.com/industry-trends/the-rise-of-the-internet-of-things/

[10] R. Saracco, "IoT, and IwT", 2012. Dostopno na: http://www.blog.telecomfuturecentre.it/2012/05/21/iot-and-iwt/

[11] (2012) Internet of Things: what is it?. Dostopno na: http://www.theinternetofthings.eu/internet-of-things-what-is-it%3F

[12] (2012) Eclipse. Dostopno na: http://www.eclipse.org/

[13] (2012) Gson. Dostopno na: https://sites.google.com/site/gson/Home

[14] (2012) Highcharts. Dostopno na: http://www.highcharts.com/

[15] (2012) Internet of Services. Dostopno na: http://en.wikipedia.org/wiki/Internet_of_Services

[16] (2012) Internet of Things. Dostopno na: http://en.wikipedia.org/wiki/Internet_of_Things

[17] (2012) Javolution. Dostopno na: http://javolution.org/

[18] (2012) JBoss AS 7. Dostopno na: http://www.jboss.org/as7

[19] (2012) JBoss Hibernate. Dostopno na: http://www.hibernate.org/

[20] (2012) JBoss Tools. Dostopno na: http://www.jboss.org/tools

[21] (2012) Joda Time. Dostopno na: http://joda-time.sourceforge.net/

[22] (2012) JScience. Dostopno na: `http://jscience.org/`

[23] (2012) jQuery. Dostopno na: `http://jquery.com/`

[24] (2012) JQuery UI. Dostopno na: `http://jqueryui.com/`

[25] (2012) Multitier architecture. Dostopno na: `http://en.wikipedia.org/wiki/Multitier_architecture`

[26] (2012) MySQL. Dostopno na: `http://www.mysql.com/`

[27] (2012) MySQL Workbench. Dostopno na: `http://www.mysql.com/products/workbench/`

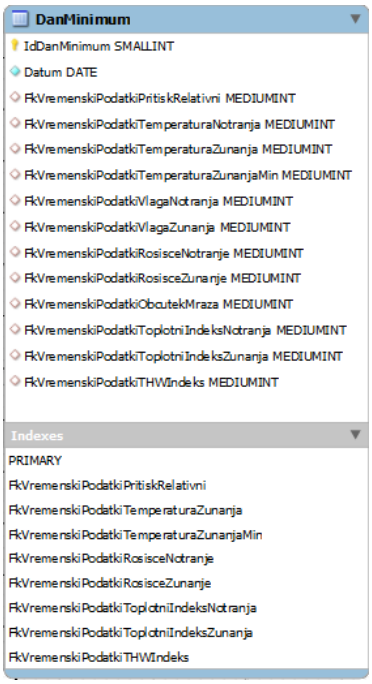
[28] (2012) Oracle Mojarra JavaServer Faces. Dostopno na: `http://javaserverfaces.java.net/`

[29] (2012) PrimeFaces. Dostopno na: `http://primefaces.org/`

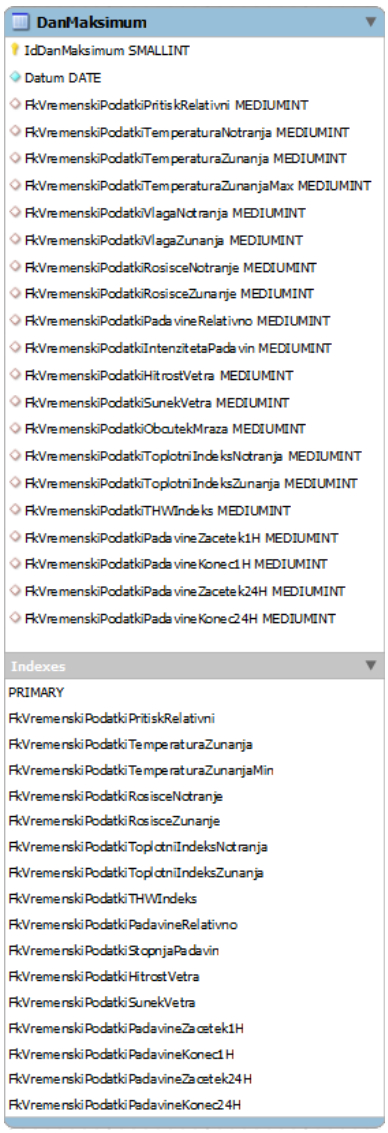
Dodatek A

Podatkovni model

A.1 Struktura statističnih tabel

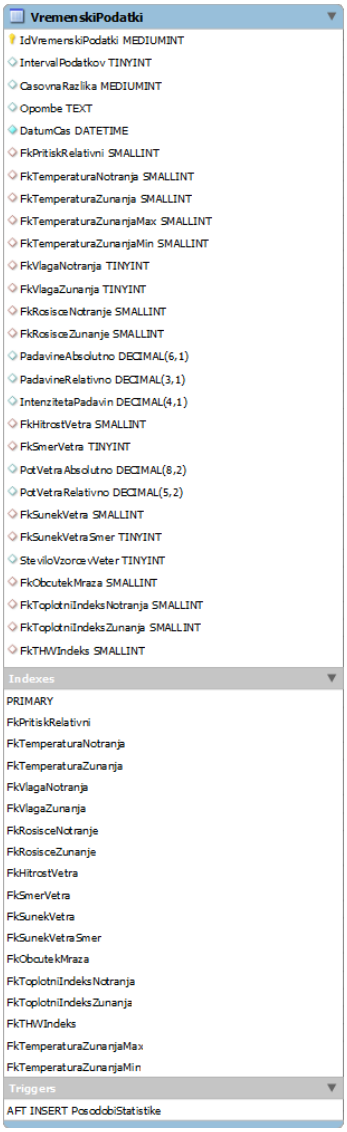


Slika A.1: Struktura tabele DanMinimum.



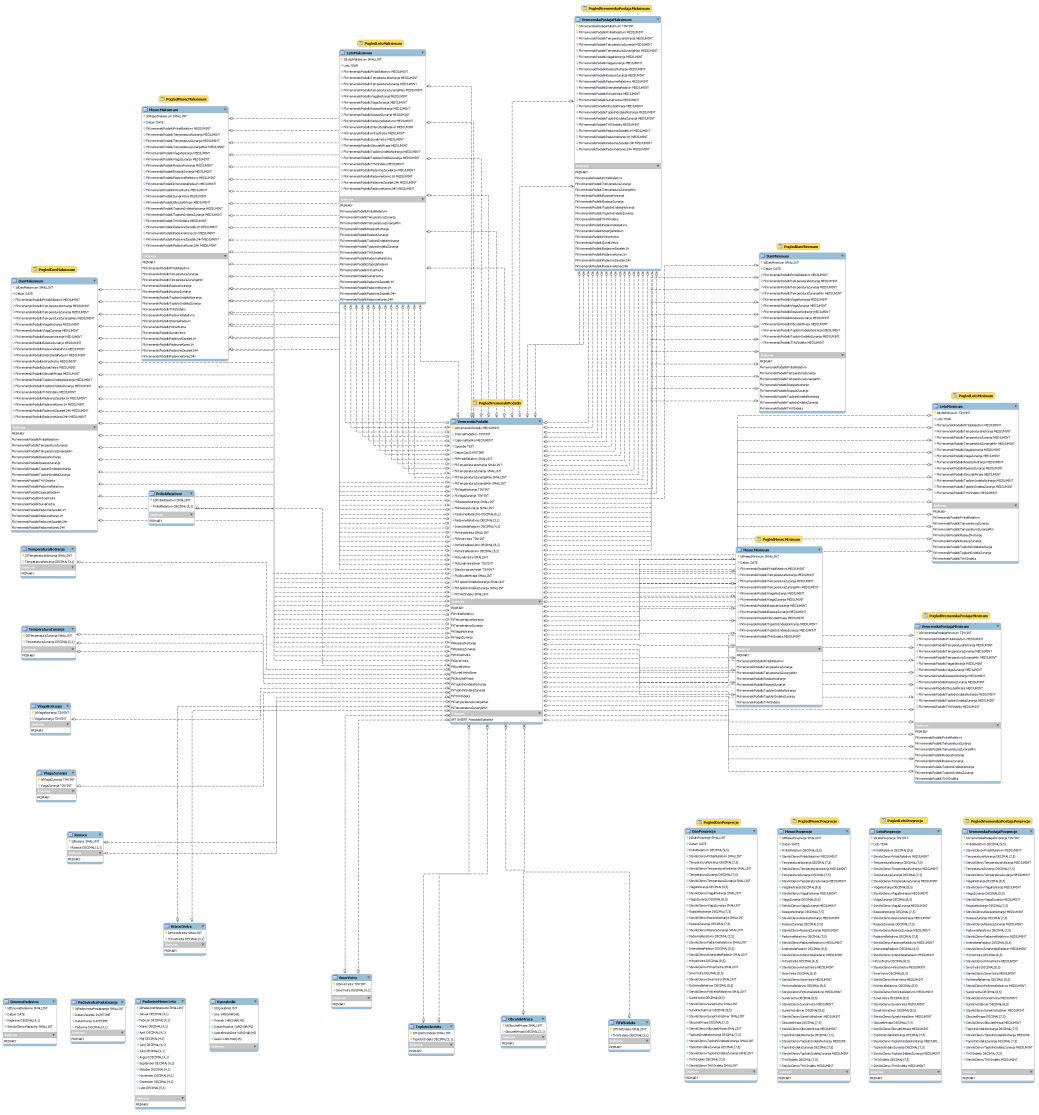
Slika A.2: Struktura tabele DanMaksimum.

A.2 Struktura glavne tabele



Slika A.3: Struktura glavne tabele VremenskiPodatki.

A.3 Shema podatkovnega modela



Slika A.4: Shema podatkovnega modela.

A.4 Shranjena procedura

```
CREATE PROCEDURE `VremeBaza`.`ProceduraDanMaksimum` (IN
    zadnjaNtericaPodatki MEDIUMINT)
COMMENT 'Procedura za posodabljanje dnevnih maksimumov vseh
    vremenskih parametrov.'
BEGIN
    /* Najprej deklariramo spremenljivke, ki jih bomo
       potrebovali v shranjeni proceduri. */
    DECLARE trenutniMaksimum SMALLINT;
    DECLARE zadnjaNterica SMALLINT;
    DECLARE trenutnaVrednost SMALLINT;
    DECLARE sklicMaksimum MEDIUMINT;

    /* Pridobimo id zadnje n-terice v tabeli dnevnih maksimumov.
       */
    SET zadnjaNterica = (SELECT MAX(IdDanMaksimum) FROM
        DanMaksimum);

    /* RELATIVNI PRITISK */
    /* Shranimo sklic na glavno n-terico. */
    SET sklicMaksimum = (SELECT
        FkVremenskiPodatkiPritiskRelativni FROM DanMaksimum WHERE
         IdDanMaksimum = zadnjaNterica);

    /* Potem pridobimo trenutno vrednost iz glavne tabele. */
    SET trenutnaVrednost = (SELECT FkPritiskRelativni FROM
        VremenskiPodatki WHERE IdVremenskiPodatki =
        zadnjaNtericaPodatki);

    /* Ce je sklic na maksimum NULL in ce je sklic na dejansko
        vrednost v glavni tabeli razlicen od NULL, potem
        nastavimo kar na trenutno vrednost. */
    IF sklicMaksimum IS NULL AND trenutnaVrednost IS NOT NULL
       THEN
        UPDATE DanMaksimum SET
            FkVremenskiPodatkiPritiskRelativni =
            zadnjaNtericaPodatki WHERE IdDanMaksimum =
```

```
            zadnjaNterica;
    END IF;
    /* Ce tako sklic kot trenutna vrednost nista NULL, potem
       lahko preverimo za novim maksimumom. */
    IF sklicMaksimum IS NOT NULL AND trenutnaVrednost IS NOT
       NULL THEN
        /* Najprej pridobimo trenutni maksimum. */
        SET trenutniMaksimum = (SELECT FkPritiskRelativni FROM
            VremenskiPodatki WHERE IdVremenskiPodatki =
            sklicMaksimum);

        /* Preverimo ali je bil dosezen nov maksimum. */
        IF trenutnaVrednost >= trenutniMaksimum THEN
            UPDATE DanMaksimum
            SET FkVremenskiPodatkiPritiskRelativni =
                zadnjaNtericaPodatki
            WHERE IdDanMaksimum = zadnjaNterica;
        END IF;
    END IF;
  ...
END
```

Listing A.1: Primer odseka shranjene procedure za izračun maksimuma.

A.5 Prožilec podatkovne baze

```
DELIMITER $$
USE `VremeBaza`$$
CREATE TRIGGER `VremeBaza`.`PosodobiStatistike`
AFTER INSERT ON `VremenskiPodatki` FOR EACH ROW
    BEGIN
        /* Preverimo ali gre za prvi vnos. Ce gre, klicemo
           proceduro, ki nam kreira n-terice v tabelah za
           statistiko. */
        IF NEW.IdVremenskiPodatki = 1 THEN
            CALL ProceduraVzpostavljanjeBaze(1);
        END IF;
```

```
11          /* PROCEDURE ZA MINIMUME */
            /* Klicemo procedure za tabele minimumov. Kot parameter
               podamo id, da nam ga ni potrebno pridobivati v
               procedurah. */
13          CALL ProceduraDanMinimum(NEW.IdVremenskiPodatki);
            CALL ProceduraMesecMinimum(NEW.IdVremenskiPodatki);
15          CALL ProceduraLetoMinimum(NEW.IdVremenskiPodatki);
            CALL ProceduraVremenskaPostajaMinimum(NEW.
               IdVremenskiPodatki);
17
            /* PROCEDURE ZA MAKSIMUME */
19          CALL ProceduraDanMaksimum(NEW.IdVremenskiPodatki);
            CALL ProceduraMesecMaksimum(NEW.IdVremenskiPodatki);
21          CALL ProceduraLetoMaksimum(NEW.IdVremenskiPodatki);
            CALL ProceduraVremenskaPostajaMaksimum(NEW.
               IdVremenskiPodatki);
23
            /* PROCEDURE ZA POVPRECJA */
25          CALL ProceduraDanPovprecje(NEW.IdVremenskiPodatki);
            CALL ProceduraMesecPovprecje(NEW.IdVremenskiPodatki);
27          CALL ProceduraLetoPovprecje(NEW.IdVremenskiPodatki);
            CALL ProceduraVremenskaPostajaPovprecje(NEW.
               IdVremenskiPodatki);
29
            /* PROCEDURA ZA DNEVNE PADAVINE */
31          CALL ProceduraDnevnePadavine(NEW.IdVremenskiPodatki);

33          /* PROCEDURA ZA MESECNE IN LETNE PADAVINE */
            CALL ProceduraPadavineMesecLeto(NEW.IdVremenskiPodatki);
35      END
    $$
```

Listing A.2: Primer odseka prožilca

A.6 Pogled podatkovne baze

```
CREATE VIEW `VremeBaza`.`PogledVremenskiPodatki` AS
SELECT `VP`.`IdVremenskiPodatki`, `VP`.`IntervalPodatkov`, `VP
    `.`CasovnaRazlika`, `VP`.`Opombe`, `VP`.`DatumCas`,`PR`.`
    PritiskRelativni`, `TN`.`TemperaturaNotranja`,
`TZ`.`TemperaturaZunanja`, `TZMax`.`TemperaturaZunanja` AS `
    TemperaturaZunanjaMax`, `TZMin`.`TemperaturaZunanja` AS `
    TemperaturaZunanjaMin`, `VN`.`VlagaNotranja`,
`VZ`.`VlagaZunanja`
...
FROM `VremenskiPodatki` `VP`
LEFT JOIN `PritiskRelativni` `PR`
ON `VP`.`FkPritiskRelativni` = `PR`.`IdPritiskRelativni`
LEFT JOIN `TemperaturaNotranja` `TN`
ON `VP`.`FkTemperaturaNotranja` = `TN`.`IdTemperaturaNotranja`
LEFT JOIN `TemperaturaZunanja` `TZ`
ON `VP`.`FkTemperaturaZunanja` = `TZ`.`IdTemperaturaZunanja`
LEFT JOIN `TemperaturaZunanja` `TZMax`
ON `VP`.`FkTemperaturaZunanjaMax` = `TZMax`.`
    IdTemperaturaZunanja`
LEFT JOIN `TemperaturaZunanja` `TZMin`
ON `VP`.`FkTemperaturaZunanjaMin` = `TZMin`.`
    IdTemperaturaZunanja`
LEFT JOIN `VlagaNotranja` `VN`
ON `VP`.`FkVlagaNotranja` = `VN`.`IdVlagaNotranja`
LEFT JOIN `VlagaZunanja` `VZ`
ON `VP`.`FkVlagaZunanja` = `VZ`.`IdVlagaZunanja`
...
ORDER BY `DatumCas` ASC
```

Listing A.3: Primer odseka pogleda.

Dodatek B

Specifikacije

B.1 Glava datoteke

```
class HeaderBlock
{
   char idCode[16]; // = {'W', 'D', 'A', 'T', '5', '.', '0', 0,
       0, 0, 0, 0, 0, 0, 5, 0}
   long totalRecords;
   DayIndex dayIndex[32]; // index records for each day. Index
      0 is not used
                          // (i.e. the 1'st is at index 1, not
                             index 0)
};
```

Listing B.1: Razred glave datoteke

B.2 Dnevni indeks

```
struct DayIndex
{
   short recordsInDay;  // includes any daily summary records
   long startPos;     // The index (starting at 0) of the first
      daily summary record
};
```

Listing B.2: Razred dnevnega indeksa

B.3 Dnevni povzetek 1

```
struct DailySummary1
{
   BYTE dataType = 2;
   BYTE reserved;    // this will cause the rest of the fields
       to start on an even address

   short dataSpan;   // total # of minutes accounted for by
       physical records for this day
   short hiOutTemp, lowOutTemp; // tenths of a degree F
   short hiInTemp, lowInTemp;   // tenths of a degree F
   short avgOutTemp, avgInTemp; // tenths of a degree F (
       integrated over the day)
   short hiChill, lowChill;     // tenths of a degree F
   ...
};
```

Listing B.3: Del strukture dnevnega povzetka 1.

B.4 Dnevni povzetek 2

```
struct DailySummary2
{
  BYTE dataType = 3;
  BYTE  reserved;   // this will cause the rest of the fields to
       start on an even address

  // this field is not used now.
  unsigned short todaysWeather; // bitmapped weather conditions
      (Fog, T-Storm, hurricane, etc)

```

```
  short numWindPackets;       // # of valid packets containing
     wind data,
                 // this is used to indicate reception quality
  short hiSolar;              // Watts per meter squared
  short dailySolarEnergy;     // 1/10'th Ly
  ...
};
```

Listing B.4: Del strukture dnevnega povzetka 2.

B.5 Zapis vremenskega podatka

```
struct WeatherDataRecord
{
   BYTE dataType = 1;
   BYTE archiveInterval;       // number of minutes in the
      archive
   // see below for more details about these next two fields)
   BYTE iconFlags;             // Icon associated with this
      record, plus Edit flags
   BYTE moreFlags;             // Tx Id, etc.

   short packedTime;           // minutes past midnight of the
      end of the archive period
   short outsideTemp;          // tenths of a degree F
   short hiOutsideTemp;        // tenths of a degree F
   short lowOutsideTemp;       // tenths of a degree F
   ...
};
```

Listing B.5: Del strukture vremenskih podatkov.

Dodatek C

Implementacija specifikacij

C.1 Skupna koda

```
public ByteOrder byteOrder() {
  return ByteOrder.LITTLE_ENDIAN;
}
public boolean isPacked() {
  return true;
}
```

Listing C.1: Del kode, ki določa tanki konec in poravnanost pomnilniških naslovov.

C.2 Glava datoteke

```
public class GlavaBlok extends Struct {
  public final UTF8String idKoda = new UTF8String(16);
  public final Signed32 skupajZapisov = new Signed32();
  public final DanIndeks[] danIndeks = array(new DanIndeks[32]);
  ...
}
```

Listing C.2: Del kode razreda, ki določa glavo datoteke.

C.3 Dnevni indeks

```
public class DanIndeks extends Struct {
  public final Signed16 steviloZapisovVDnevu = new Signed16();
  public final Signed32 zacetnaPozicija = new Signed32();
  ...
}
```

Listing C.3: Del kode razreda, ki določa dnevne indekse.

C.4 Dnevni povzetek 1

```
public class DnevniPovzetek1 extends Struct {
  public final Signed8 podatkovniTip = new Signed8();
  public final Signed8 rezervirano = new Signed8();
  public final Signed16 steviloMinut = new Signed16();
  public final Signed16 najvisjaZunanjaTemperatura = new
      Signed16();
  public final Signed16 najnizjaZunanjaTemperatura = new
      Signed16();
  ...
}
```

Listing C.4: Del kode razreda, ki določa dnevni povzetek 1.

C.5 Dnevni povzetek 2

```
public class DnevniPovzetek2 extends Struct {
  public final Signed8 podatkovniTip = new Signed8();
  public final Signed8 rezervirano = new Signed8();
  public final Unsigned16 trenutnoVreme = new Unsigned16();
  public final Unsigned16 steviloPaketovVeter = new Unsigned16()
      ;
  ...
}
```

Listing C.5: Del kode razreda, ki določa dnevni povzetek 2.

C.6 Zapis vremenskega podatka

```
public class VremenskiPodatkiZapis extends Struct {
  public final Signed8 podatkovniTip = new Signed8();
  public final Signed8 intervalArhiva = new Signed8();
  public final Signed8 ikonaZastavice = new Signed8();
  public final Signed8 vecZastavic = new Signed8();
  public final Signed16 pakiranCas = new Signed16();
  public final Signed16 zunanjaTemperatura = new Signed16();
  public final Signed16 najvisjaZunanjaTemperatura = new
      Signed16();
  public final Signed16 najnizjaZunanjaTemperatura = new
      Signed16();
  ...
}
```

Listing C.6: Del kode razreda, ki določa vremenski podatek.

Dodatek D

Branje datoteke

D.1 Pretvarjanje podatkov

```
public class PretvorbaPodatkov {
  // Enote za pretvorbo imperialnih enot v metricne enote.
  static Unit<Pressure> HECTO_PASCAL = SI.HECTO(SI.PASCAL);
  static Unit<Pressure> INCH_OF_MERCURY_MILI = SI.MILLI(NonSI.
      INCH_OF_MERCURY);
  ...
  protected static BigDecimal izracunajPritiskRelativni(Short
      pritiskRelativni) {
    if (pritiskRelativni == Short.MIN_VALUE) {
      return null;
    } else {
      // Pretvorimo iz imerialnih enot v metricne enote (hekta
          pascale).
      BigDecimal pritiskRelativniPretvorjen = BigDecimal.valueOf
          (INCH_OF_MERCURY_MILI.getConverterTo(HECTO_PASCAL).
          convert(pritiskRelativni));

      // Zaokrozimo na eno decimalno mesto.
      pritiskRelativniPretvorjen = pritiskRelativniPretvorjen.
          setScale(1, RoundingMode.HALF_UP);

      return pritiskRelativniPretvorjen;
```

```
17      }
    }
19  ...
}
```

Listing D.1: Del kode razreda za pretvarjanje v decimalni metrični relativni pritisk.

D.2 Pridobivanje vmesnikov

```
public class VmesnikiZaZrna {
2   @SuppressWarnings({ "rawtypes", "unchecked" })
    protected static VremenskiPodatkiZrnoRemote
        vrniVmesnikZaVremenskiPodatki() throws NamingException {
4     //Kreiramo tabela kljuc - vrednost za shranjevanje lastnosti
          .
      final Hashtable jndiLastnosti = new Hashtable();
6     jndiLastnosti.put(Context.URL_PKG_PREFIXES, "org.jboss.ejb.
          client.naming");
          final Context context = new InitialContext(jndiLastnosti
              );
8         // Kreiramo niz za poizvedbo sestavljen iz prej
              nastavljenih nizov.
          String poizvedba = vrniJNDIPoizvedbo(
              VremenskiPodatkiZrno.class.getSimpleName(),
              VremenskiPodatkiZrnoRemote.class.getName());
10        // Vrnemo referenco glede na poizvedbo.
          return (VremenskiPodatkiZrnoRemote) context.lookup(
              poizvedba);
12  }
    ...
14  }
```

Listing D.2: Del kode razreda za dostop do oddaljenih vmesnikov.

D.3 Prenos podatkov

```
public class VremenskiPodatekPrenos implements Serializable {
  private static final long serialVersionUID = 1L;
  private DateTime datumCas;
  private Byte intervalPodatkov;
  ...
  public DateTime getDatumCas() {
    return datumCas;
  }
  public void setDatumCas(DateTime datumCas) {
    this.datumCas = datumCas;
  }
  public Byte getIntervalPodatkov() {
    return intervalPodatkov;
  }
  public void setIntervalPodatkov(Byte intervalPodatkov) {
    this.intervalPodatkov = intervalPodatkov;
  }
  ...
}
```

Listing D.3: Del kode razreda za prenos podatkov na AS.

Dodatek E

Zapisovanje podatkov

E.1 Zapisovanje novih podatkov

```
@Stateless
public class ObdelavaNovihPodatkovZrno implements
    ObdelavaNovihPodatkovZrnoRemote,
    ObdelavaNovihPodatkovZrnoLocal {
  @EJB
  private DanMinimumZrnoLocal danMinimumZrnoLocal;
  ...
    public ObdelavaNovihPodatkovZrno() {
    }
  @Override
  public void obdelajPodatke(VremenskiPodatekPrenos
      vremenskiPodatekPrenos) {
    VremenskiPodatki vremenskiPodatki = new VremenskiPodatki();
    VremenskiPodatki zadnjiZapis = vremenskiPodatkiZrnoLocal.
        vrniZadnjiZapis();
    DateTime datumPrejsnji = null;
    if (zadnjiZapis != null) {
      datumPrejsnji = new DateTime(zadnjiZapis.getDatumCas());
    }
    DateTime datumZdajsnji = vremenskiPodatekPrenos.getDatumCas
        ();
    vremenskiPodatki.setDatumCas(datumZdajsnji.toDate());
```

```
        ...
19    }
}
```

Listing E.1: Del kode zrna za obdelavo novih podatkov.

Dodatek F

Realizacija na AS

F.1 Zrna

```
@Stateless
public class VremenskiPodatkiZrno implements
    VremenskiPodatkiZrnoRemote, VremenskiPodatkiZrnoLocal {
  @PersistenceContext(unitName="VremenskaPostajaJPA")
    private EntityManager upravljavecEntitet;
    public VremenskiPodatkiZrno() {
    }
    @Override
  public void kreirajVremenskiPodatki(VremenskiPodatki
      vremenskiPodatki) {
       upravljavecEntitet.persist(vremenskiPodatki);
  }
    @Override
  public VremenskiPodatki branjeVremenskiPodatki(int
      idVremenskiPodatki) {
    VremenskiPodatki vremenskiPodatki = upravljavecEntitet.find(
        VremenskiPodatki.class, idVremenskiPodatki);
    return vremenskiPodatki;
  }
    @Override
  public void posodobiVremenskiPodatki(VremenskiPodatki
      vremenskiPodatki) {
```

```
18      upravljavecEntitet.merge(vremenskiPodatki);
    }
20      @Override
    public void brisanjeVremenskiPodatki(VremenskiPodatki
        vremenskiPodatki) {
22      upravljavecEntitet.remove(vremenskiPodatki);
    }
24      ...
}
```

Listing F.1: Del kode zrna za upravljanje z vremenskimi podatki.

```
1  @Stateless
public class RosisceZrno implements RosisceZrnoRemote,
     RosisceZrnoLocal {
3    @PersistenceContext(unitName="VremenskaPostajaJPA")
      private EntityManager upravljavecEntitet;
5    ...
    @Override
7    public BigDecimal izracunajRosisce(double temperatura, double
        vlaga) {
      double nasicenaPara = 6.112 * Math.pow(10, ((7.5 *
          temperatura) / (237.7 + temperatura)));
9     double rosisceIzracunano = (243.5 * Math.log((nasicenaPara *
           vlaga) / 611)) / (17.67 - Math.log((nasicenaPara * vlaga
          ) / 611));
      BigDecimal rosisceZaokrozeno = BigDecimal.valueOf(
          rosisceIzracunano);
11    rosisceZaokrozeno = rosisceZaokrozeno.setScale(1,
          RoundingMode.HALF_UP);
      return rosisceZaokrozeno;
13   }
}
```

Listing F.2: Del kode zrna za izračun rosišča.

F.2 Lokalni vmesniki zrn

```
@Local
public interface VremenskiPodatkiZrnoLocal {
  public void kreirajVremenskiPodatki(VremenskiPodatki
      vremenskiPodatki);
  public VremenskiPodatki branjeVremenskiPodatki(int
      idVremenskiPodatki);
  public void posodobiVremenskiPodatki(VremenskiPodatki
      vremenskiPodatki);
  public void brisanjeVremenskiPodatki(VremenskiPodatki
      vremenskiPodatki);
  public BigDecimal izracunajPadavineAbsolutno(BigDecimal
      padavineAbsolutno, BigDecimal padavineRelativno);
  ...
}
```

Listing F.3: Del kode lokalnega vmesnika.

F.3 Oddaljeni vmesniki zrn

```
@Remote
public interface VremenskiPodatkiZrnoRemote {
  public VremenskiPodatki vrniZadnjiZapis();
}
```

Listing F.4: Del kode oddaljenega vmesnika.

F.4 JPA entitete

```
@Entity
@NamedQueries({
  @NamedQuery(name="vrniVlagaZunanjaGledeNaVrednost", query="
      SELECT vz FROM VlagaZunanja vz WHERE vz.vlagaZunanja = :
      vlagaZunanja")
})
@Table(name="vlagazunanja")
```

```
public class VlagaZunanja implements Serializable {
  private static final long serialVersionUID = 1L;
  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  @Column(unique=true, nullable=false)
  private byte idVlagaZunanja;
  private Byte vlagaZunanja;
  ...
    @XmlTransient
  public byte getIdVlagaZunanja() {
    return this.idVlagaZunanja;
  }
  public void setIdVlagaZunanja(byte idVlagaZunanja) {
    this.idVlagaZunanja = idVlagaZunanja;
  }
  @XmlValue
  public Byte getVlagaZunanja() {
    return this.vlagaZunanja;
  }
  public void setVlagaZunanja(Byte vlagaZunanja) {
    this.vlagaZunanja = vlagaZunanja;
  }
  ...
}
```

Listing F.5: Del implementacija entitete VlagaZunanja.

F.5 Konverterji

```
Gson gsonTemperatura = new GsonBuilder()
  .registerTypeAdapter(VremenskiPodatki.class, new
     KonverterGrafTemperatura())
  .serializeNulls()
  .create();
public class KonverterGrafTemperatura implements JsonSerializer<
   VremenskiPodatki> {
  @Override
```

```
7   public JsonElement serialize(VremenskiPodatki src, Type
        typeOfSrc,
        JsonSerializationContext context) {
9     JsonElement jsonElement;
      if (src.getTemperaturaZunanja() != null) {
11      jsonElement = new JsonPrimitive(src.getTemperaturaZunanja
            ().getTemperaturaZunanja());
      } else {
13      jsonElement = JsonNull.INSTANCE;
      }
15    return jsonElement;
    }
17 }
```

Listing F.6: Del implementacija konverterja za temperaturo.

F.6 Kontrolerji

```
1 @Inject
  VremenskiPodatkiZrnoLocal vremenskiPodatkiZrnoLocal;
```

Listing F.7: Primer vstavljanja v kontrolerju.

```
  @PostConstruct
2 public void init() {
    vremenskiPodatki = vremenskiPodatkiZrnoLocal.vrniZadnjiZapis()
        ;
4   danMaksimum = danMaksimumZrnoLocal.vrniZadnjiZapis();
    ...
6   vremenskiPodatkiZadnjih24H = vremenskiPodatkiZrnoLocal.
        vrniZadnjih24H(vremenskiPodatki);
    ...
8 }
```

Listing F.8: Primer inicializacije v kontrolerju.

```
public void getTemperaturaRosisceObcutekMrazaZacetni() {
```

```
  Gson gsonTemperatura = new GsonBuilder()
    .registerTypeAdapter(VremenskiPodatki.class, new
        KonverterGrafTemperatura())
    .serializeNulls()
    .create();
  Gson gsonRosisce = new GsonBuilder()
    .registerTypeAdapter(VremenskiPodatki.class, new
        KonverterGrafRosisca())
    .serializeNulls()
    .create();
  Gson gsonObcutekMraza = new GsonBuilder()
    .registerTypeAdapter(VremenskiPodatki.class, new
        KonverterGrafObcutekMraza())
    .serializeNulls()
    .create();
  RequestContext requestContext = RequestContext.
      getCurrentInstance();
  requestContext.addCallbackParam("podatkiTemperatura",
      gsonTemperatura.toJson(vremenskiPodatkiZadnjih24H));
  requestContext.addCallbackParam("podatkiRosisce", gsonRosisce.
      toJson(vremenskiPodatkiZadnjih24H));
  requestContext.addCallbackParam("podatkiObcutekMraza",
      gsonObcutekMraza.toJson(vremenskiPodatkiZadnjih24H));
}
```

Listing F.9: Primer pridobivanja podatkov za izris grafov.

F.7 Spletne storitve

```
@Path("/trenutni")
public class TrenutniPodatkiStoritev {
  @Inject
  private VremenskiPodatkiZrnoLocal vremenskiPodatkiZrnoLocal;
  ...
}
```

Listing F.10: Primer glave storitve za trenutni podatki.

```
@GET()
@Path("trenutniTXT")
@Produces("text/plain; charset=UTF-8")
public String trenutniTXT() {
  VremenskiPodatki vremenskiPodatki = vremenskiPodatkiZrnoLocal.
      vrniZadnjiZapis();
  StringBuilder trenutniTxt = new StringBuilder("");
  if (vremenskiPodatki != null) {
    trenutniTxt.append("Datum in cas: ");
    DateTime datumCas = new DateTime(vremenskiPodatki.
        getDatumCas());
    trenutniTxt.append(datumCas.toString("HH:mm dd.MM.yyyy") + "
        \n");
    trenutniTxt.append("Temperatura: ");
    if (vremenskiPodatki.getTemperaturaZunanja() != null) {
      trenutniTxt.append(vremenskiPodatki.getTemperaturaZunanja
          ().getTemperaturaZunanja().toPlainString() + "°C\n");
    } else {
      trenutniTxt.append("null\n");
    }
    ...
  }
  return trenutniTxt.toString();
}
```

Listing F.11: Primer implementacije storitve trenutni podatki v TXT obliki.

```
@GET()
@Path("trenutniHTML")
@Produces("text/html; charset=UTF-8")
public String trenutniHTML() {
  VremenskiPodatki vremenskiPodatki = vremenskiPodatkiZrnoLocal.
      vrniZadnjiZapis();
  StringBuilder trenutniHtml = new StringBuilder("<html><head>
      meta http-equiv=\"Content-Type\" content=\"text/html;
      charset=UTF-8\" /><title>Trenutni podatki</title></head>
      body>");
  if (vremenskiPodatki != null) {
```

```
8       trenutniHtml.append("Datum in cas: ");
        DateTime datumCas = new DateTime(vremenskiPodatki.
            getDatumCas());
10      trenutniHtml.append(datumCas.toString("HH:mm dd.MM.yyyy") +
            "<br />");
        trenutniHtml.append("Temperatura: ");
12      if (vremenskiPodatki.getTemperaturaZunanja() != null) {
          trenutniHtml.append(vremenskiPodatki.getTemperaturaZunanja
              ().getTemperaturaZunanja().toPlainString() + "°C<br />"
              );
14      } else {
          trenutniHtml.append("null<br />");
16      }
        ...
18    }
      trenutniHtml.append("</body></html>");
20    return trenutniHtml.toString();
  }
```

Listing F.12: Primer implementacije storitve trenutni podatki v HTML obliki.

```
1 @GET()
  @Path("trenutniXML")
3 @Produces(MediaType.APPLICATION_XML)
  public VremenskiPodatki trenutniXML() {
5   // Pridobimo zadnji zapis.
    VremenskiPodatki vremenskiPodatki = vremenskiPodatkiZrnoLocal.
        vrniZadnjiZapis();
7   return vremenskiPodatki;
  }
```

Listing F.13: Primer implementacije storitve trenutni podatki v XML obliki.

```
@GET()
2 @Path("trenutniJSON")
  @Produces(MediaType.APPLICATION_JSON)
4 public String trenutniJSON() {
```

```
    VremenskiPodatki vremenskiPodatki = vremenskiPodatkiZrnoLocal.
        vrniZadnjiZapis();
    Gson gson = new GsonBuilder()
      .registerTypeAdapter(VremenskiPodatki.class, new
          KonverterTrenutniPodatkiStoritev())
      .serializeNulls()
      .create();
    String json = gson.toJson(vremenskiPodatki);
    return json;
}
```

Listing F.14: Primer implementacije storitve trenutni podatki v JSON obliki.

F.8 Spletne strani

```
<h:outputStylesheet name="osnovni.css" library="css" />
<h:outputStylesheet name="ogrodje.css" library="css" />
<f:facet name="last">
  <ui:insert name="skriptaDoloceneStrani" />
  <h:outputScript name="skripta.js" library="javascript" />
  <ui:insert name="stilDoloceneStrani" />
</f:facet>
```

Listing F.15: Definirani knjižnice in stili v ogrodju.

```
<h:body>
  <div id="prekrivanje">
    <div id="glava-crta" />
    <div id="glava">
      <div id="logo">
        <h:graphicImage library="images" name="Logo-VP-Zalec.png
            " />
      </div>
      <div id="menu">
        <ul>
          <c:set var="urlSpletneStoritve" value="#{fn:contains(
              request.requestURI, 'spletneStoritve')}" />
```

```
11          <li class="${urlSpletneStoritve ? 'izbran-menu' : ''}"
                >
              <h:outputLink id="spletneStoritve" value="${
                  facesContext.externalContext.requestContextPath}/
                  pages/spletneStoritve.xhtml" style="padding-top:
                  30px; padding-bottom: 30px; margin-top: -14px;
                  background-color: ${urlSpletneStoritve ? '#FFFFFF
                  ;' : '#F5F5ED;'} color: ${urlSpletneStoritve ?
                  '#3F3FF1;' : '#2E494C;'}">Spletne storitve</h:
                  outputLink>
13          </li>
            <c:set var="urlTrenutniPodatki" value="#{fn:contains(
                request.requestURI, 'trenutniPodatki')}" />
15          <li class="${urlTrenutniPodatki ? 'izbran-menu' : ''}"
                >
              <h:outputLink id="trenutniPodatki" value="${
                  facesContext.externalContext.requestContextPath}/
                  pages/trenutniPodatki.xhtml" style="padding-top:
                  30px; padding-bottom: 30px; margin-top: -14px;
                  background-color: ${urlTrenutniPodatki ? '#FFFFFF
                  ;' : '#F5F5ED;'} color: ${urlTrenutniPodatki ?
                  '#3F3FF1;' : '#2E494C;'}">Podatki</h:outputLink>
17          </li>
            <c:set var="urlOpis" value="#{fn:contains(request.
                requestURI, 'index')}" />
19          <li class="${urlOpis ? 'izbran-menu' : ''}">
              <h:outputLink id="index" value="${facesContext.
                  externalContext.requestContextPath}/pages/index.
                  xhtml" style="padding-top: 25px; padding-bottom:
                  25px; margin-top: -10px; background-color: ${
                  urlOpis ? '#FFFFFF;' : '#F5F5ED;'} color: ${
                  urlOpis ? '#3F3FF1;' : '#2E494C;'}">Opis</h:
                  outputLink>
21          </li>
            <li class="clear"></li>
23        </ul>
        </div>
25      <div class="pocisti" />
```

```
    </div>
    <div id="vsebina">
      <ui:insert name="vsebina">
        <ui:include src="/WEB-INF/templates/privzeto/
            privzetaVsebina.xhtml" />
      </ui:insert>
    </div>
    <div id="noga">
      © 2012 Jernej Jerin. Vse pravice pridržane.
    </div>
  </div>
</h:body>
```

Listing F.16: Glavni del ogrodja.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "
    http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:h="http://java.sun.com/jsf/html">
  <ui:composition template="/WEB-INF/templates/ogrodje.xhtml">
    <ui:define name="naslovStrani">Opis vremenske postaja Žalec<
        /ui:define>
    <ui:define name="skriptaDoloceneStrani">
      <script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.0/
          jquery.min.js"></script>
      <script src="//ajax.googleapis.com/ajax/libs/jqueryui
          /1.8.22/jquery-ui.min.js"></script>
    </ui:define>
    <ui:define name="stilDoloceneStrani">
      <h:outputStylesheet name="opis.css" library="css" />
    </ui:define>
    <ui:define name="vsebina">
      <div id="uvod">
        <div id="kratek-opis">
          <h2>Avtomatska meteorološka</h2>
          <h1>postaja Žalec</h1>
          <p>Vremenska postaja Žalec se nahaja na severo-
```

```
            vzhodnem predelu mesta Žalec na nadmorski višini
            255m. Postaja je tipa Davis Vantage Pro2,
20          thermo-hygro oddajnik je postavljen v vremenski hišici
            na višini 2m. Dežemer se nahaja na dodatnem
            nosilcu, le ta pa je pritrjen na glavno
            stojalo. Je dovolj odmaknjen od ostalih objektov okoli
            njega, tako da meri kolicino pravilno. Dežemer je
            tudi po potrebi ogrevan za taljenje
22          snega. Merilnik za hitrost vetra (anemometer) se
            nahaja na strehi na višini 10m. Vremenska hišica se
            tako nahaja na travniku pred hišo ter je
            oddaljena od hiše 5m.</p>
24        </div>
          <div id="zadnja-slika-postaje">
26          <h:graphicImage library="images" name="vremenska-
              postaja-zadnja.jpg" />
          </div>
28        <div class="pocisti" />
        </div>
30      <div id="zacetki">
          <div id="zacetki-slika-postaje">
32          <h:graphicImage library="images" name="zacetki-postaje
              -prva.jpg" />
          </div>
34        <div id="zacetki-opis">
            <h2>Zgodovina meteorološke</h2>
36          <h1>postaje Žalec</h1>
            <p>Postaja deluje že od 14.8.2008 dalje, tocni podatki
              pa so na voljo na internetu od 5.8.2009 dalje,
38          namrec takrat je bila postaja premešcena v vremensko
              hišico in tudi od takrat je na internetu dostopna
              spletna stran.
            Arhiv podatkov se zacne z 02:10 20.12.2009 kot tudi
              maksimumi in minimumi se merijo od takrat dalje.</p
              >
40        </div>
          <div class="pocisti" />
42      </div>
```

```
    </ui:define>
  </ui:composition>
</html>
```

Listing F.17: Primer uporabe ogrodja.