**UNIVERZA V LJUBLJANI**
**FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO**

**Tomaž Sabo**

# Napoved jedilnika in nakupovalne liste v inteligentnem hladilniku

**DIPLOMSKO DELO**
**NA UNIVERZITETNEM ŠTUDIJU**

**Mentorica: doc. dr. Mira Trebar**

**Ljubljana, 2008**

# I Z J A V A  O  A V T O R S T V U

## diplomskega dela

Spodaj podpisani Tomaž Sabo

z vpisno številko 63960129

sem avtor diplomskega dela z naslovom:

Napoved jedilnika in nakupovalne liste v inteligentnem hladilniku

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  doc. dr. Mire Trebar

- 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 12.12.2008 Podpis avtorja: _____

**ZAHVALA**

Zahvaljujem se mentorici doc. dr. Miri Trebar za vložen trud, vodenje, strokovno pomoč in nasvete pri izdelavi diplomske naloge in vsem, ki so kakorkoli pripomogli k nastanku te diplomske naloge.

Hvala Magdi, Rajku, Sergeji in Katji, ker so me vzpodbujali in mi stali ob strani skozi celoten študij. Hvala vsem prijateljem, zaradi katerih je bil čas študija nepozaben.

**KAZALO**


**1. Uvod ... 1**

**2. Inteligentni hladilnik ... 3**

**3. Uporabljena orodja in tehnologije ... 5**

3.1. Tehnologija RFID ... 5

3.2. Podatkovna baza PostgreSQL ... 7

3.3. Tehnologija Java ... 7

3.4. Razvojno okolje Eclipse ... 8

3.5. Apache Tomcat ... 9

3.6. Spletna storitev ... 10

3.6.1. XML ... 11

3.6.2. WSDL ... 12

3.6.3. SOAP ... 13

3.6.4. UDDI ... 15

**4. Zasnova in izvedba koncepta napovedovanja ... 17**

4.1. Opis sistema inteligentnega hladilnika ... 17

4.2. Značka RFID in struktura podatkov ... 19

4.3. Zasnova podatkovne baze ... 23

4.4. Izvedba programske aplikacije ... 30

4.4.1. Pregledovanje zaloge ... 31

4.4.2. Predlogi kuharskih receptov ... 32

4.4.3. Tehtanje izdelkov ... 34

4.4.4. Urejanje nakupovalnega seznama (listka) ... 35

4.5. Spletna storitev za iskanje receptov ... 37

4.5.1. Izdelava shem XML ... 37

4.5.2. Preslikava shem XML v razrede Java ... 39

4.5.3. Deserializacija in serializacija podatkov ... 40

4.5.4. Kreiranje dokumenta WSDL ... 41

4.5.5. Kreiranje odjemalca ... 42

4.5.6. Namestitev spletne storitve ... 44

4.5.7. Preizkus spletne storitve ... 44

**5. Sklepne ugotovitve ... 47**

**6. Viri ... 49**

## SEZNAM UPORABLJENIH KRATIC IN SIMBOLOV

| | |
|---|---|
| **AAR** | Axis Archive |
| **ACID** | Atomicity, Consistency, Isolation, Durability |
| **AFI** | Application Family Identifier |
| **AXIOM** | Axis Object Model |
| **CORBA** | Common Object Request Broker Architecture |
| **EAS** | Electronic Article Surveillance |
| **EEPROM** | Electrically Erasable Programmable Read-Only Memory |
| **EPC** | Electronic Product Code |
| **FK** | Foreign Key |
| **HTML** | Hyper Text Markup Language |
| **IDE** | Integrated Development Environment |
| **JAVA EE** | Java Platform, Enterprise Edition |
| **JAXB** | Java Architecture for XML Binding |
| **JDBC** | Java Database Connectivity |
| **JSP** | JavaServer Pages |
| **PK** | Primary Key |
| **RFID** | Radio Frequency Identification |
| **RPC** | Remote Procedure Call |
| **SMTP** | Simple Mail Transfer Protocol |
| **SOAP** | Simple Object Access Protocol |
| **SQL** | Structured Query Language |
| **UDDI** | Universal Description, Discovery and Integration |
| **URL** | Uniform Resource Locator |
| **WAR** | Web Archive |
| **WSDL** | Web Service Definition Language |
| **XML** | Extensible Markup Language |

# POVZETEK

V diplomski nalogi sem izdelal programsko aplikacijo za sistem inteligentnega hladilnika. Namen aplikacije je ugotavljanje zaloge prehrambnih izdelkov opremljenih z značko RFID, napovedovanje jedilnika, ki ga lahko naredimo iz zaloge in receptov, ter kreiranje in urejanje nakupovalnega seznama manjkajočih izdelkov. Za potrebe aplikacije sem zasnoval podatkovno zbirko izdelkov in receptov in definiral ustrezno strukturo podatkov za zapis na značke RFID. Funkcijo napovedovanja jedilnika iz zaloge sem implementiral tudi kot spletno storitev, s čimer sem prikazal eno od možnosti uporabe spletnih tehnologij v takem sistemu inteligentnega hladilnika. Aplikacija in spletna storitev sta napisani v programskem jeziku Java, za podatkovno bazo je uporabljena PostgreSQL.

Delovanje aplikacije sem testiral v realnem prototipu sistema inteligentnega hladilnika. Z uporabo testnih čitalcev RFID je aplikacija zadovoljivo delovala, tako da sem preizkusil vse njene funkcionalnosti. Podal sem nekaj možnih izboljšav in razširitev sistema, kot so razširitev podatkovne baze z večjim številom izdelkov in receptov, podrobnejšo kategorizacijo izdelkov, razširitev podatkov o izdelkih, povezava aplikacije s spletno trgovino in možnost analiziranja uporabnikovih navad na osnovi beleženja zgodovine. Programska aplikacija je zasnovana dovolj splošno, da omogoča enostavno nadgradnjo sistema z dodatnimi funkcijami.



**Ključne besede:**

inteligentni hladilnik, napovedovanje jedilnika, spletna storitev, sistem RFID, kuharski recepti

## ABSTRACT

In my diploma work I designed and implemented an application for a system of the intelligent refrigerator. The purpose of such application is an assessment of provisions equipped with RFID tags, a prediction of the menu based on the recipes that can be made of such food supplies, a creation and editing of the shopping lists for the lacking food products. For the requirements of the application I designed a database of food products and recipes and determined a suitable data structure for writing information on the RFID tags. I implemented the refrigerator's function of the prediction of the menu with regard to the food supplies also as a web service and by doing so I demonstrated one of many possible uses of web services in such intelligent refrigerator system. The application and the web service are both written in Java, while PostgreSQL is used for the database.

I tested the application on a real intelligent refrigerator system prototype. With the use of test RFID readers the application worked sufficiently so I could test all its functions. I offered some possible improvements and expansions of the system, for example, the expansion of the database with a larger amount of food products and recipes, the more detailed categorization of food products, the expansion of the food products data, the connection of the application with online grocery stores and the possibility of analysing the consumer's eating habits on the basis of the recorded history. The application is designed generally enough to enable a simple expansion of the system with additional functions.



**Key words:**

Intelligent refrigerator, prediction of the menu, web service, RFID system, cooking recipes

# 1. Uvod

Princip pametne hiše je sicer poznan že nekaj časa, a zelo počasi prodira v življenje povprečnega človeka in vprašanje je, v kolikšni meri sploh bo. Vendar so možnosti neskončne in zato me je ideja o hladilniku s tehnologijo RFID in izdelavi modela takega sistema takoj pritegnila.

Z razvojem tehnologije se vedno znova srečujemo z idejami, kako bi lahko nove tehnologije z uporabo obstoječih tehnologij spremenile in olajšale življenje ljudi. Včasih so te ideje zgolj zamisli ali sanje znanstvenikov, inženirjev in inovatorjev, ki ostanejo nekje na papirju in nikoli ne ugledajo luči sveta, bodisi zato, ker preprosto niso tehnološko mogoče, bodisi še ne poznamo tehnologije za njihovo uresničitev. Spet drugič se rodijo ideje, ki so lahko realizirane takoj, ker tehnologija in z njo možnosti za njihovo realizacijo že obstajajo, le nihče se takšne rešitve še ni domislil. Tako se lahko hitro uveljavijo in začnejo uporabljati v vsakdanjem življenju. Včasih pa določena tehnologija že obstaja, idej za njeno uporabo je veliko, vendar pa še ni dovolj razvita oziroma je še predraga za množično uporabo. Ena takih je tehnologija radiofrekvenkčne identifikacije oz. krajše tehnologija RFID (Radio Frequency IDentification).

Tehnologija RFID je znana že nekaj desetletij, a se v množični rabi resneje uveljavlja šele zadnjih deset let. Možnosti njene uporabe so ogromne in segajo praktično na vsa področja, kot so: proizvodnja, logistika in transport, trgovina, zdravstvo, šolstvo, kultura, šport, turizem, itd. V tem trenutku se množično še ne uporablja v vsakdanjem življenju ljudi, saj je še predraga, pričakuje pa se, da bo v bližnji prihodnosti (ocenjuje se, da v roku petih do desetih let) tudi veliko izdelkov za domačo uporabo opremljenih s tehnologijo RFID, tako kot se pri večini izdelkov danes uporablja črtna koda. Če, oziroma ko se bo to zgodilo, bo mogoče to tehnologijo uporabljati tudi doma.

Ena od bolj privlačnih možnosti uporabe je v t.i. "inteligentnih" napravah, ki bi bile programirane tako, da bi s pomočjo RFID prepoznavale posamezne izdelke, ki jih uporabljamo v vsakdanjem življenju, svetovale o možnostih njihove uporabe, opozarjale na določene lastnosti izdelkov ali pa preprosto opravile določene naloge namesto človeka. Tako bi nas lahko na primer pralni stroj opozoril, če je v bobnu preveč perila, ali pa sam izbral primeren program pranja in predvidel potrebno količino vode in detergenta glede na to, kakšno perilo smo vstavili vanj. Koš za smeti bi nas lahko pri ločevanju odpadkov, katerega pomena se vedno bolj zavedamo tudi v gospodinjstvih, opozoril na neprimerno odvržen odpadek, hladilnik pa bi nas lahko opozoril, če je kateremu živilu potekel rok uporabe, katerih živil nam zmanjkuje, nam predlagal nakup novih zalog in jih v primeru, da se z nakupom strinjamo, naročil preko interneta v spletni trgovini.

Pri izdelavi diplomske naloge so bili zastavljeni naslednji cilji:

- preučiti tehnologije, ki se lahko uporabijo v takem sistemu in analizirati podatkovno strukturo značk pri uporabi tehnologije RFID,

- pripraviti bazo podatkov,
- izdelati programsko opremo za tak hladilnik,
- izdelati spletno storitev, s katero bi prikazal možnosti povezave hladilnika s spletnimi storitvami, razvitimi v ta namen,
- preizkusiti delovanje aplikacije,
- oceniti uporabnost, slabosti in predlagati morebitne izboljšave v povezavi z realnimi testiranji izdelanega prototipa.

# 2. Inteligentni hladilnik

Hladilnik, ki je opremljen s tehnologijo RFID, bi se s pomočjo drugih vgrajenih tehnologij in programske opreme lahko "zavedal" svoje vsebine. Lastniku takega hladilnika na sliki 1 bi bili preko vgrajenega zaslona na dotik dostopni podatki o vsebini hladilnika in tudi o drugih živilih v kuhinji. Hladilnik, povezan z internetom, nudi še dodatne možnosti, kot so povezovanje s spletnimi storitvami, spletno naročanje in nakupovanje izdelkov, dostop do informacij o zalogi živil preko spleta, mobilnih naprav in podobno.

*Slika 1. Hladilnik z vgrajenim zaslonom.*

Predpogoj za uporabo takega hladilnika je, da je vsako živilo opremljeno z značko RFID. V hladilnik vgrajen čitalnik RFID z značke prebere podatke o živilu. Tako lahko za vsako živilo kar preko zaslona spremljamo podatke, kot so:

- ime in vrsta živila,
- njegova količina in teža,
- rok uporabe,
- katere sestavine vsebuje,
- kdo je njegov proizvajalec,
- kakšna mora biti njegova temperatura skladiščenja, ipd.

Zbrane podatke uporabljamo za sprejemanje različnih odločitev, ki nam jih ponudi ali pa uresniči že sam hladilnik oz. vanj vgrajena programska oprema.

Inteligentni hladilnik bi bil dobrodošla in koristna pridobitev za marsikatero gospodinjstvo. Hitro lahko najdemo in naštejemo nekaj zanimivih prednosti, ki bi jih lahko imel tak hladilnik, na primer:

- prihranek energije – ker bi lahko zalogo živil v hladilniku spremljali preko zaslona, ne bi bilo potrebno pogosto odpiranje vrat hladilnika. Število odpiranj vrat in čas, ko so vrata odprta, imata namreč velik vpliv na porabo energije. Raziskave kažejo, da se z vsakim odpiranjem vrat porabi od 10-28 Wh energije in da, v kolikor vrata v enem dnevu odpremo 26-krat, povečamo porabo energije za 12% [8].
- manj zavržene hrane – hladilnik bi nas opomnil, da bo živilu potekel rok uporabe in da naj ga čim prej porabimo. S tem bi zmanjšali količino hrane, ki bi jo zaradi morebitne nepazljivosti sicer zavrgli.
- opozarjanje na izdelke, ki jim je potekel rok uporabe nas lahko odvrne od zaužitja pokvarjenega izdelka.
- raznolikost prehranjevanja – hladilnik bi nam lahko predlagal kuharske recepte, ki jih je možno pripraviti z našo trenutno zalogo živil. Tako bi se verjetno večkrat odločili za preizkušanje novih receptov, saj je siceršnje brskanje po internetu ali kuharskih knjigah bolj zamudno. Vsak recept, za katerega nimamo dovolj sestavin, moramo namreč izločiti ali pa iti v nakup manjkajočih sestavin.
- ažurnost količine živil – tehtnica v hladilniku bi sproti merila in na značke izdelkov zapisovala njihovo težo. Tako bi primerjava med trenutno bruto težo posameznega izdelka, njegovo bruto težo in neto količino polnega ali celega izdelka dala informacijo o tem, koliko izdelka imamo še na voljo. Ko bi ga začelo primanjkovati, bi nas hladilnik na to opomnil, ga dodal na nakupovalni seznam ali pa ga naročil v spletni trgovini.
- zdravju nevarni izdelki – v primeru, da ima uporabnik določene zdravstvene težave, bi lahko hladilnik zanj vodil seznam nedovoljenih živil. Tako bi ga lahko opozoril, če določen izdelek vsebuje nedovoljene sestavine oziroma je zanj neprimeren.
- lažji nadzor kaloričnih vrednosti – na ekranu bi se skupaj z izdelkom ali receptom izpisovala tudi njegova kalorična vrednost, s čimer bi povečali nadzor nad energetsko vrednostjo zaužite hrane.
- prihranek časa – če bi izdelke, ki nam jih je zmanjkalo, kar preko hladilnika skupaj z dostavo naročili v spletni trgovini, bi s tem privarčevali tudi nekaj časa. Poleg tega je iskanje ali brskanje po receptih na podlagi zaloge dosti bolj pregledno in časovno učinkovito, saj nam ni treba pregledovati velikega števila receptov, za katere pri branju potrebnih sestavin ugotovimo, da jih sploh nimamo.
- večpredstavnostna uporabniška izkušnja – zaradi vgrajenega ekrana in povezave z internetom se odprejo nove možnosti predvajanja večpredstavnostnih vsebin in drugih aktivnosti, ki jih sicer opravljamo na računalniku: od branja novic, poslušanja glasbe, gledanja video vsebin, kuharskih oddaj, elektronskega sporočanja, itd.

# 3. Uporabljena orodja in tehnologije

## 3.1. Tehnologija RFID

RFID je kratica za identifikacijo z radijskimi valovi (Radio Frequency IDentification). Izraz se uporablja za sisteme identifikacije, v katerih imajo predmeti vgrajeno ali pripeto elektronsko napravo, ki za komunikacijo uporablja radijsko frekvenco ali magnetno polje. Tej elektronski napravi rečemo značka RFID (ang. tag) in je ena od bistvenih komponent sistema RFID. V grobem se značke RFID ločijo glede na način napajanja, tako da poznamo pasivne in aktivne značke:

- Pasivne značke nimajo lastnega napajanja in se aktivirajo šele, ko se nahajajo v območju čitalnika, ki oddaja elektromagnetne valove. So manjše in cenejše, a imajo krajši domet in so manj odporne na napake.
- Aktivne značke imajo lasten vir napajanja in zato večjo moč oddajanja, daljši domet, so bolj zanesljive, a večje in dražje.

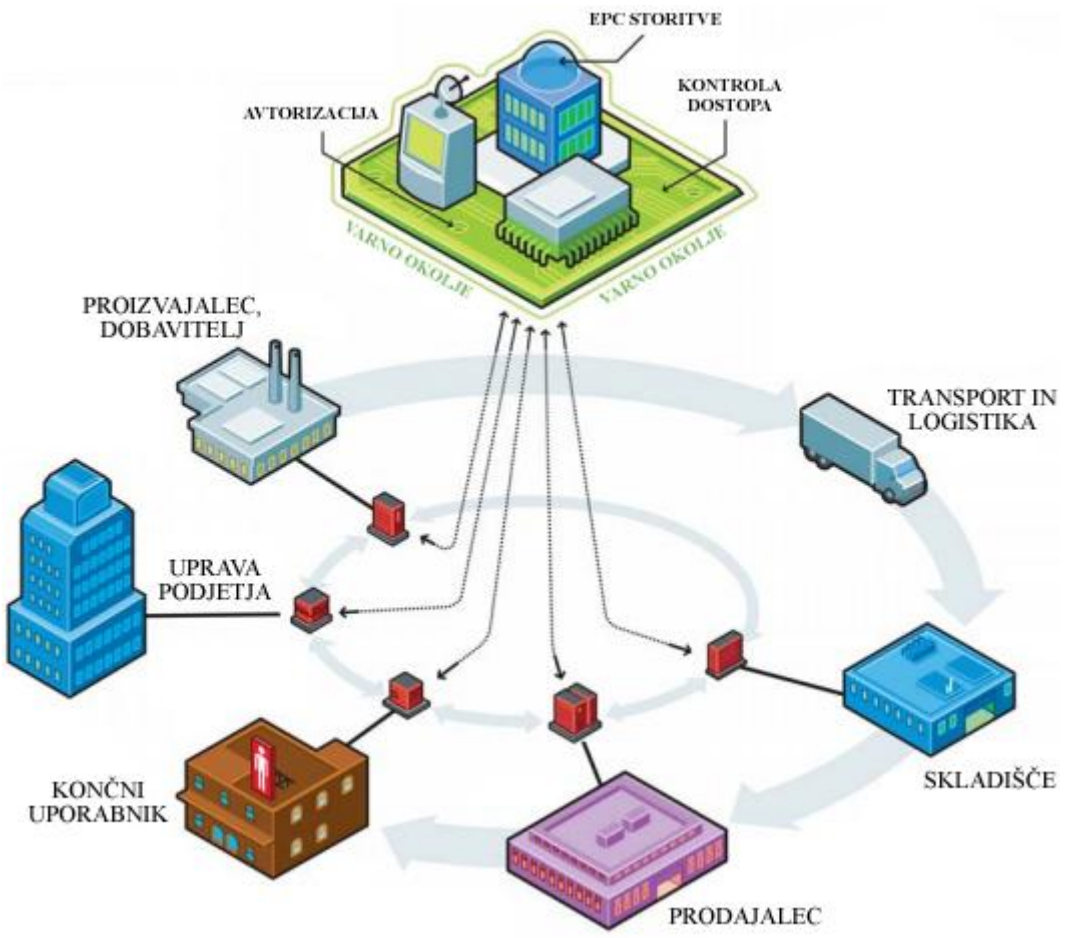
Druga bistvena komponenta sistema RFID pa je čitalnik, to je elektronska naprava, ki lahko zazna prisotnost značk RFID in prebere na njih shranjene podatke ali pa zapiše podatke na te značke. Čitalnik lahko nato obvešča druge sisteme o prisotnosti označenih predmetov. Ti sistemi za komunikacijo s čitalniki po navadi uporabljajo posebno programsko opremo, ki je vmesni člen med čitalniki in aplikacijami in ji zato rečemo vmesna programska oprema (ang. middleware). Predstavljeni sistem RFID je prikazan na sliki 2 [1].

*Slika 2. Zgradba tipičnega sistema RFID.*

Viri navajajo, da lahko začetke uporabe pasivne tehnologije RFID zaznamo že v času druge svetovne vojne, prvi patent za aktivno značko RFID pa je star okoli petintrideset let [9]. Tehnologija se je začela bolj množično uporabljati v zadnjih desetih letih. Z zniževanjem cen značk in razvojem tehnologij se je pocenila tudi uporaba sistemov RFID za različne namene:

- za plačevanje javnega transporta (avtobusi, železnice),
- za sledenje izdelkov in opreme,
- za identifikacijo živali,
- za merjenje rezultatov na športnih tekmovanjih,
- za določanje lokacije knjig v knjižnicah, za samostojno izposojo knjig,
- za preprečevanje tatvin v trgovinah, itd.

Poleg že uveljavljenih se pojavljajo in razvijajo vedno nove možnosti uporabe, kjer bo ena od najbolj zanimivih nadomestitev črtne kode. Obstaja kodirna shema, imenovana elektronska koda produkta (EPC – Electronic Product Code), ki je bila razvita za namen poceni sledenja objektov z uporabo tehnologije RFID. Razvil jo je konzorcij preko sto dvajsetih svetovnih korporacij in univerzitetnih laboratorijev. Koda EPC se uporablja za enolično identifikacijo produktov in ne vsebuje drugih informacij o produktu kot npr., kaj ta produkt je, kdo je njegov proizvajalec ali kakšna je njegova življenjska doba [10]. Ko se s čitalnikom koda prebere, se lahko sistem preko interneta poveže s sistemom EPC in dobi ali posreduje podatke o produktu. Na ta način lahko vsak, ki vzpostavi stik s produktom v shemi na sliki 3, pride do zanj pomembnih informacij o samem produktu, ki so mu dostopne v omrežju "EPCglobal Network", in jih potem po želji uporabi na različne načine [2].




*Slika 3. Shema omrežja EPCglobal.*

Z uporabo tehnologije RFID lahko proizvajalec v realnem času sledi svojim proizvodom ali njihovim sestavnim delom v proizvodnem procesu. Dobavitelj za vsak izdelek natančno ve, kdaj je v fazi transporta in kdaj je dostavljen v skladišče. V skladišču lahko skladiščnik vsak trenutek natančno določi lokacijo posameznega izdelka. Prodajalec lahko poleg lokacije izdelkov na policah sledi tudi prodaji izdelkov.

V trgovini bi se lahko na blagajni v hipu zaračunalo kupcu vse izdelke, saj ne bi bilo več potrebno branje črtne kode vsakega izdelka posebej, ampak bi se celotna vsebina košarice prebrala naenkrat, ko bi se premikala mimo čitalnika. Končni uporabnik bi lahko podatke o izdelkih uporabil doma v pametni hiši, kar bi mu olajšalo ponavljajoča se opravila in s tem izboljšalo kvaliteto življenja. Uprave vseh podjetij, vključenih v to oskrbovalno verigo, bi lahko vse pridobljene podatke uporabile za analize in izboljšave svojih poslovnih procesov, v inteligentnem hladilniku pa bi informacije o prehrambnih izdelkih uporabljali za ugotavljanje zalog in napovedi jedilnika, itd.

## 3.2. Podatkovna baza PostgreSQL

Podatkovna baza PostgreSQL je zmogljiva in odprtokodna podatkovna baza. V svojih petnajstih letih razvoja si je priborila ugled zaradi zanesljivosti, celovitosti podatkov in pravilnosti delovanja. Deluje v številnih operacijskih sistemih, vključno z Windows, Linux, MAC OS X, UNIX in drugimi [21]. Je popolnoma skladna z modelom ACID (ang. atomicity, consistency, isolation, durability), kar pomeni, da ima vse štiri naštete lastnosti (atomarnost, konsistentnost, izolacijo in trajnost), ki zagotavljajo zanesljivo izvedbo transakcij v podatkovnih bazah [4].

PostgreSQL vključuje večino podatkovnih tipov SQL92 in SQL99, podpira shranjevanje velikih binarnih objektov, ima vmesnike za programske jezike C/C++, Java, .Net, Perl, Python, Ruby, ODBC in je obširno dokumentirana. Izvorna koda PostgreSQL je na voljo pod licenco BSD, ki je najbolj liberalna odprtokodna licenca. Dopušča namreč uporabo, spreminjanje in razširjanje PostgreSQL v kakršnikoli obliki [21].

## 3.3. Tehnologija Java

Tehnologija Java je objektno orientirano večnitno programsko okolje, neodvisno od računalniškega okolja (ang. platform), v katerem ga uporabljamo. Je temelj spletnih in mrežnih storitev, aplikacij in namizij, ki lahko tečejo v kateremkoli okolju, je tudi temeljna tehnologija na področju robotike in drugih namenskih naprav.

Programski jezik je dobro dokumentiran, napisanih je veliko knjižnic, poleg tega pa je napisan program prenosljiv in se izvaja v različnih operacijskih sistemih. Razvijalci lahko kombinirajo ponovno uporabne javanske objekte za hitro kreiranje unikatnih in posebej prikrojenih strežniških aplikacij, kot so npr. spletni forumi, trgovine, ankete, obdelava obrazcev HTML,

itd. Uporaba programskega jezika Java in jezika XML je primerna za okolje, kjer tečejo spletne storitve, saj sta namreč ena od bolj razširljivih in široko sprejetih računalniških jezikov [22].

Programsko kodo sem pisal v programskem jeziku Java. Namestil sem računalniško okolje Java SE 6 (Java Platform, Standard Edition 6). Okolje vsebuje Java SE Development Kit 6 (JDK 6) in Java SE Runtime Environment 6 (JRE 6).

Vsaka različica JDK vsebuje [23]:

- prevajalnik Java,
- Java navidezni stroj (ang. Java virtual machine),
- knjižnice Java,
- pregledovalnik Java programčkov (ang. Java AppletViewer),
- razhroščevalnik in
- dokumentacijo.

Za povezavo s podatkovno bazo je uporabljen standardni programski vmesnik JDBC, ki omogoča povezljivost programskega jezika Jave s podatkovno bazo neodvisno od izbire podatkovne baze – lahko je SQL baza ali drugi viri, v katerih so podatki ločeni z ločevalcem (ang. delimiter), kot so tabele ali navadne tekstovne datoteke.

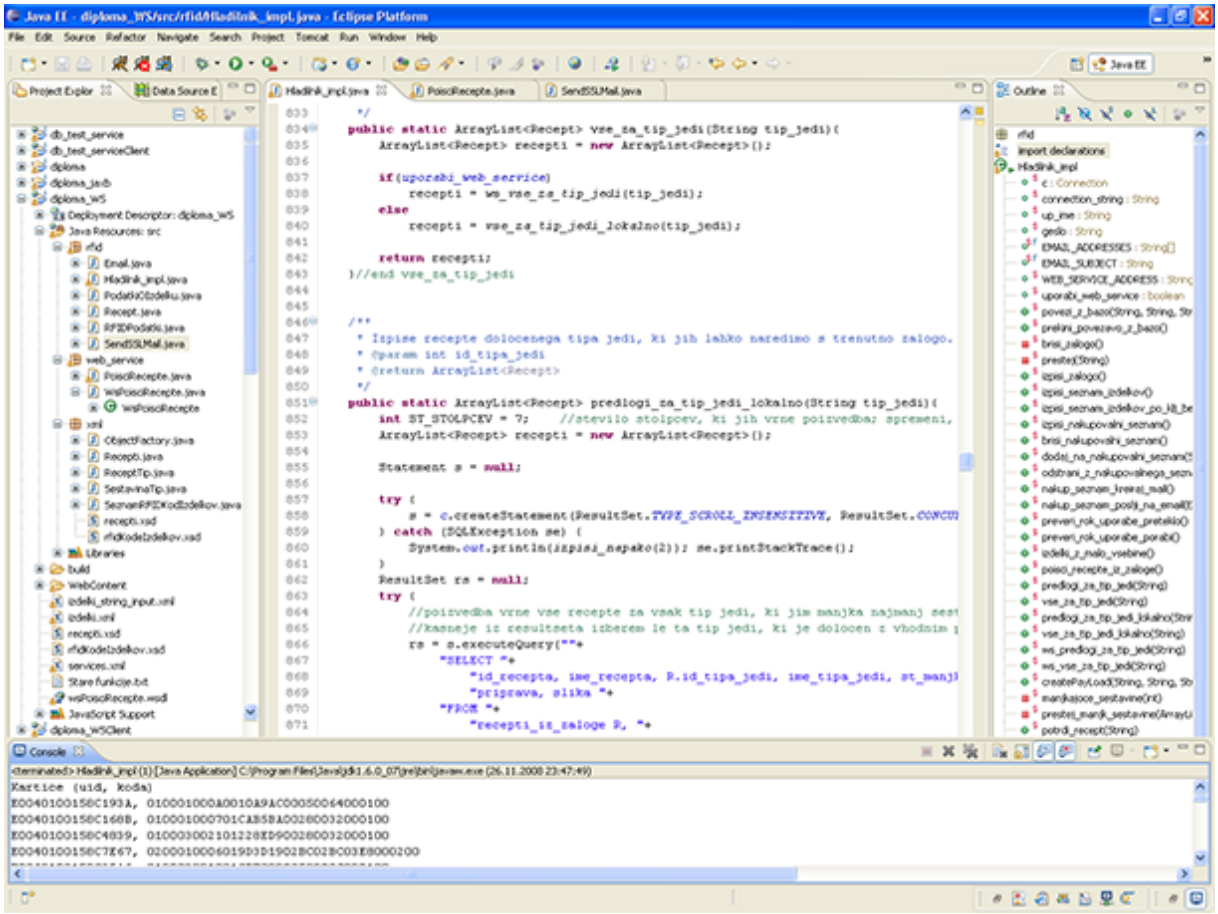
## 3.4. Razvojno okolje Eclipse

Eclipse je odprtokodno razvojno okolje, primarno namenjeno razvijalcem v okolju Java [24]. Za diplomsko nalogo sem uporabljal Eclipse IDE za Java EE razvijalce verzijo 3.4.0, ki je del paketa Ganymede.

Za delovanje Eclipsa je potrebno imeti nameščeno izvajalno okolje Jave (angl. Java Runtime Environment), za samo programiranje v Javi pa še Java JDK. Eclipse deluje v operacijskih sistemih Windows, Linux, MAC OS X in drugih.

Dobra lastnost okolja Eclipse, prikazanega na sliki 4, je, da je izjemno razširljiv in odprtokoden. Zanj obstajajo številni dodatki (ang. plugins), ki jih razvijajo posamezniki in podjetja zastonj ali proti plačilu. Dodatki dajejo razvojnemu okolju nove zmožnosti, mu povečujejo zmogljivost, olajšujejo programiranje in pospešijo proces razvijanja programske opreme.

Namestitev Eclipsa je preprosta, saj le razširimo stisnjeno zip datoteko v poljubno mapo na disku in zaženemo izvršljivo datoteko.



*Slika 4. Posnetek razvojnega okolja Eclipse*

## 3.5. Apache Tomcat

Apache Tomcat je aplikacijski strežnik, ki podpira računalniško okolje Java. Omogoča izvajanje spletnih programov (ang. servlet) Java in strani JSP (ang. JavaServer Pages) ter zagotavlja "čisto Java" http strežniško okolje, na katerem lahko teče koda, napisana v Javi. Uporabniki imajo prost dostop do izvorne kode Tomcat-a pod licenco Apache.

Lastnosti strežnika Tomcat verzije 6.0.18, ki sem jo uporabil pri diplomski nalogi, sta:

- podpira specifikacije Servlet 2.5 in JSP 2.1,
- zasnovan je tako, da se izvaja na različici Java SE 5 ali kasnejši.

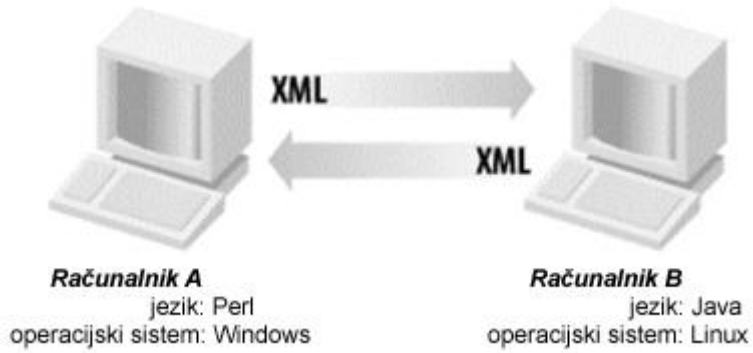
## 3.6. Spletna storitev

Spletna storitev je storitveno orientirana arhitektura, za katero velja naslednje [16]:

- vmesniki spletnih storitev so opisani z jezikom WSDL (Web Service Description Language),
- za prenašanje sporočil med spletno storitvijo in odjemalcem se uporablja protokol SOAP preko HTTP,
- kot imenik storitev je lahko uporabljen UDDI.

Možne so tudi druge kombinacije tehnologij, a je ta najpogostejša in zato pogosto razlog, da se izraza "storitveno orientirana arhitektura" in "spletne storitve" uporabljata kot sinonima.

Lahko bi rekli tudi, da je spletna storitev katera koli storitev, ki je dostopna na spletu, uporablja standardiziran sistem sporočanja v obliki XML in ni vezana na določen operacijski sistem ali programski jezik (slika 5) [3].



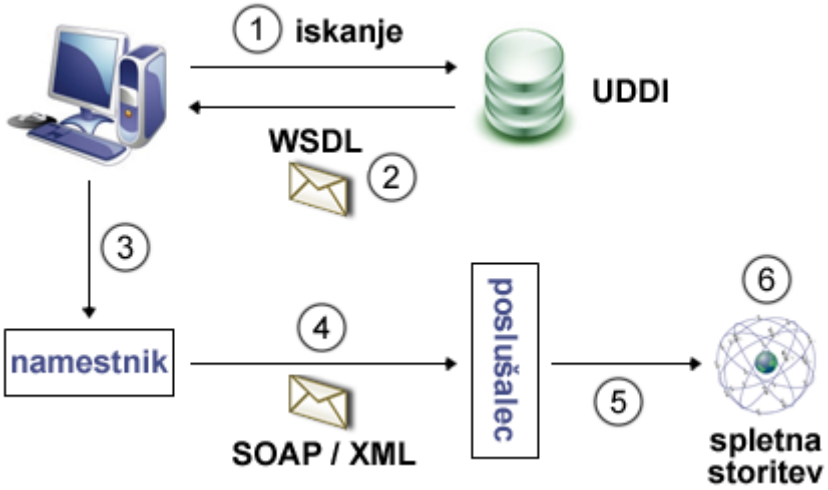

*Slika 5. Shema enostavne spletne storitve*

Težko je najti definicijo spletne storitve, za katero bi se vsi strinjali, da je dobra. Konzorcij W3C, ki je vodilna organizacija mednarodnih standardov za svetovni splet, definira spletno storitev kot "programski sistem, zasnovan tako, da podpira interakcije medsebojno povezljivih računalnikov preko omrežja" [20].

Na sliki 6 je prikazan procesni tok spletne storitve, kjer si procesi sledijo v naslednjem zaporedju [19]:

1. Iskanje spletne storitve v imeniku UDDI.
2. Opis izbrane spletne storitve se kot datoteka WSDL prenese do aplikacije odjemalca.
3. Kreiranje namestnika (ang. proxy), ki preslika spletno storitev v programski objekt našega okolja s pripadajočimi metodami in lastnostmi glede na WSDL opis.

4. Kreiranje sporočila SOAP/XML, ki se pošlje na naslov URL, ki je definiran v WSDL datoteki.
5. Poslušalec SOAP (ang. listener) na strani gostitelja sprejme klic in ga interpretira spletni storitvi.
6. Spletna storitev opravi svojo funkcijo in vrne rezultat odjemalcu preko poslušalca in namestnika.




*Slika 6. Potek procesov pri spletni storitvi.*

### 3.6.1. XML

XML (Extensible Markup Language) je bil sprva predstavljen kot "izboljšani HTML", a je kmalu postalo jasno, da je zelo primeren kot zapis za izmenjavo podatkov. Tako je XML postal standard za izmenjevanje podatkov preko spleta. Z njim lahko opišemo katere koli strukturirane ali pol-strukturirane podatke. Namesto uporabe vnaprej definiranih podatkovnih struktur za opis podatkov XML omogoča definiranje lastnih podatkovnih struktur za vsako problemsko domeno posebej [5].

XML je zapis v tekstovni obliki. Čeprav zavzame več prostora kot binarni formati, je postal XML hitro splošno sprejet, še posebej priljubljen je postal v elektronskem poslovanju v poznih devetdesetih letih. Z uporabo jezika XML so lahko razvijalci pripisali pomen kateri koli informaciji, ki je potovala preko spletnih protokolov [7]. Vsa računalniška okolja so imela razčlenjevalnike kode (ang. parser) XML. Nastala je uspešna kombinacija protokola (HTTP) in zapisa (XML), ki so ju lahko za komunikacijo uporabljale aplikacije v vseh računalniških okoljih. Spletne storitve so postale pogovorni jezik pri povezavi aplikacij preko interneta.

### 3.6.2. WSDL

Spletne storitve morajo biti dostopne drugim aplikacijam. Ena od posledic te zahteve je, da mora imeti spletna storitev dobro definiran vmesnik, ki je določen z dokumentom WSDL (Web Service Description Language) [5]. Dokument WSDL je zapisan v obliki XML in opisuje vse podrobnosti spletne storitve, ki jih potrebujejo odjemalci za njeno uporabo, med drugim [11]:

- URL spletne storitve,
- komunikacijske mehanizme, ki jih razume,
- operacije, ki jih izvaja,
- strukturo sporočil.

WSDL pa ni uporaben le kot vmesnik spletnih storitev, ampak lahko njegovo rabo razširimo, tako da njegovo natančno formalno definicijo izkoriščajo orodja za razvoj spletnih storitev za izvajanje operacij, kot so:

- generiranje nastavkov (ang. stubs) odjemalcev in storitev v različnih programskih jezikih,
- objava spletne storitve,
- dinamično testiranje spletne storitve.

WSDL je sestavljen iz množice elementov in njihovih lastnosti, ki opisujejo spletno storitev [12]. V diplomski nalogi je uporabljena verzija WSDL 1.1, ki je prikazana na sliki 7 [14].

```
<definitions>
<types>
    definicija podatkovnih tipov...
</types>
<message>
    definicija sporočil...
</message>
<portType>
    definicija vrat...
</portType>
<binding>
    definicija povezovanja...
</binding>
<service>
    definicija storitve...
</service>
</definitions>
```

*Slika 7. Struktura dokumenta WSDL 1.1.*

Storitev definira šest važnejših elementov:

Element `<definitions>` je korenski element vsakega dokumenta WSDL. Služi kot vsebnik (ang. container), ki vsebuje vse potrebne informacije o storitvi in njenih atributih. Definira imenski prostor (ang. namespace) in določa predpone imenskih prostorov.

Element `<types>` definira podatkovne tipe, ki se uporabljajo v sporočilih, ki se pošiljajo in sprejemajo. Ponavadi so podatkovni tipi definirani v obliki sheme. Priporoča se shema W3C XML, podprte pa so tudi druge sheme kot npr. enostavnejša DTD.

Element `<message>` predstavlja abstraktno definicijo podatkov, ki se prenašajo. Sporočilo je sestavljeno iz več logičnih delov, vsak od njih je povezan z definicijo znotraj določene vrste sistema.

Element `<portType>` je abstrakten opis funkcionalnosti in vsebuje množico abstraktnih operacij in abstraktnih sporočil. Vsaka operacija ima definirano ime, vhodno in izhodno sporočilo in sporočila o napakah.

Element `<binding>` definira protokol transporta in definira format sporočil za operacije in sporočila, definirana v določenem elementu `<portType>`.

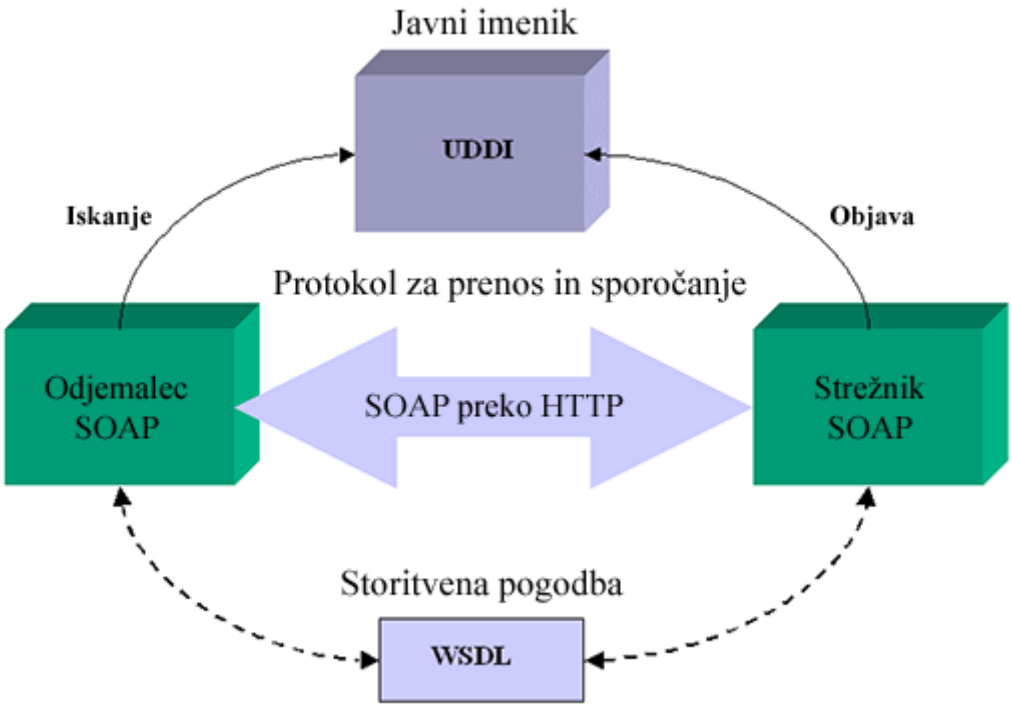
Element `<service>` razvršča množico med seboj povezanih elementov `<port>`. Storitev vsebuje množico končnih točk `<endpoint>`, vsaka končna točka pa vsebuje omrežni naslov in referenco na `binding`. Vse ostale, za vsak protokol specifične informacije, so zapisane v samem elementu `binding` [12, 13].

### 3.6.3. SOAP

Protokol SOAP (Simple Object Access Protocol) je specifikacija za izmenjavo strukturiranih informacij pri implementacijah spletnih storitev v računalniških omrežjih.

Kot oblika sporočil se uporablja XML, za pogajanja in prenos sporočil pa se ponavadi uporabljajo drugi protokoli aplikacijskega sloja, predvsem HTTP in RPC (Remote Procedure Call). SOAP je temeljni sloj kopice protokolov spletnih storitev in zagotavlja ogrodje za osnovno sporočanje, na katerem se lahko gradijo drugi abstraktni protokoli. Na sliki 8 je prikazana funkcija protokola SOAP v spletni storitvi.

Arhitekturo SOAP sestavlja več slojev, ki določajo obliko sporočil, vzorce izmenjave sporočil (ang. Message Exchange Patterns), povezovanje osnovnih transportnih protokolov, modelov obdelave sporočil in razširljivosti protokolov.




*Slika 8. Funkcija protokola SOAP v spletni storitvi.*

Prednosti protokola SOAP:

- Ker uporablja protokol HTTP, je komuniciranje skozi namestniške strežnike (ang. proxies) in požarne zidove lažje kot pri starejših tehnologijah oddaljenega izvajanja.
- Je dovolj prilagodljiv, da dovoljuje uporabo različnih transportnih protokolov (HTTP, SMTP).
- Neodvisen je od računalniškega okolja.
- Neodvisen je od uporabljenega programskega jezika.
- Je preprost in razširljiv.

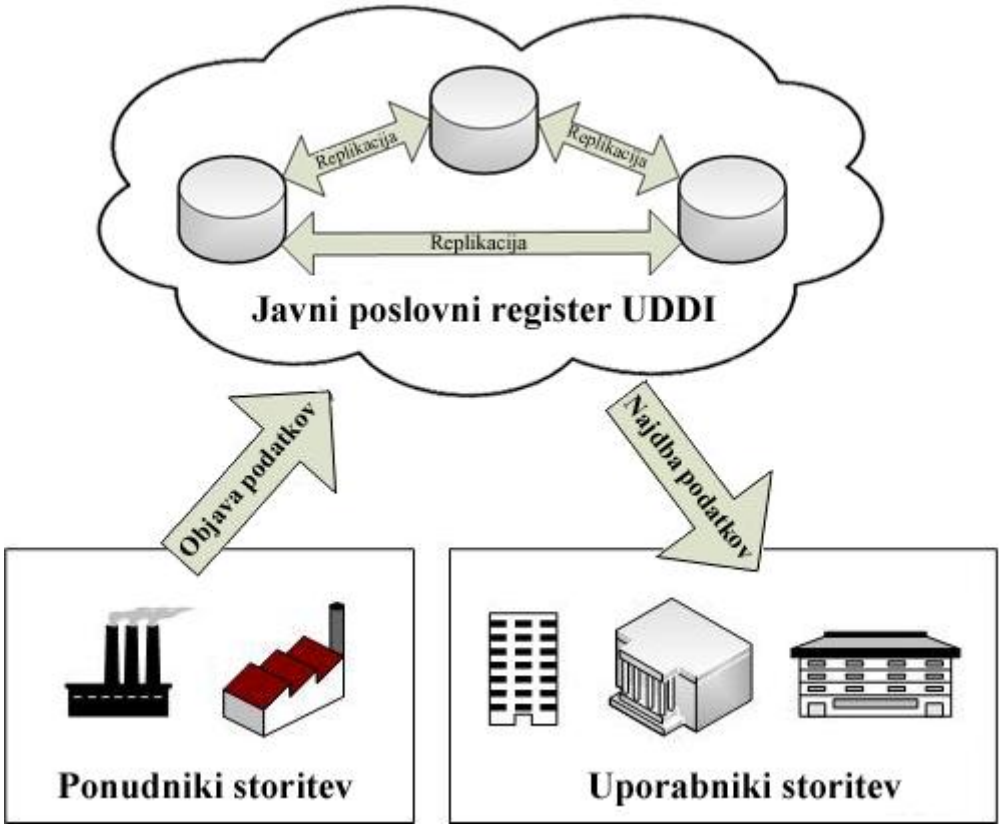
Slabosti:

- Zaradi uporabe XML je lahko občutno počasnejši od konkurenčnih tehnologij vmesne opreme kot npr. CORBA (Common Object Request Broker Architecture). To se pozna predvsem pri daljših sporočilih [18].
- Zaradi uporabe protokola HTTP mora požarni zid, ki omogoča le brskanje po spletu, izvajati natančnejšo (in zato manj učinkovito) analizo paketov, ker ne more kar prepuščati vseh paketov, ki uporabljajo HTTP.

### 3.6.4. UDDI

UDDI (Universal Description, Discovery and Integration) je specifikacija za kreiranje registra storitev, ki katalogizira organizacije in njihove spletne storitve. Najnovejša različica specifikacije UDDI je trenutno 3.0.2, potrjen pa je bil pri konzorciju OASIS (Organization for the Advancement of Structured Information Standards) leta 2005. V uvodu specifikacije so zapisali, da dobijo spletne storitve pomen šele takrat, ko morebitni uporabniki lahko najdejo informacije, ki jih potrebujejo za njihovo izvajanje [15].

Implementacija specifikacije UDDI se imenuje register UDDI. Register UDDI je podatkovna baza, ki podpira množico standardnih podatkovnih struktur, ki so definirane v specifikaciji UDDI. Podatkovne strukture oblikujejo model informacij o organizacijah (poslovnih enotah, vladnih agencijah, itd.) in tehničnih zahtevah za dostop do spletnih storitev, ki jih te organizacije nudijo. Po registru UDDI lahko iščemo določena podjetja ali spletne storitve, prav tako pa lahko v registru objavimo podatke o lastnem podjetju ali svojo spletno storitev (slika 9). Do podatkov v imeniku UDDI lahko dostopamo z uporabo protokola SOAP, saj je tudi sam register UDDI spletna storitev [6].




*Slika 9. Shema javnega poslovnega registra UDDI.*

# 4. Zasnova in izvedba koncepta napovedovanja

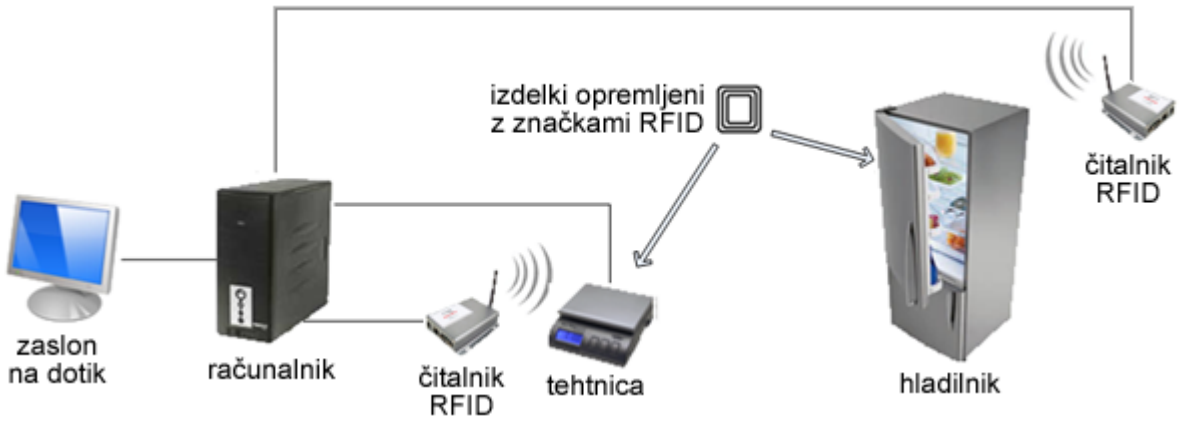
V okviru diplomskega dela sem izdelal aplikacijo za napovedovanje jedilnika in nakupovalne liste v prototipu sistema inteligentnega hladilnika. Sistem z uporabo tehnologije RFID zazna prehrambne izdelke, ki so opremljeni z značko RFID. Na značkah so zapisani podatki o posameznem izdelku. Ti podatki se preberejo in zapišejo v bazo, od koder jih aplikacija prebere, obdela in prikaže v grafičnem uporabniškem vmesniku kot zalogo izdelkov. Aplikacija predlaga kuharske recepte, ki jih lahko naredimo z izdelki iz zaloge, zapisuje težo izdelkov in računa njihovo količino, omogoča ustvarjanje nakupovalnega seznama in opozarja uporabnika na določene lastnosti izdelkov.

Razvoj in izvedba aplikacije sta vključevala naslednje naloge:

- definicijo zapisa podatkov o izdelkih na značkah RFID,
- izdelavo podatkovne baze za shranjevanje podatkov o živilih in receptih, določitev ustreznih šifrantov za izdelke, tipe jedi, merske enote in tabele za shranjevanje zgodovine zaloge in izborov receptov,
- zapis funkcij za delovanje aplikacije,
- zapis spletne storitve, ki ji pošljemo seznam izdelkov, ki jih imamo na zalogi, storitev pa nam vrne seznam receptov, ki jih lahko s temi izdelki uporabimo,
- vključitev aplikacije v prototip inteligentnega hladilnika in testiranje v realnem sistemu.

## 4.1. Opis sistema inteligentnega hladilnika

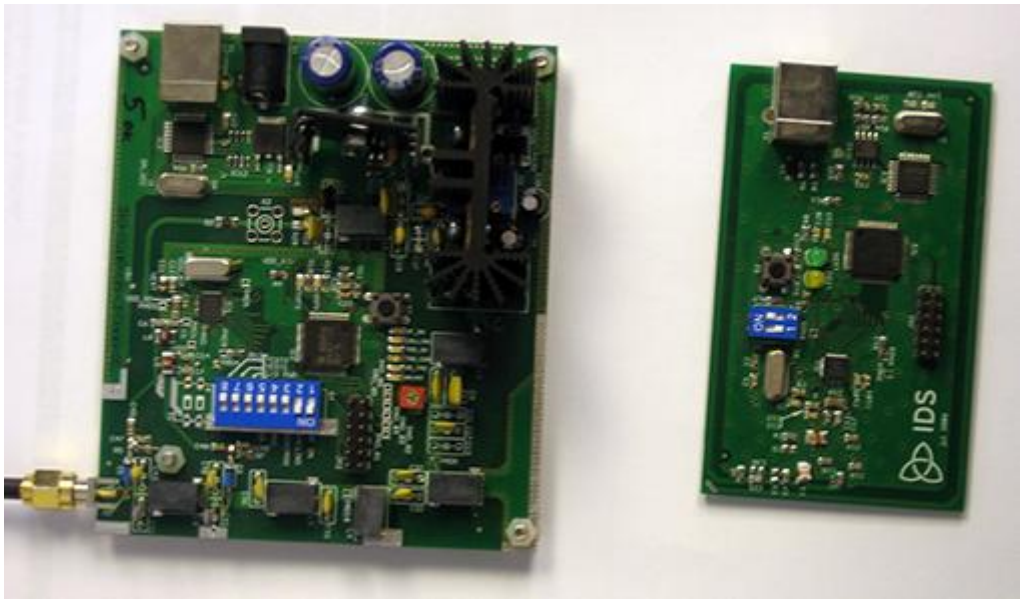
Slika 10 prikazuje shemo prototipa sistema vodenja vsebine inteligentnega hladilnika, prototip sistema pa je prikazan na sliki 11. Sestavljen je iz dveh čitalnikov RFID s pripadajočima antenama, značk RFID, tehtnice, osebnega računalnika, zaslona na dotik in programske opreme.




*Slika 10. Shema prototipa sistema inteligentnega hladilnika.*

*Slika 11. Sistem inteligentnega hladilnika.*

Za branje podatkov smo uporabljali dva čitalnika RFID, ki sta priključena na računalnik preko USB, delujeta pa na frekvenci 13,56 MHz. S čitalnikom na sliki 12 a), ki ima močnejšo anteno in zato večji domet (r = 50 cm), smo brali podatke s kartic, ki so predstavljale zalogo izdelkov. Z manjšim čitalnikom na sliki 12 b) (r = 10 cm) smo prebirali podatke z izdelka, ki smo ga tehtali in nanj zapisovali novo težo. Tehtnica je bila priključena na računalnik preko serijskega priključka RS232.




*a) r = 50 cm* *b) r = 10 cm*

*Slika 12. Čitalnika RFID za branje podatkov s kartic in zapisovanje teže na kartice.*

## 4.2. Značka RFID in struktura podatkov

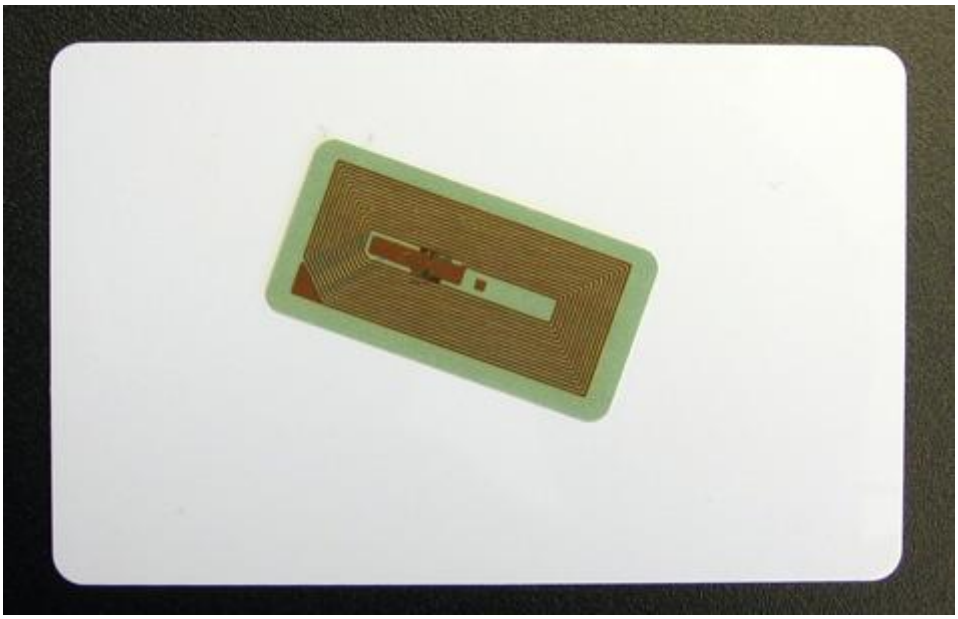
Značke RFID vsebujejo podatke o živilih, ki so shranjena v hladilniku ali bližnji omari. Čeprav je govora le o inteligentnem hladilniku, se vse podane specifikacije lahko uporabijo tudi za prehrambne izdelke, ki so shranjeni na drugem mestu.

V aplikaciji smo uporabili značke v obliki kartic znamke Philips SL2 ICS20, SLI RF (ISO/IEC 15693) in nalepke (slika 13) z naslednjimi značilnostmi:

- pasivna značka, ki ne potrebuje napajanja;
- deluje na razdalji do 1,5 m oz. do 0,5 m v povezavi s čitalcem (na sliki 12 a) in anteno;
- hitrost prenašanja podatkov: do 53 kbit/s;
- frekvenca delovanja: 13,56 MHz.

Vsaka značka ima pomnilnik EEPROM z naslednjimi značilnostmi:

- velikost: 1024 bitov, organiziranih v 32 blokov po 4 bajte. Prvih 128 bitov je namenjenih unikatnemu identifikatorju (64 bitov), posebnim podatkom (EAS, AFI) in kontroli dostopa pisanja, ostalih 896 bitov pa je namenjenih uporabniškim podatkom;
- predviden čas hrambe podatkov: 10 let;
- število pisalnih ciklov: 100.000;
- zaščita pisanja za vsak uporabnikov pomnilniški blok.



*Slika 13. Znački RFID v obliki kartice (spodnja bela) in nalepke (zgornja sivo–rjava)*

Za zapis podatkov v spomin značke smo definirali kodirno tabelo. Za to obstaja več razlogov, najvažnejši pa je ta, da se s tem zmanjša količina podatkov, ki jih je treba prenesti. Vsak bajt

podatka na kartici je zapisan z dvema šestnajstiškima znakoma (1B = XX). V vsak blok pomnilnika EEPROM se zapiše štiri bajte oz. osem šestnajstiških znakov. Pri branju pomnilnika je potrebno vedno prebrati cel blok.

Definirali smo dva tipa zapisa podatkov.

1. Zapis A za izdelke, katerih polna neto količina je na embalaži zapisana v enotah za merjenje mase (g). Taki izdelki so bili na primer: maslo, sladkor, goveje meso, sir, brokoli, korenje, itd. Zapis A je dolg 16 bajtov (32 šestnajstiških znakov) oz. štiri bloke pomnilnika EEPROM.
2. Zapis B pa za izdelke, katerih polna neto količina je na embalažah zapisana v enotah za merjenje volumna (ml) ali v številu kosov (kos) izdelka. Primer takih izdelkov: jajca, kis, olivno olje, sladka smetana, mleko, itd. Zapis B je dolg 18 bajtov oz. 36 šestnajstiških znakov. Ker se v blok pišejo po štirje bajti in se iz pomnilnika vedno bere cel blok, se zadnjemu podatku, ki zasede polovico petega bloka, dopišejo štiri ničle, tako da zapis B zavzema pet blokov pomnilnika.

V *tabeli 1* in *tabeli 3* sta predstavljeni shemi zapisa A in zapisa B. Na shemah je prikazano, v kateri blok EEPROM-a se posamezen parameter zapiše. Parametri so označeni s številkami, da se lahko v *tabeli 2* in *tabeli 4* razbere njihov pomen in njihova dolžina v bajtih.

| št. bloka EEPROM-a | 1. blok | | | 2. blok | | 3. blok | | | 4. blok | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| šestnajstiški znaki | XX | XXXX | XX | XX | XXXXXX | XX | XXXX | XX | XX | XXXX | XX |
| parameter | 1 | 2 | 3 | 4 | | 5 | 6 | | 7 | 8 | |

*Tabela 1. Oblika zapisa A.*

| parameter | št. bajtov | pomen parametra |
|---|---|---|
| 1 | 1 | tip zapisa |
| 2 | 2 | id tipa izdelka |
| 3 | 2 | id izdelka |
| 4 | 4 | datum poteka uporabe v obliki ddmmllll |
| 5 | 2 | stehtana bruto teža izdelka |
| 6 | 2 | teža izdelka (napisana na embalaži, ...) |
| 7 | 2 | id merske enote |
| 8 | 1 | teža embalaže |

*Tabela 2. Pomen posameznih parametrov zapisa A.*

<table>
<tr><th>št. bloka EEPROM</th><th colspan="3">1. blok</th><th colspan="2">2. blok</th><th colspan="3">3. blok</th><th colspan="3">4. blok</th><th colspan="3">5. blok</th></tr>
<tr><td>šestnajstiški znaki</td><td colspan="3">XX XXXX XX</td><td colspan="2">XX XXXXXX</td><td colspan="3">XX XXXX XX</td><td colspan="3">XX XXXX XX</td><td colspan="3">XX XX 0000</td></tr>
<tr><td>parameter</td><td>1</td><td>2</td><td>3</td><td colspan="2">4</td><td>5</td><td colspan="2">6</td><td colspan="2">7</td><td colspan="2">8</td><td>9</td><td></td></tr>
</table>

*Tabela 3. Oblika zapisa B.*

| parameter | št. bajtov | pomen parametra |
|---|---|---|
| 1 | 1 | tip zapisa |
| 2 | 2 | id tipa izdelka |
| 3 | 2 | id izdelka |
| 4 | 4 | datum poteka uporabe v obliki ddmmllll |
| 5 | 2 | stehtana bruto teža izdelka |
| 6 | 2 | teža izdelka (napisana na embalaži, ...) |
| 7 | 2 | količina izdelka (napisana na embalaži, ...) |
| 8 | 2 | id merske enote |
| 9 | 1 | teža embalaže |

*Tabela 4. Pomen posameznih parametrov zapisa B*

Na značke smo zapisali podatke o izdelkih v enem od zgoraj naštetih zapisov (zapis A ali zapis B). Tabela 5 prikazuje tipe izdelkov, ki smo jih v aplikaciji definirali.

| Tip izdelka | Opis | Izdelki |
|---|---|---|
| 1 | Osnovna živila | 1 – 10 |
| 2 | Meso in mesni izdelki | 11 – 16 |
| 3 | Mleko in mlečni izdelki | 17 – 21, 41, 42 |
| 4 | Sadje in zelenjava | 22 – 35, 39, 40 |
| 5 | Žitarice in mlevski izdelki | 36 – 38 |

*Tabela 5. Tipi izdelkov.*

V tabeli 6 so predstavljeni vsi izdelki, ki se lahko nahajajo v hladilniku.

| Izdelek | Opis | Izdelek | Opis | Izdelek | Opis |
|---|---|---|---|---|---|
| 1 | Jajca | 15 | Ribe | 29 | Krompir |
| 2 | Jušna kocka | 16 | Šunka | 30 | Limona |
| 3 | Kis | 17 | Kisla smetana | 31 | Paprika |
| 4 | Kvas | 18 | Mleko | 32 | Paradižnik |
| 5 | Maslo | 19 | Sir | 33 | Peteršilj |
| 6 | Olivno olje | 20 | Skuta | 34 | Por |
| 7 | Poper | 21 | Sladka smetana | 35 | Šampinjoni |
| 8 | Rastlinsko olje | 22 | Brokoli | 36 | Moka |
| 9 | Sladkor | 23 | Bučke | 37 | Riž |
| 10 | Sol | 24 | Cvetača | 38 | Testenine |
| 11 | Goveje meso | 25 | Čebula | 39 | Jabolka |
| 12 | Mleto meso | 26 | Česen | 40 | Pomaranče |
| 13 | Piščančje meso | 27 | Grah | 41 | Sadni jogurt |
| 14 | Puranje meso | 28 | Korenje | 42 | Navadni jogurt |

*Tabela 6. Izdelki*

Tabela 7 definira tri različne tipe jedi, ki jih lahko predlagamo v povezavi z zalogami.

| Id tipa jedi | Opis |
|---|---|
| 1 | Predjed |
| 2 | Juha |
| 3 | Glavna jed |

*Tabela 7. Tipi jedi*

Tabela 8 označuje merske enote, v katerih so izdelki.

| Id merske enote | Opis |
|---|---|
| 1 | g |
| 2 | ml |
| 3 | kos |

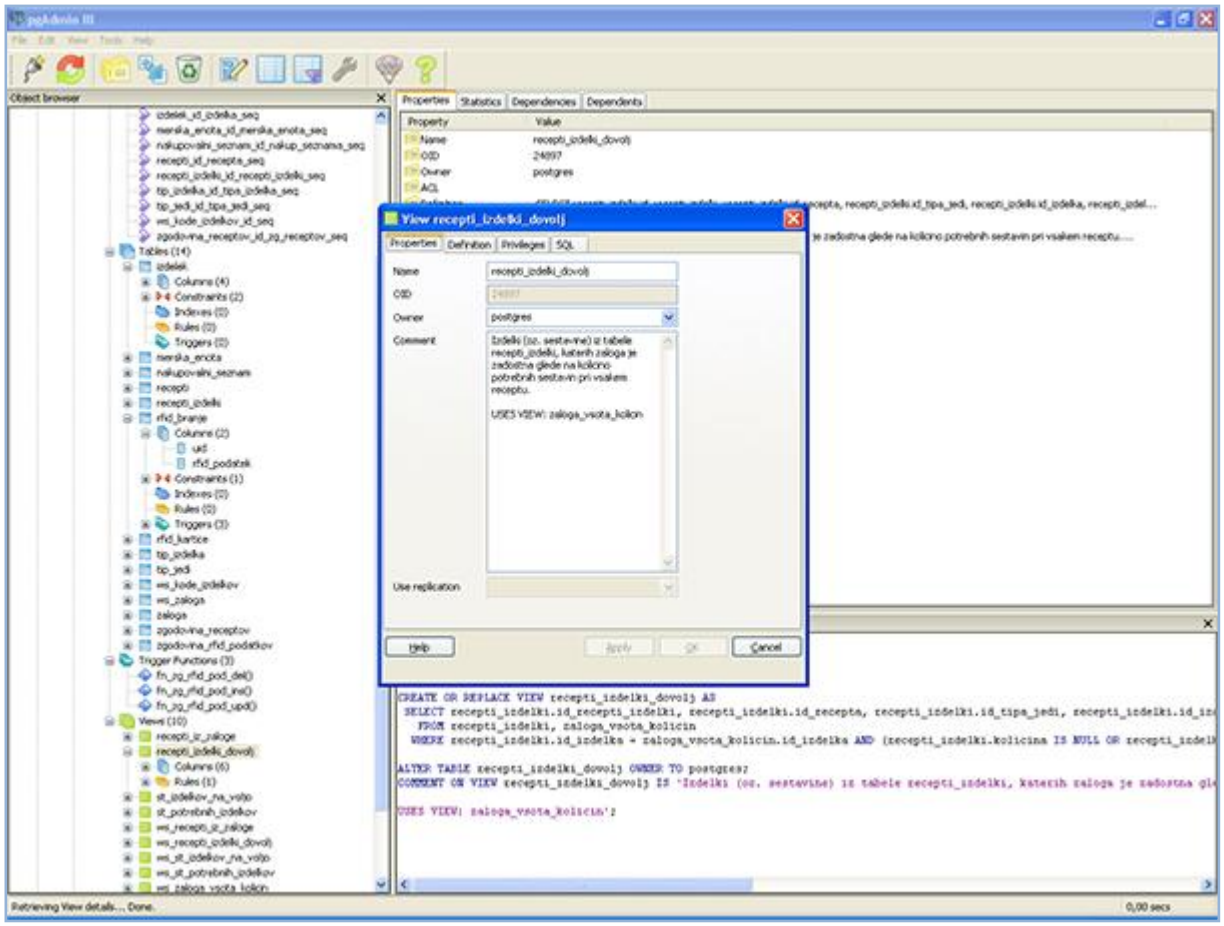
*Tabela 8. Merske enote*

## 4.3. Zasnova podatkovne baze

V namestitvenem paketu odprtokodne baze PostgreSQL verzije 8.3 so na voljo orodja pgAdmin III za upravljanje in administracijo baze. Uporabil sem:

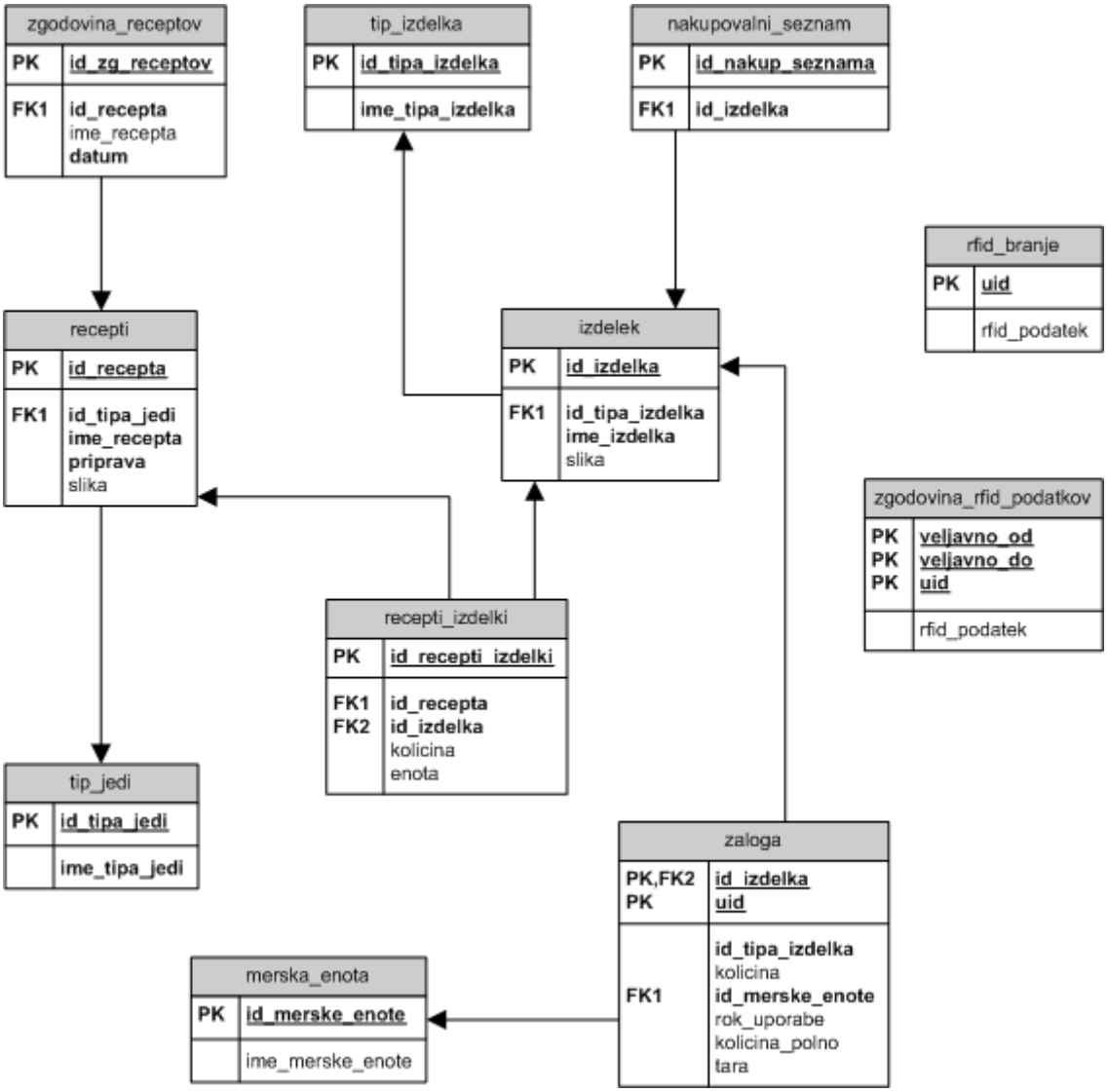
- grafični vmesnik za administracijo baze,
- urejevalnik programske kode SQL z vgrajenim barvanjem sintakse in
- orodje za poizvedbe SQL.

Grafični vmesnik, ki ga vidimo na sliki 14, zadostuje za osnovno administracijo baze in nam olajša delo, saj ni potrebno za vsako poizvedbo pisati stavkov SQL, temveč lahko uporabimo že vgrajene ukaze in vnosne obrazce.




*Slika 14. Grafični vmesnik pgAdmin III.*

Na sliki 15 je prikazana shema izdelane podatkovne baze, na kateri so predstavljene pomembnejše tabele, njihovi primarni ključi (PK), tuji ključi (FK), atributi ter odvisnosti med tabelami.




*Slika 15. Shema podatkovne baze s pomembnejšimi tabelami in njihovimi atributi.*

Snovanje podatkovne baze sem začel s tabelo, v katero se zapišejo podatki z značk RFID. Ko se v sistemu inteligentnega hladilnika katera od značk RFID približa čitalniku, le ta s pomočjo antene značko zazna in z nje prebere podatke. Ti podatki se zapišejo v tabelo **rfid_branje** (slika 16), ki vsebuje podatke z značk RFID v dometu čitalnika in predstavljajo našo zalogo živil v hladilniku. V prvi stolpec tabele (uid) se zapiše unikatna številka kartice, ki je primarni ključ tabele. V drugi stolpec (rfid_podatek) se zapiše podatek o izdelku, ki je definiran kot niz šestnajstiških števil tipa zapisa A ali B. Tip zapisa definirata prva dva znaka niza: 01 pomeni zapis A, 02 pa pomeni zapis B. V skladu z obliko zapisa, se podatek dekodira in zapiše v tabelo *zaloga*.

V tabeli **zaloga** (slika 17) je niz *rfid_podatek* pretvorjen v decimalni zapis in dekodiran v več podatkov različnih podatkovnih tipov (stolpci id_tipa_izdelka, id_izdelka, kolicina, id_merske_enote, rok_uporabe, kolicina_polno in tara).

| | uid [PK] character varying(128) | rfid_podatek character varying(2000) |
|---|---|---|
| 1 | E0040100158C0BD0 | 010001000401221 9A9000A000A010000 |
| 2 | E0040100158C0C12 | 01000100050104CDA9017201F4010064 |
| 3 | E0040100158C168B | 010001000701CAB5BA0028003201000F |
| 4 | E0040100158C18D8 | 01000100090132946902260 3E8010000 |
| 5 | E0040100158C193A | 010001000A0010A9AC00640064010000 |

*Slika 16. Tabela rfid_branje vsebuje podatke z značk, ki jih prebere čitalnik RFID.*

| | uid [PK] character var | id_tipa_izd integer | id_izdelka integer | kolicina integer | id_mers integer | rok_uporabe date | kolicina_polno integer | tara integer |
|---|---|---|---|---|---|---|---|---|
| 1 | E0040100158C0BD0 | 1 | 4 | 10 | 1 | 2009-01-19 | 10 | 0 |
| 2 | E0040100158C0C12 | 1 | 5 | 270 | 1 | 2009-09-17 | 500 | 100 |
| 3 | E0040100158C168B | 1 | 7 | 25 | 1 | 2010-06-30 | 50 | 15 |
| 4 | E0040100158C18D8 | 1 | 9 | 550 | 1 | 2009-09-20 | 1000 | 0 |
| 5 | E0040100158C193A | 1 | 10 | 100 | 1 | 2012-09-01 | 100 | 0 |

*Slika 17. Tabela zaloga vsebuje podatke o izdelkih na zalogi.*

Primer: podatek stolpca *rfid_podatek* v prvi vrsti tabele *rfid_branje* na sliki 16 se dekodira v podatke v prvi vrstici tabele *zaloga* na sliki 17:

- prva dva znaka niza (01) pomenita, da je podatek zapisan kot zapis A,
- naslednji štirje znaki od 3 - 6 (0001) predstavljajo id tipa izdelka = 1 (osnovna živila)
- štirje znaki od 7 - 10 (0004) predstavljajo id izdelka = 4 (kvas)
- osem znakov od 11 - 18 (012219A9) predstavlja rok uporabe 19.01.2009 (rok_uporabe = 2009-01-19)
- štirje znaki od 19 – 22 (000A) predstavljajo trenutno (stehtano) količino izdelka = 10 (10)
- štirje znaki od 23 – 26 (000A) predstavljajo deklarirano neto količino izdelka = 10 (10)
- dva znaka od 27 – 28 (01) predstavljata id merske enote = 1 (g)
- zadnji štirje znaki od 29 – 32 (0000) predstavljajo težo embalaže izdelka = 0

V tabeli **recepti** so zapisani podatki o receptih (slika 18). V tabeli so definirani atributi z imeni *id_recepta*, *id_tipa_jedi*, *ime_recepta*, *priprava* in *slika*.

Vse sestavine posameznih receptov so zapisane v vmesni tabeli **recepti_izdelki**. Vsebuje podatke o sestavinah in količinah sestavin v posameznem receptu, z drugimi besedami podatke o tem, v kateremu receptu je posamezna sestavina v tej količini uporabljena (slika 19).

| | id_recepta [PK] serial | id_tipa_jedi integer | ime_recepta character varying(1 | priprava character varying(1 | slika character varying(12 |
|---|---|---|---|---|---|
| 1 | 1 | 3 | Bučke z mletim mesom | V posodi zmešamo mlet | bucke-z-mletim-mesom.jp |
| 2 | 2 | 3 | Riba s šampinjoni | Šampinjone operemo in | riba-s-sampinjoni.jpg |
| 3 | 3 | 3 | Riba z zelenjavo | Krompir olupimo in nare | riba-z-zelenjavo.jpg |
| 4 | 4 | 3 | Ribji file v foliji | V posodi zmešamo mari | ribji-file-v-foliji.jpg |
| 5 | 5 | 3 | Popečene palačinke z z | Priprava testa | popecene-palacinke-z-zel |

*Slika 18. Tabela recepti.*

| | id_recepti_izdelki [PK] serial | id_recepta integer | id_izdelka integer | kolicina integer | enota character varying(10) |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 12 | 200 | g |
| 2 | 2 | 1 | 19 | 50 | g |
| 3 | 3 | 1 | 1 | 1 | kos |
| 4 | 4 | 1 | 10 | | |
| 5 | 5 | 1 | 23 | 1000 | g |

*Slika 19. Vmesna tabela recepti_izdelki s sestavinami, njihovimi količinami in podatkom o tem, kateremu receptu pripadajo.*

V tabeli **izdelek** (id_tipa_izdelka, id_izdelka, ime_izdelka, slika) so podatki o izdelkih. Za vsak izdelek je določeno, katerega tipa je, kakšen je njegov id, kako se imenuje in kakšno je ime njegove slike. Na sliki 20 so vidni podatki prvih petih izdelkov šifranta izdelkov.

| | id_tipa_izdel integer | id_izdelka [PK] serial | ime_izdelka character varying(50) | slika character varying(128) |
|---|---|---|---|---|
| 1 | 1 | 1 | Jajca | jajca.jpg |
| 2 | 1 | 2 | Jušna kocka | jusna-kocka.jpg |
| 3 | 1 | 3 | Kis | kis.jpg |
| 4 | 1 | 4 | Kvas | kvas.jpg |
| 5 | 1 | 5 | Maslo | maslo.jpg |

*Slika 20. Tabela izdelek predstavlja šifrant izdelkov.*

Tabele **tip_izdelka** (ime_tipa_izdelka, id_tipa_izdelka), **tip_jedi** (id_tipa_jedi, ime_tipa_jedi) in **merska_enota** (id_merske_enote, ime_merske_enote) so šifranti, v katerih se že iz imen tabel razbere njihova vsebina. V tabeli *tip_izdelka* so definirani tipi izdelkov, podani v tabeli 5: osnovna živila, meso in mesni izdelki, mleko in mlečni izdelki, sadje in zelenjava ter žitarice in mlevski izdelki. V tabeli *izdelki* so definirani vsi izdelki (tabela 6), v tabeli *tip_jedi* (tabela 7) pa tipi jedi receptov, in sicer: predjed, juha in glavna jed. V tabeli *merska_enota* (tabela 8) so definirane merske enote: g, ml in kos.

Tabela **nakupovalni_seznam** (id_nakup_seznama, id_izdelka) vsebuje izdelke, ki smo jih dodali v nakupovalni seznam, v tabeli **zgodovina_receptov** (id_zg_receptov, id_recepta, ime_recepta, datum) pa je zapisano, katere recepte smo izbrali pri kuhanju. Iz te zgodovine receptov bi bilo možno izvajati analizo izbire receptov in s tem uporabnikovih navad.

V bazi sem kreiral tudi tabelo **zgodovina_rfid_podatkov**, v katero se ob vsaki spremembi zaloge zapisuje, kateri izdelki so bili od kdaj do kdaj na zalogi in kakšna je bila njihova količina. Ta tabela je namenjena spremljanju uporabnikove zaloge. Za vsak trenutek imamo na voljo vsebino zaloge in tako lahko analiziramo, kateri izdelki so bili porabljeni in v kakšnih količinah. Tabela *zgodovina_rfid_podatkov* se spreminja ob vsaki spremembi tabele *rfid_branje* – ob novem zapisu izdelka v tabeli, ob posodobitvi teže izdelka iz zaloge ali ob porabi izdelka. Za vsakega od teh dogodkov se sproži prožilec (ang. trigger), ki spreminja in posodablja podatke tabele *zgodovina_rfid_podatkov*. Na sliki 21 je zapisana koda prožilca, ki se proži ob posodobljenem podatku izdelka iz zaloge. Ko se izdelku iz zaloge spremeni njegova količina, se v obstoječi zapis v tabeli nastavi vrednost parametra `veljavno_do` na čas, ko se je sprememba zgodila. Kreira se nov zapis z novo - spremenjeno - količino izdelka, parameter `veljavno_od` pa dobi vrednost parametra `veljavno_do` starega zapisa. Vrednost parametra `veljavno_do` v novem zapisu se nastavi na 31.12.9999. Ko se bo količina izdelka zopet spremenila, se bo veljavnost datuma in ure količine izdelka spremenila z 31.12.9999 na novo vrednost v trenutku spremembe.

```
DECLARE
BEGIN
  UPDATE zgodovina_rfid_podatkov
  SET "veljavno_do" = localtimestamp
  WHERE ("uid" = OLD."uid") AND
        ("veljavno_do" = timestamp '9999-12-31 23:59:59');
  INSERT INTO zgodovina_rfid_podatkov("veljavno_od",
         "veljavno_do", "uid", "rfid_podatek")
  VALUES (localtimestamp, timestamp '9999-12-31 23:59:59',
         NEW."uid", NEW."rfid_podatek");
  RETURN NEW;
END;
```

*Slika 21. Prožilec, ki nastavlja veljavnost trajanja določene količine izdelka in se proži ob posodabljanju količine izdelka v tabeli rfid_branje.*

V bazi so kreirani trije pogledi (ang. view) za enostavnejše in bolj pregledno pisanje poizvedb, ki so uporabljene v programski kodi aplikacije. Pogled **zaloga_vsota_kolicin** (id_izdelka, "Vsota kolicin", id_merske_enote) na sliki 22 sešteje količine enakih izdelkov (imajo enak id_izdelka), pri čemer ne upošteva količin tistih izdelkov, ki jim je rok uporabe že potekel.

| | id_izdelka integer | Vsota kolicin bigint | id_merske_enote integer |
|---|---|---|---|
| 1 | 1 | 12 | 3 |
| 2 | 6 | 1000 | 2 |
| 3 | 7 | 40 | 1 |
| 4 | 8 | 71 | 2 |
| 5 | 10 | 50 | 1 |

*Slika 22. Pogled zaloga_vsota_kolicin.*

Pogled **recepti_izdelki_zaloga** je razširjena vmesna tabela *recepti_izdelki*, sestavljena iz tabel *recepti_izdelki*, *merska_enota* in pogleda *zaloga_vsota_kolicin*. Za vsak izdelek oz. sestavino receptov se primerjata potrebna količina za recept in količina izdelka, ki ga imamo na zalogi. V atribut *na_zalogi* se zapiše 0, če imamo dovolj izdelka in -1, če imamo izdelka premalo ali ga nimamo na zalogi. Na sliki 23 je prikazana poizvedba SQL, ki definira pogled *recepti_izdelki_zaloga*, slika 24 pa prikazuje pet vrstic tabele pogleda *recepti_izdelki_zaloga*.

```
SELECT
   s_recepti.id_recepti_izdelki, s_recepti.id_recepta,
   izdelek.id_tipa_izdelka, s_recepti.id_izdelka, izdelek.ime_izdelka,
   s_recepti.kolicina, s_recepti.enota, s_zaloga."Vsota kolicin",
   s_zaloga.ime_merske_enote,
   CASE WHEN s_recepti.kolicina IS NULL AND
             s_zaloga."Vsota kolicin" IS NULL THEN (-1)
        WHEN s_zaloga."Vsota kolicin" IS NULL THEN (-1)
        WHEN s_recepti.kolicina IS NULL THEN 0
        WHEN (s_zaloga."Vsota kolicin" - s_recepti.kolicina) >= 0 THEN 0
        ELSE (-1)
   END AS na_zalogi
FROM (SELECT
         recepti_izdelki.id_recepti_izdelki,
         recepti_izdelki.id_recepta, recepti_izdelki.id_izdelka,
         recepti_izdelki.kolicina, recepti_izdelki.enota
      FROM recepti_izdelki) s_recepti LEFT JOIN
     (SELECT
         zaloga_vsota_kolicin.id_izdelka, zaloga_vsota_kolicin.
         "Vsota kolicin", merska_enota.ime_merske_enote
      FROM zaloga_vsota_kolicin, merska_enota
      WHERE merska_enota.id_merske_enote =
            zaloga_vsota_kolicin.id_merske_enote) s_zaloga
ON s_recepti.id_izdelka = s_zaloga.id_izdelka, izdelek
WHERE izdelek.id_izdelka = s_recepti.id_izdelka
ORDER BY s_recepti.id_recepti_izdelki;
```

*Slika 23. Poizvedba pogleda recepti_izdelki_zaloga.*

| | id_re integ | id_recepta integer | id_tipa_iz integer | id_izdelka integer | ime_izdelka character va | kolicina integer | enota chara | Vsota kolicin bigint | ime_ chara | na_zalogi integer |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2 | 12 | Mleto meso | 200 | g | 500 | g | 0 |
| 2 | 2 | 1 | 3 | 19 | Sir | 50 | g | | | -1 |
| 3 | 3 | 1 | 1 | 1 | Jajca | 1 | kos | 12 | kos | 0 |
| 4 | 4 | 1 | 1 | 10 | Sol | | | 50 | g | 0 |
| 5 | 5 | 1 | 4 | 23 | Bučke | 1000 | g | 1500 | g | 0 |

*Slika 24. Pogled recepti_izdelki_zaloga.*

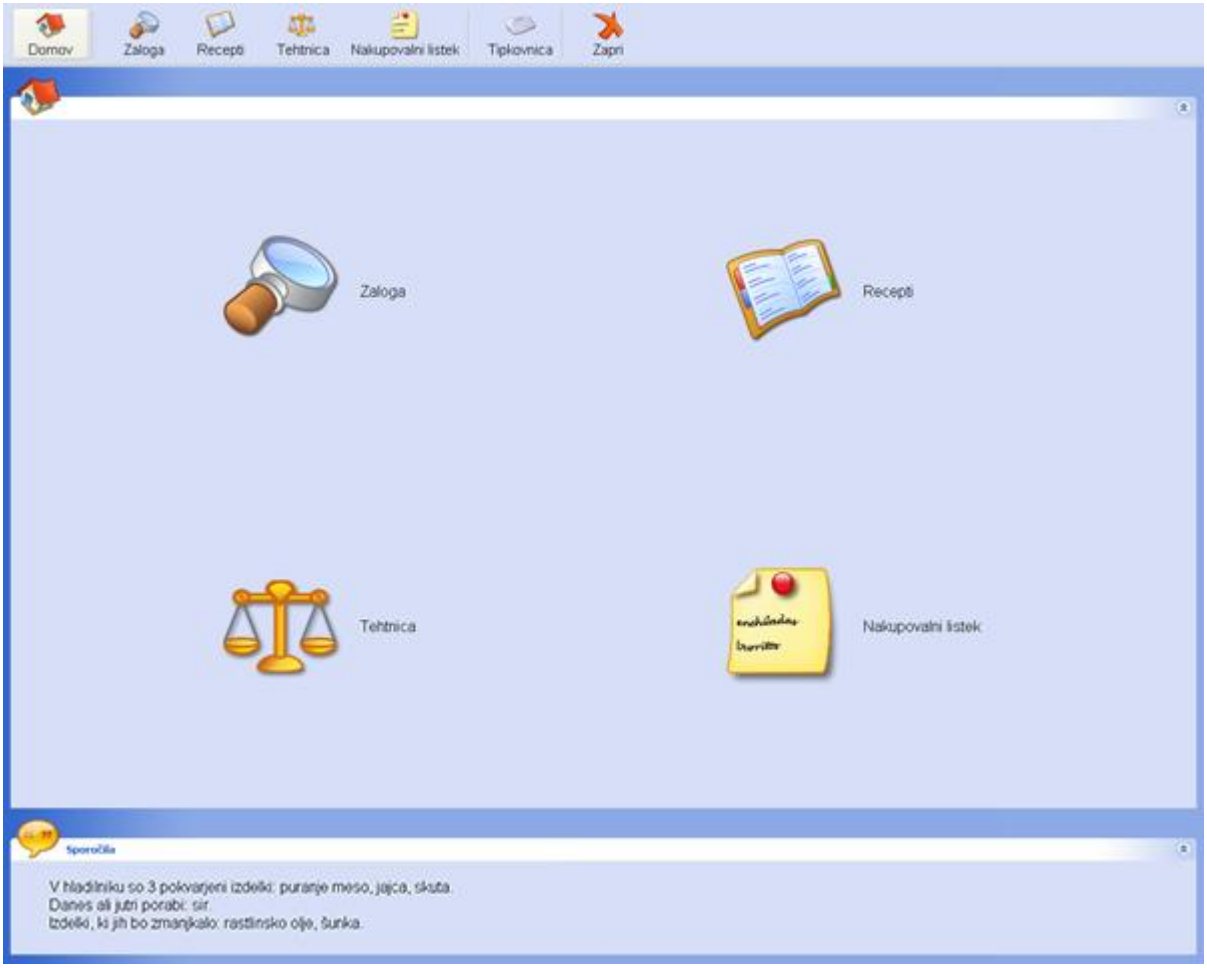
Pogled **recepti_iz_zaloge** je sestavljen iz pogleda *recepti_izdelki_zaloga* in tabel *recepti* in *tip_jedi*. Ima atribut *st_manjkajocih_izdelkov*, kamor se za vsak recept zapiše vsota vrednosti atributa *na_zalogi* (iz pogleda recepti_izdelki_zaloga). Pogled *recepti_iz_zaloge* je glavni vir podatkov aplikaciji pri funkcionalnosti "predlogi kuharskih receptov", ki je opisana v poglavju 4.3.2. Tabela, ki je rezultat poizvedbe pogleda *recepti_iz_zaloge* je prikazana na sliki 25.

| | id_recepta integer | ime_recepta character var | id_tipa_jed integer | ime_tipa_jec character va | st_manjkajocih bigint | priprava character var | slika character v |
|---|---|---|---|---|---|---|---|
| 1 | 1 | Bučke z mletim r | 3 | Glavna jed | 1 | V posodi zmešar | bucke-z-mletim |
| 2 | 2 | Riba s šampinjoı | 3 | Glavna jed | 2 | Šampinjone oper | riba-s-sampinjc |
| 3 | 3 | Riba z zelenjavc | 3 | Glavna jed | 5 | Krompir olupimo i | riba-z-zelenjav |
| 4 | 4 | Ribji file v foliji | 3 | Glavna jed | 2 | V posodi zmešar | ribji-file-v-foliji |
| 5 | 5 | Popečene palač | 3 | Glavna jed | 7 | Priprava testa | popecene-pala |

*Slika 25. Pogled recepti_iz_zaloge.*

## 4.4. Izvedba programske aplikacije

Uporabniški vmesnik aplikacije za uporabo v sistemu inteligentnega hladilnika (slika 26) je sestavljen iz treh delov. Zgornji del sestavlja orodna vrstica z gumbi: domov, zaloga, recepti, tehtnica, nakupovalni listek, tipkovnica in zapri. Osrednji – glavni – del programa je vsebinsko razdeljen na štiri pomembnejše sklope oz. funkcionalnosti: pregledovanje zaloge, predlogi kuharskih receptov, tehtanje izdelkov ter urejanje in prikaz nakupovalnega seznama. Spodnji del je namenjen sporočilom, s katerimi nas sistem opozarja na izdelke, ki jim je potekel rok uporabe, na izdelke, ki bi jih bilo treba porabiti, ker jim bo rok uporabe potekel čez dva dneva, in izdelke, ki jih bo zmanjkalo.



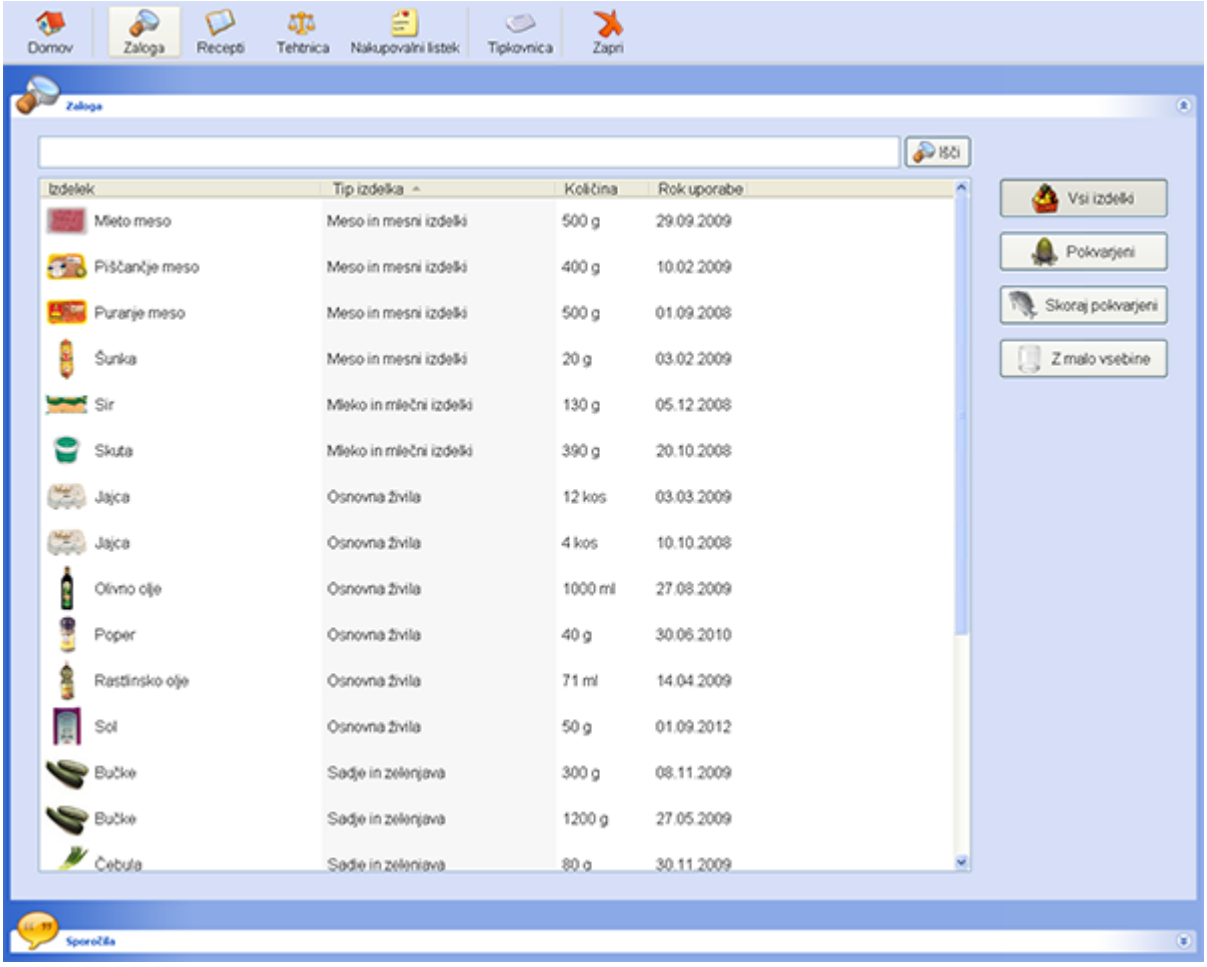

*Slika 26. Posnetek zaslona aplikacije ob zagonu.*

Aplikacija za predstavitev funkcionalnosti vključuje uporabo podatkovnih baz in spletno storitev za napovedovanje jedilnika in nakupovalnega seznama.

### 4.4.1. Pregledovanje zaloge

Slika 27 prikazuje funkcionalnost pregledovanja zaloge prehrambnih izdelkov v inteligentnem hladilniku. Podatki se zajamejo iz ustvarjenih podatkovnih tabel, opisanih v poglavju 4.3. (*izdelek*, *tip_izdelka*, *zaloga* in *merska_enota*). Seznam zalog bo prazen, če nimamo nobenega izdelka v območju čitalnika RFID ali če sistem ne deluje. Na izpisu zaloge je prikazana slika izdelka, njegovo ime, količina izdelka na zalogi in njegov rok uporabe. Na desni strani so gumbi, ki omogočajo pregled zaloge po naslednjih kriterijih:

- "Vsi izdelki" – osnovni prikaz, ki prikaže vse izdelke zaloge;
- "Pokvarjeni" – prikaže le izdelke, ki jim je potekel rok uporabe;
- "Skoraj pokvarjeni" – prikaže izdelke, ki jim bo čez dva dneva potekel rok uporabe (imamo torej dva dneva časa, da jih porabimo);
- "Z malo vsebine" – prikaže izdelke, katerih vsebina je manjša od 10 % neto količine polnega izdelka.




*Slika 27. Izpis zaloge.*

Izdelke lahko sortiramo po vseh stolpcih, za katere je to smiselno, tj. po izdelkih, tipu izdelka in roku uporabe.

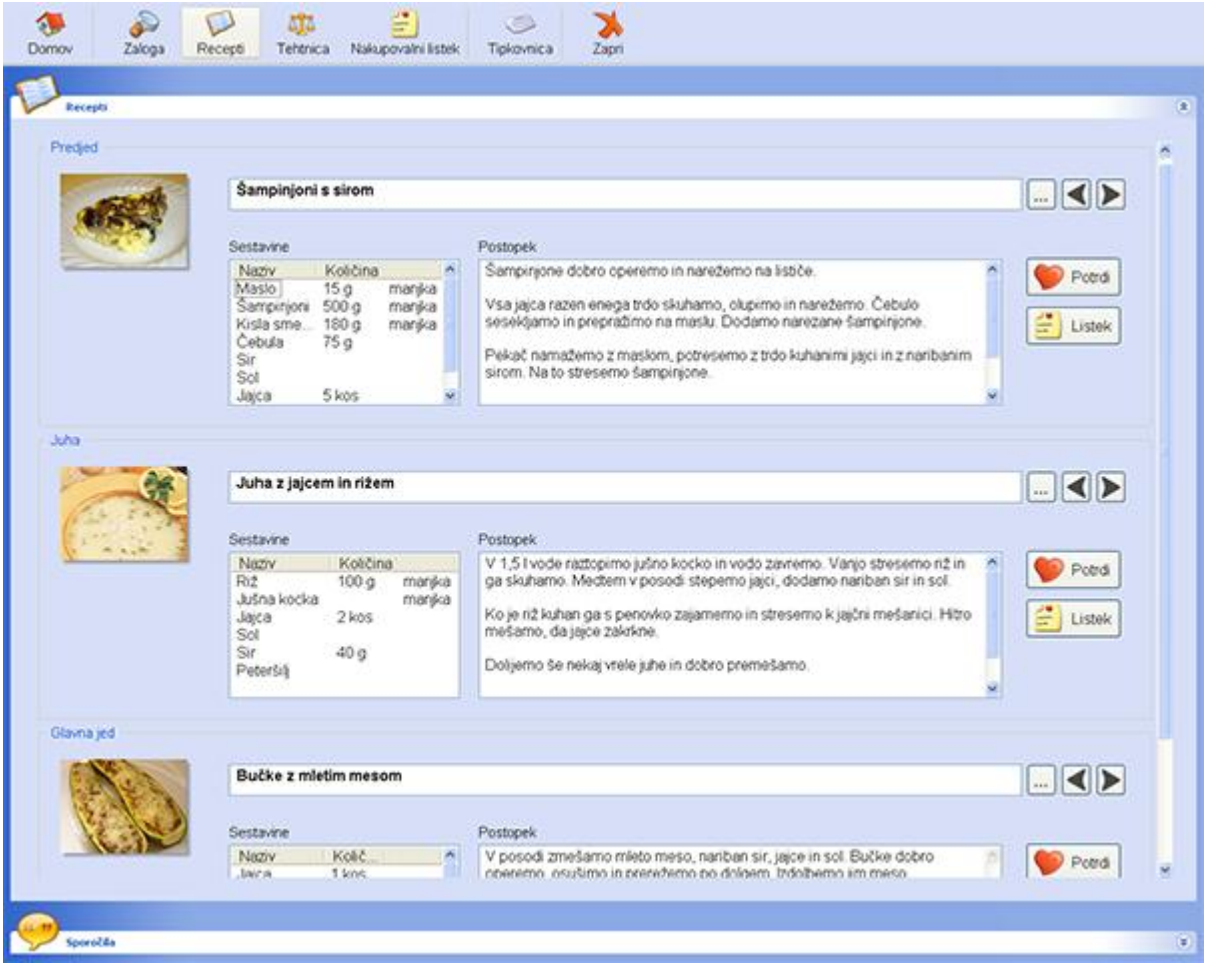
### 4.4.2. Predlogi kuharskih receptov

Aplikacija ponuja dva načina pregledovanja kuharskih receptov:

- napovedovanje receptov na podlagi zaloge
- brskanje po receptih

Pri prvem načinu nam na podlagi zaloge izdelkov in z upoštevanjem njihovih količin aplikacija predlaga recepte, ki jih lahko naredimo (slika 28). Prikažejo se možni recepti za vse tipe jedi, ki so definirane: v našem primeru za predjed, juho in glavno jed.

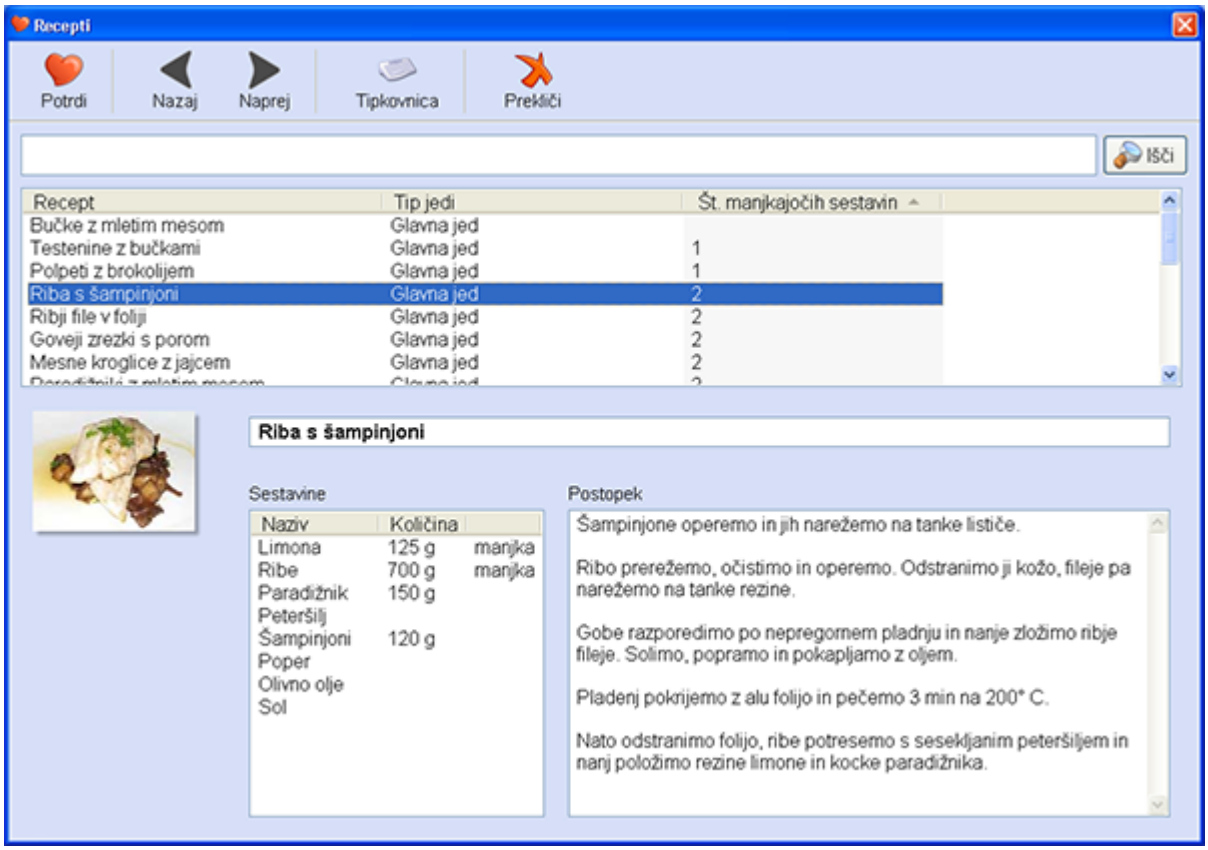
Če za določen tip jedi ne moremo narediti nobenega recepta, ker za nobenega nimamo vseh sestavin ali pa jih imamo premalo, nam aplikacija prikaže vse recepte tega tipa jedi, ki jim manjka najmanj sestavin. Tako nam aplikacija za vsak tip jedi vedno ponudi vsaj en recept, kar nam olajšuje njihov izbor. V primeru, da za recept nimamo vseh sestavin, se na seznamu sestavin za izdelavo recepta v desnem stolpcu izpiše tekst "manjka". Ob pritisku na gumb "Listek" se vsi manjkajoči izdelki recepta dodajo na nakupovalni listek.




*Slika 28. Predlogi kuharskih receptov glede na našo zalogo.*

Z gumbom "Potrdi" potrdimo, da bomo recept pripravili. Recept se zapiše v bazo v tabelo *zgodovina_receptov*, ki zapisuje zgodovino vseh potrjenih oz. izbranih receptov.

Drugi način pregledovanja receptov je brskanje po receptih. Če pritisnemo gumb "…" desno od imena recepta, se odprejo vsi recepti za tip jedi, ki smo ga izbrali. Slika 29 prikazuje seznam receptov za glavne jedi. Ob vsakem receptu je tudi število manjkajočih sestavin (če katera manjka), recepte pa lahko sortiramo po imenu in po številu manjkajočih sestavin. Če recept potrdimo, se okno za brskanje po vseh receptih tipa jedi zapre, potrjen recept pa je dodan in prikazan na seznamu receptov, ki nam jih je aplikacija predlagala.




*Slika 29. Okno, ki prikazuje brskanje po vseh receptih glavnih jedi.*

Podatki o številu manjkajočih sestavin pri posameznih receptih in ostali podatki, potrebni za izpis predlogov receptov, se v bazi zajamejo iz pogleda *recepti_iz_zaloge*. Iskanje receptov se v aplikaciji lahko izvaja lokalno – s poizvedbami po lastni bazi ali pa preko spletne storitve. Aplikacija pošlje spletni storitvi kodirane podatke o izdelkih iz zaloge, spletna storitev podatke dekodira (pri tem uporabi enako kodno tabelo kot aplikacija), poišče recepte v lastni bazi receptov in aplikaciji vrne predloge receptov. Verjetno se bi v prihodnosti na podoben način pošiljale kode EPC izdelkov (glej poglavje 3.1.). Te bi se dekodirale s poizvedbo v centralni sistem kod EPC, kakršen je na primer že omenjeno omrežje EPCglobal. V tem primeru lastne baze za kodiranje podatkov niti ne bi potrebovali.

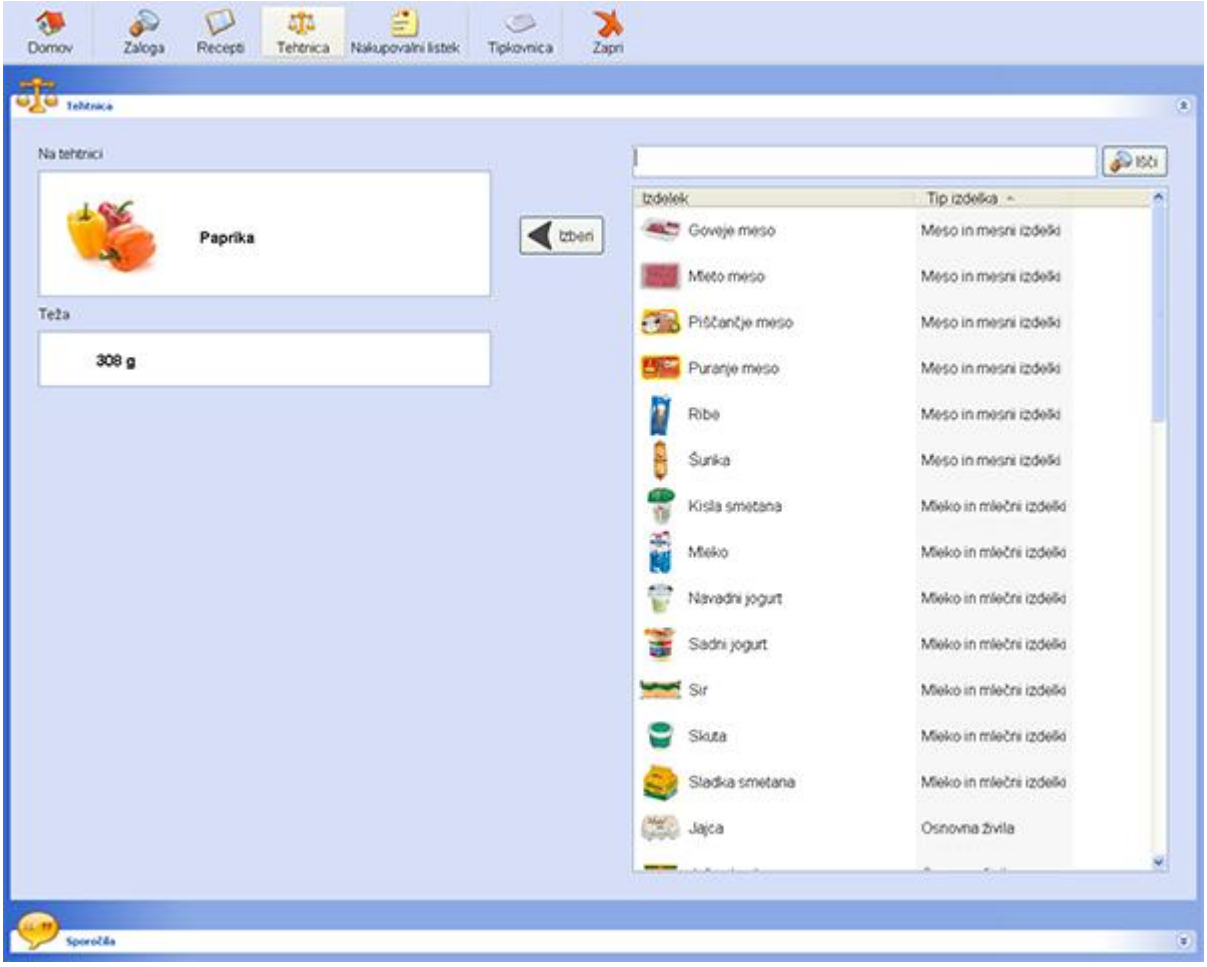
### 4.4.3. Tehtanje izdelkov

Tehtanje izdelkov je v sistemu realizirano tako, da se teža izdelka, ki jo tehtnica izmeri, zapiše na značko, ko izdelek približamo malemu čitalniku RFID. Ker sta v sistemu tehtnica in mali čitalnik dve fizično ločeni napravi, je čitalnik stal na tehtnici, ta pa je bila nastavljena tako, da teže čitalnika ni upoštevala. Izdelek smo tako položili ob čitalnik na tehtnico, s čimer se je izdelek stehtal, novo težo pa je čitalnik zapisal na značko izdelka.

Funkcionalnost "Tehtnica" omogoča dve funkciji:

- tehtanje izdelkov
- zapis novega izdelka, ki je v bazi zalog

Če želimo spremljati tehtanje izdelkov, v aplikaciji izberemo funkcionalnost "Tehtnica". V poljih se nam prikaže slika, ime in teža izdelka, ki smo ga stehtali (slika 30). Če želimo na značko zapisati drug izdelek, pa pri tehtanju na seznamu izdelkov izberemo izdelek in pritisnemo gumb "Izberi". Seznam izdelkov se ustvari s poizvedbo v bazi iz tabel *izdelek* in *tip_izdelka*.

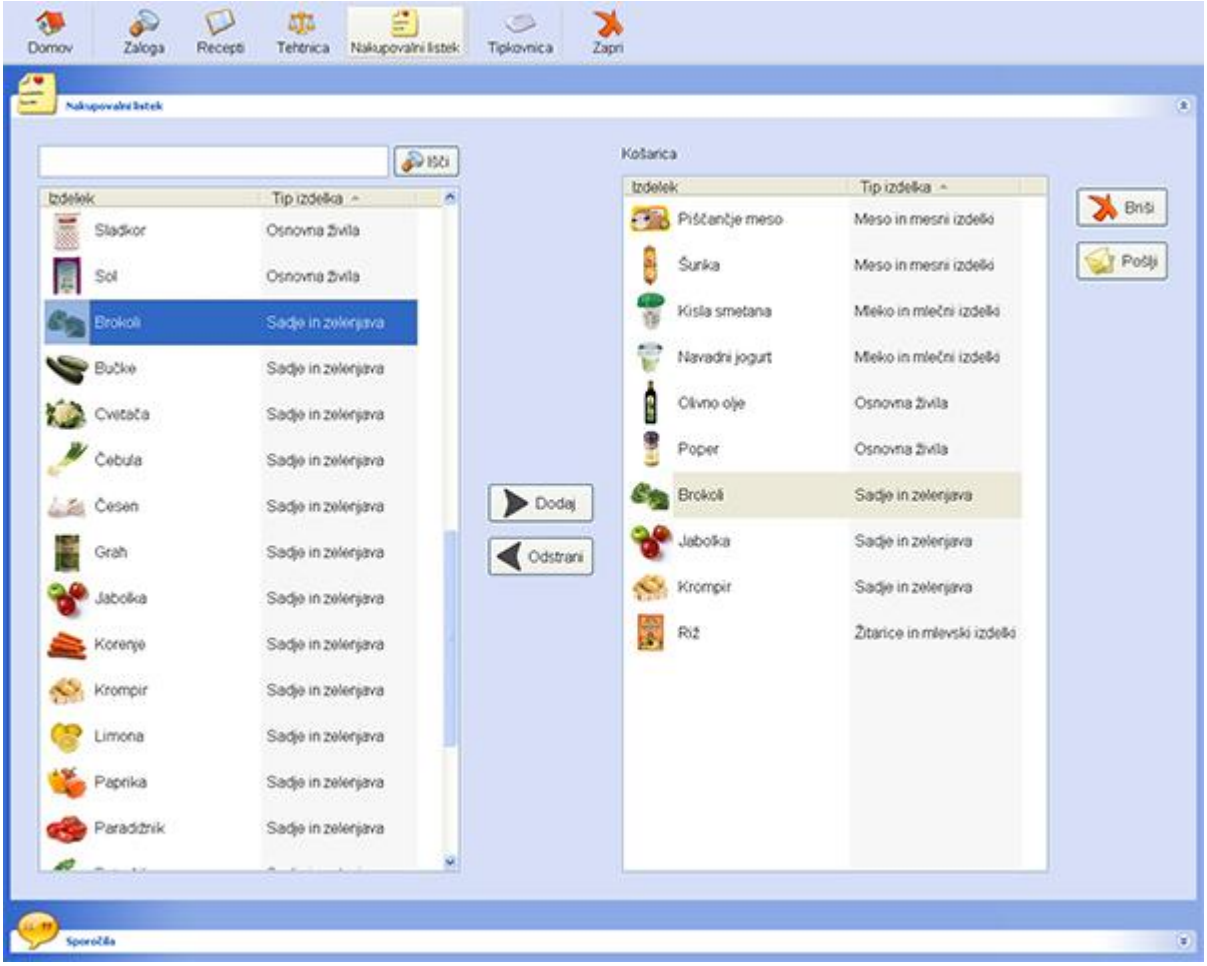



*Slika 30. Prikaz tehtanja v aplikaciji.*

### 4.4.4. Urejanje nakupovalnega seznama (listka)

Aplikacija omogoča kreiranje in urejanje nakupovalnega seznama (slika 31). Izdelke lahko dodajamo na seznam na dva načina:

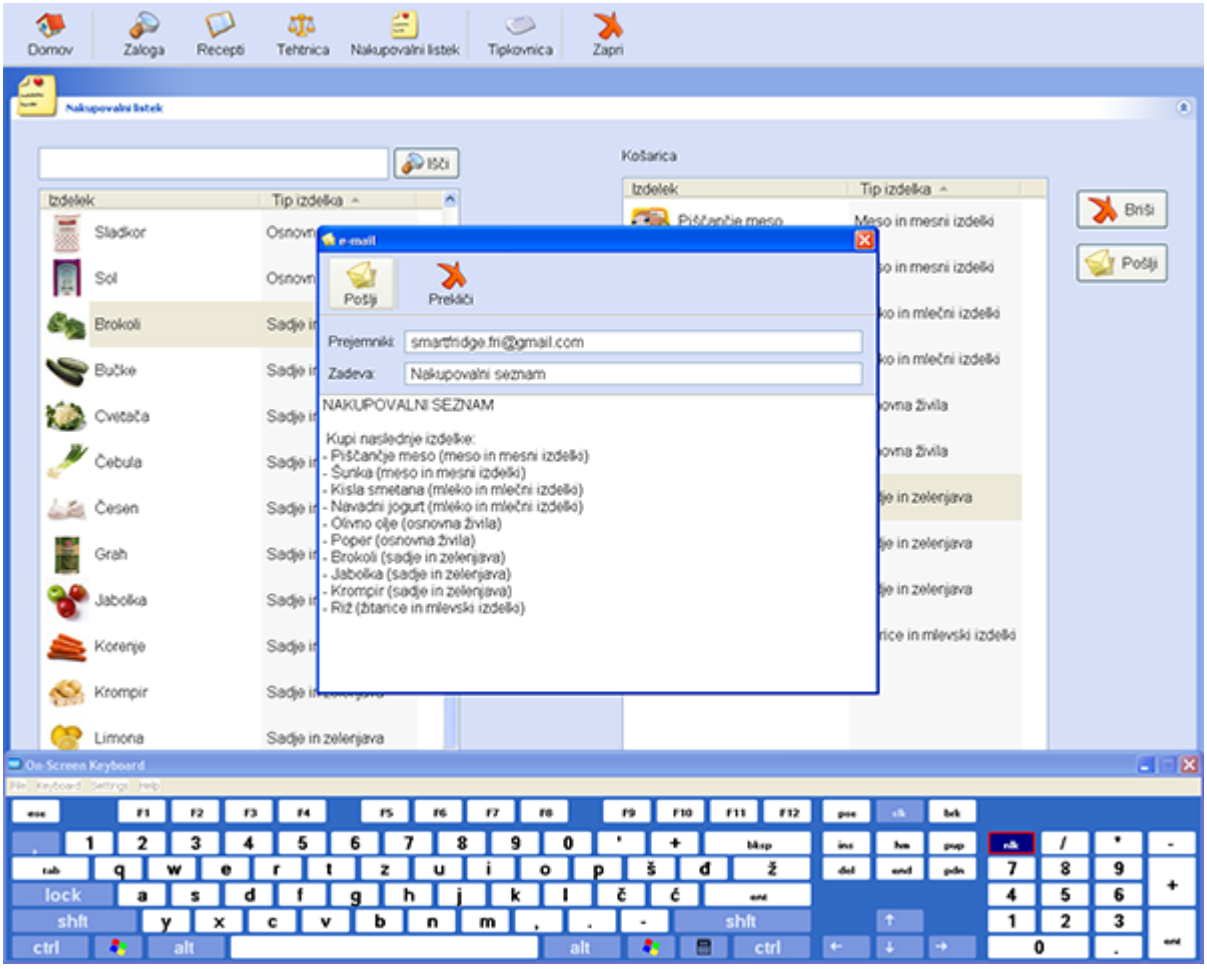
- med pregledovanjem seznama vseh izdelkov lahko na nakupovalni seznam dodamo vsak izdelek posebej;
- med pregledovanjem receptov pritisnemo na gumb "Listek" in s tem na nakupovalni seznam dodamo vse manjkajoče izdelke (sestavine) recepta, ki nam ga je predlagal sistem; če brskamo po vseh receptih tipa jedi, moramo recept najprej potrditi, nato pa pritisnemo gumb "Listek".




*Slika 31. Urejanje nakupovalnega seznama.*

S klikom na gumb "Briši" se z nakupovalnega seznama odstranijo vsi izdelki in seznam postane prazen.

S pritiskom na gumb "Pošlji" se odpre okno z vnosno formo za urejanje in pošiljanje elektronske pošte. Forma je že predizpolnjena, v sporočilu so našteti izdelki in v oklepajih njihovi tipi izdelkov. Za izpis nakupovalnega seznama se izvede poizvedba po tabelah *nakupovalni_seznam*, *izdelek* in *tip_izdelka*.




*Slika 32. Okno za urejanje elektronskega sporočila s tipkovnico na spodnjem delu zaslona.*

Uporabnik lahko vsa polja elektronske pošte ureja. S pritiskom na gumb "Tipkovnica" v orodni vrstici aplikacije se na spodnjem delu ekrana prikaže tipkovnica. Z uporabo zaslona na dotik lahko to tipkovnico preprosto uporabljamo s tipkanjem po ekranu (slika 32).

## 4.5. Spletna storitev za iskanje receptov

V diplomski nalogi je bila v sistemu inteligentnega hladilnika predvidena tudi uporaba spletnih storitev. Definiral sem spletno storitev *wsPoisciRecepte*, ki sprejme tip jedi in seznam izdelkov ter vrne seznam receptov za podani tip jedi, ki jih lahko iz teh izdelkov naredimo. V nastavitveni datoteki aplikacije obstaja parameter *webService*, ki se ga lahko nastavi na "true" ali "false". Če je nastavljen na true, bo aplikacija za predloge receptov uporabila spletno storitev, v nasprotnem primeru pa jih bo poiskala s poizvedbo v lokalni bazi.

Za razvoj in delovanje spletne storitve sem namestil Apache Axis2 in dodatke za Eclipse:

- **Axis2** [25]
  Axis2 je ogrodje za razvoj spletnih storitev. Je na novo začrtan in napisan model starejše arhitekture Axis, ki je implementacija protokola SOAP. Je učinkovitejši, bolj modularen in bolj XML-orientiran kot starejša verzija Axis.
- **Axis2 Code Generator** [26]
  Generator kode se uporablja za generiranje datotek WSDL iz javanskih razredov (Java2WSDL) in generiranje javanskih razredov iz datotek WSDL (WSDL2Java).
- **Axis2 Service Archiver** [26]
  Omogoča kreiranje arhivskih datotek spletnih storitev (s končnico ".aar" ali ".jar"), ki jih je mogoče postaviti (ang. deploy) kot spletne storitve na pogon Axis2.
- **XJC** [27]
  Dodatek XJC za Eclipse je orodje z grafičnim vmesnikom in prevajalnikom za preslikovanje xjc. Iz sheme XML generira javanske razrede z metodami, ki jih lahko uporabimo v programskemu vmesniku JAXB (Java Architecture for XML Binding).

### 4.5.1. Izdelava shem XML

Pri načrtovanju spletne storitve sem se odločil, da bo podatke sprejemala in vračala v obliki XML. Shema XML opisuje strukturo dokumenta XML in definira njegove veljavne elemente. Za razvoj spletne storitve sem izdelal dve shemi XML:

- shemo `rfidKodeIzdelkov.xsd`,
- shemo `recepti.xsd`.

Za podatke o izdelkih, ki jih spletna storitev sprejme, sem izdelal shemo `rfidKodeIzdelkov.xsd`, ki definira strukturo seznama izdelkov. Sestavlja ga zaporedje elementov `RFIDKodaIzdelka` tipa `string`. Shema je prikazana na sliki 33. Atributa `minOccurs` in `maxOccurs` določata najmanjše in največje možno število elementov.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
           jaxb:version="1.0">

<xs:element name="seznamRFIDKodIzdelkov">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="RFIDKodaIzdelka" type="xs:string"
                  minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

*Slika 33. Shema rfidkodeIzdelkov.xsd, ki določa strukturo dokumenta poslanih izdelkov.*

Za seznam receptov, ki ga spletna storitev vrača, sem definiral shemo `recepti.xsd` (slika 34). Shema določa, da je element `recepti` korenski element, sestavljen pa je iz elementov `receptTip`, v katerih so podatki o posameznem receptu zapisani v elementih: `imeRecepta`, `tipJedi`, `seznamSestavin`, `priprava` in `slika`. Element `seznamSestavin` je sestavljen iz podatkov o sestavinah tipa `sestavinaTip`, ki ga sestavljajo elementi: `imeSestavine`, `kolicina` in `naZalogi`, kjer sta dovoljeni vrednosti elementa `naZalogi` "manjka" ali "prazen niz ".

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
           jaxb:version="1.0">

<xs:element name="recepti">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="recept" type="receptTip" minOccurs="0"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="receptTip">
  <xs:sequence>
    <xs:element name="imeRecepta" type="xs:string"/>
    <xs:element name="tipJedi" type="xs:string"/>
    <xs:element name="seznamSestavin">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="sestavina" type="sestavinaTip"
                      minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="priprava" type="xs:string"/>
```

```
    <xs:element name="slika" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="sestavinaTip">
  <xs:sequence>
    <xs:element name="imeSestavine" type="xs:string"/>
    <xs:element name="kolicina" type="xs:string"/>
    <xs:element name="naZalogi">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="manjka"/>
          <xs:enumeration value=""/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

</xs:schema>
```

*Slika 34. Shema recepti.xsd, ki določa strukturo dokumenta možnih receptov.*

### 4.5.2. Preslikava shem XML v razrede Java

V naslednjem koraku izdelave spletne storitve sem preslikal shemi XML v javanske razrede. Preslikavo sem naredil z orodjem XJC, ki je pravzaprav prevajalnik za preslikovanje JAXB (ang. JAXB binding compiler) v obliki čarovnika. JAXB je programski vmesnik, ki poenostavi dostop programov Java do dokumenta XML, tako da predstavi dokument XML v obliki javanskih razredov (slika 35) [17]. XJC je generiral naslednje razrede:

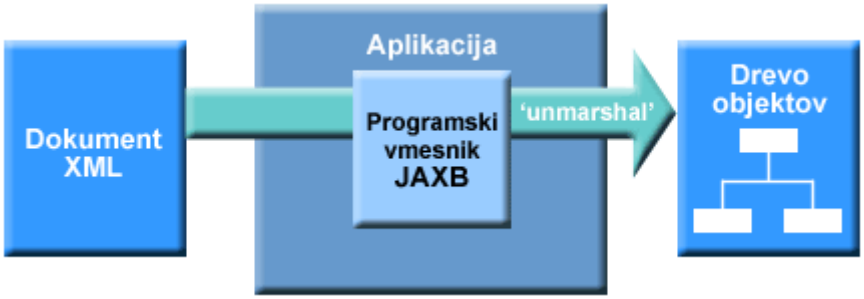
- *ObjectFactory*,
- *Recepti*,
- *ReceptTip*,
- *SestavinaTip*,
- *SeznamRFIDKodIzdelkov*.

Razred *ObjectFactory* je sestavljen iz metod za kreiranje objektov ostalih štirih razredov: *Recepti*, *ReceptTip*, *SestavinaTip* in *SeznamRFIDKodIzdelkov*. V teh štirih razredih so definirane "get" metode za dostop do vrednosti objektnih spremenljivk in "set" metode za nastavljanje teh vrednosti.

*Slika 35. Preslikava sheme XML v razrede.*

### 4.5.3. Deserializacija in serializacija podatkov

Spletna storitev sprejme podatke o izdelkih v obliki XML. Te podatke mora najprej pretvoriti v drevo java objektov (ang. java content tree), da jih lahko obdeluje. Ta proces se imenuje deserializacija (ang. deserialization) in jo omogoča razred `Unmarshaller`, ki je del programskega vmesnika JAXB. Shema deserializacije je prikazana na sliki 36.




*Slika 36. Shema procesa deserializacije podatkov XML.*

Deserializacija vhodnih podatkov v spletni storitvi *wsPoisciRecepte* poteka takole:

```
1 JAXBContext jc = JAXBContext.newInstance("xml");
2 Unmarshaller unmarshaller = jc.createUnmarshaller();
```

V prvi vrstici se kreira objekt `JAXBContext`, ki sprejme ime paketa, v katerem se nahajajo razredi, ki so se prej generirali z XJC. V drugi vrstici se kreira objekt `Unmarshaller`.

```
3 Schema schema = SchemaFactory.newInstance(
    XMLConstants.W3C_XML_SCHEMA_NS_URI).newSchema(
        ClassLoader.getSystemResource("xml/rfidKodeIzdelkov.xsd"));
4 unmarshaller.setSchema(schema);
```

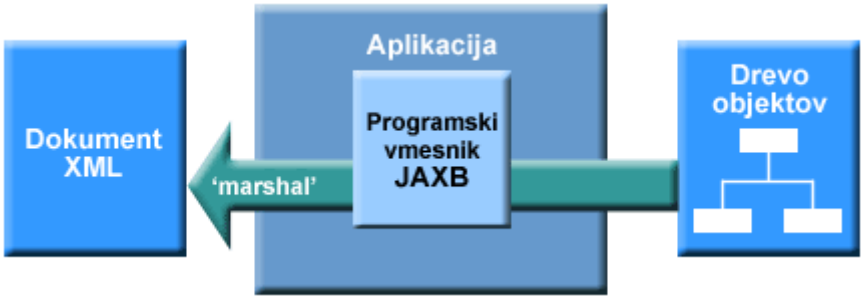
V tretji vrstici se kreira objekt `Schema`, ki kot parameter sprejme ime sheme XML in pot do nje. V naslednjo vrstico se nastavi preverjanje skladnosti podatkov o izdelkih s shemo `rfidKodeIzdelkov.xsd`, določeno eno vrstico višje.

```
5 StringReader rfidKodeStringReader =
    new StringReader(rfidKodeString);
6 SeznamRFIDKodIzdelkov seznamRfidKI = (SeznamRFIDKodIzdelkov)
    unmarshaller.unmarshal(rfidKodeStringReader);
7 List<String> seznamKod = seznamRfidKI.getRFIDKodaIzdelka();
```

Parameter `rfidKodeString` je vhodni parameter deserializacije v katerem so v tipu `string` zapisani podatki o izdelkih v obliki XML. V peti vrstici se pretvori `string` v tip

`StringReader`, da se ga lahko v šesti vrstici poda objektu `Unmarshaller`, ki z metodo `unmarshal` izvede deserializacijo in preverjanje skladnosti s shemo. V zadnji vrstici se z get metodo razreda *SeznamRFIDKodIzdelkov* potem dostopa do podatkov o izdelkih.

Obraten proces od deserializacije je proces pretvarjanja drevesa java objektov v dokument XML in se imenuje serializacija (ang. serialization). Za ta proces je zadolžen razred JAXB z imenom `Marshaller`. Shema serializacije je predstavljena na sliki 37.




*Slika 37. Shema procesa serializacije podatkov XML.*

Izvorna koda serializacije je precej podobna kodi deserializacije, zato je ne bom posebej opisoval.

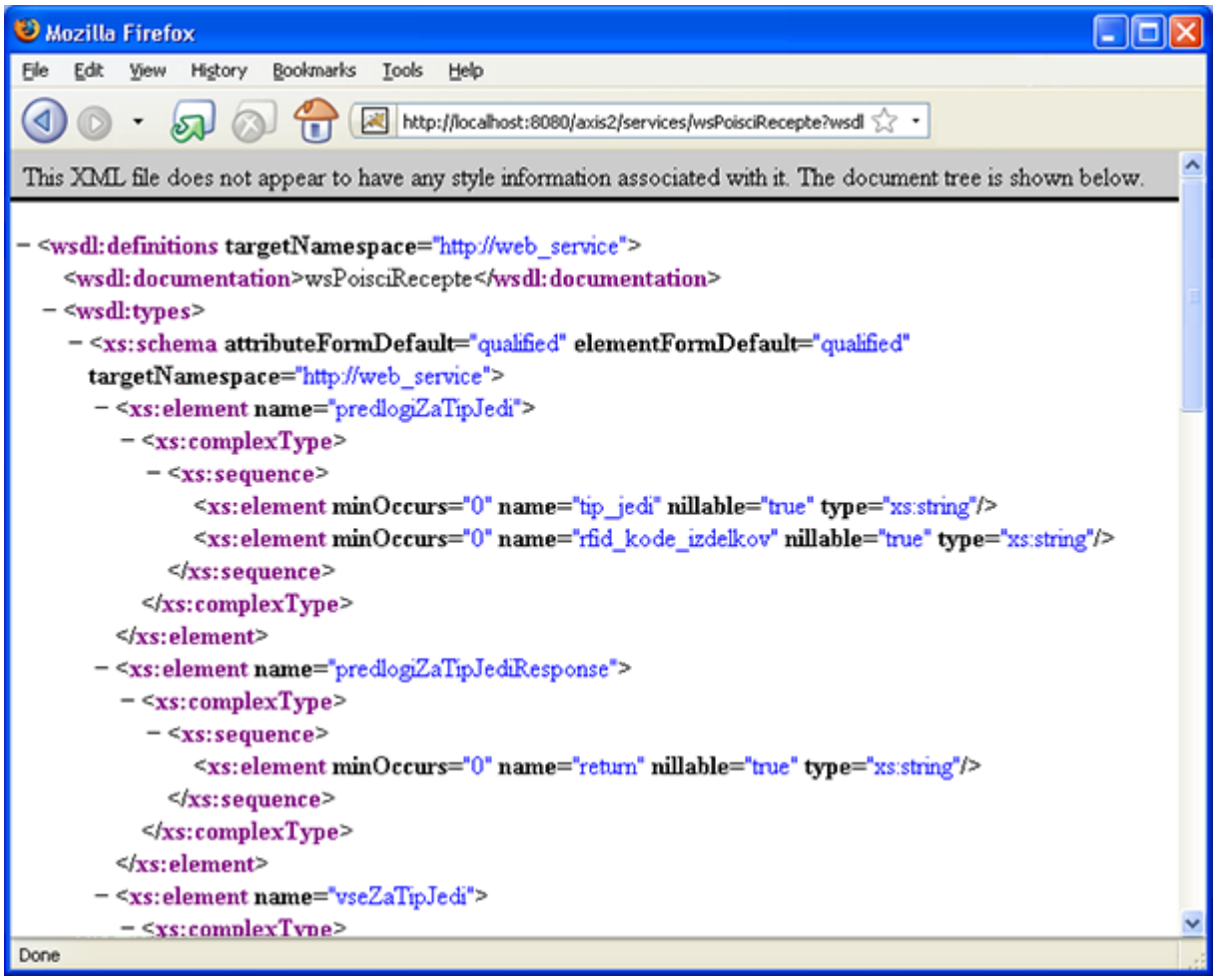
### 4.5.4. Kreiranje dokumenta WSDL

Vmesnik spletne storitve določa dokument WSDL, v katerem so opisane vse lastnosti spletne storitve. Za kreiranje dokumenta WSDL sem uporabil orodje Axis Code Generator, ki preslika razred ali vmesnik v Javi v dokument WSDL. Razred `WsPoisciRecepte`, ki sem ga preslikal v WSDL, vsebuje metodi *predlogiZaTipJedi*, ki vrne recepte za podani tip jedi, ki jih lahko naredimo iz zaloge, in *vseZaTipJedi*, ki vrne vse recepte za podani tip jedi skupaj z informacijo o številu manjkajočih sestavin za vsak recept.

V orodju najprej izberemo, da bomo kreirali WSDL iz datoteke Java. Vpišemo polno ime vmesnika v obliki paket.imeVmesnika in pokažemo na mapo, v kateri se nahajajo prevedene datoteke s končnico .class.

Zatem določimo vrednosti nekaterih atributov, kot so ciljni imenski prostor, njegova predpona, ciljni imenski prostor sheme in njegova shema ter ime storitve. Potem določimo še ime dokumenta WSDL in mapo, kamor naj se dokument shrani.

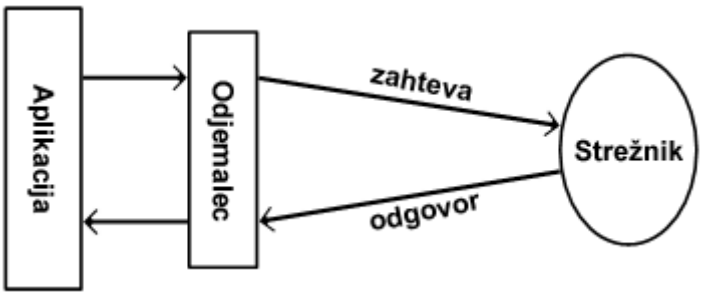
Na sliki 38 je prikazan izsek WSDL dokumenta spletne storitve *wsPoisciRecepte*.




*Slika 38. Izsek dokumenta WSDL, kot ga prikaže brskalnik.*

### 4.5.5. Kreiranje odjemalca

Za klicanje spletne storitve v aplikaciji ustvarimo odjemalca z uporabo programskega vmesnika Axis2. Odjemalec kliče spletno storitev na sinhroni način, kar pomeni, da pošlje sporočilo in čaka na odgovor. Slika 39 prikazuje shemo sinhronega načina klicanja storitve.




*Slika 39. Sinhroni način klicanja storitve.*

Izvorna koda odjemalca v aplikaciji je:

```
1 ServiceClient sc = new ServiceClient();
2 Options options = new Options();
3 opts.setTo(new EndpointReference(
      "http://localhost:8080/axis2/services/wsPoisciRecepte"));
4 opts.setAction("urn:predlogiZaTipJedi");
5 sc.setOptions(opts);
6 OMElement ws_odgovor = sc.sendReceive(
    kreirajPayLoad(tip_jedi, zalogaKodeXML, "predlogiZaTipJedi"));
```

V prvi vrstici se kreira objekt odjemalca `ServiceClient`, v drugi pa objekt za nastavitve `Options`. V tretji vrstici v nastavitve zapišemo URL naslov spletne storitve. V četrti zapišemo ime akcije soapAction, ki jo želimo sprožiti in jo najdemo v dokumentu WSDL. V peti vrstici zapišemo nastavitve v objekt odjemalca. Z metodo `sendReceive` v zadnji vrstici pokličemo storitev, še prej pa podatke s klicem funkcije `kreirajPayLoad` predstavimo kot element `OMElement` modela AXIOM (Axis Object Model).

Axis2 uporablja za predstavitev in procesiranje podatkov v obliki XML objektni model, imenovan AXIOM, zato moramo podatke predstaviti z uporabo modela AXIOM, preden jih predamo pogonu Axis2, ki jih pošlje naprej spletni storitvi. To definiram v funkciji `kreirajPayLoad`:

```
1  public static OMElement kreirajPayLoad(
        String tipJedi, String zalogaKodeXML, String imeMetode) {
2    OMFactory fac = OMAbstractFactory.getOMFactory();
3    OMNamespace omNs = fac.createOMNamespace("http://web_service", ns1");
4    OMElement method = fac.createOMElement(imeMetode, omNs);
5    OMElement tip_jedi = fac.createOMElement("v1", omNs);
6    OMElement rfid_kode_izdelkov = fac.createOMElement("v2", omNs);
7    tip_jedi.setText(tipJedi);
8    rfid_kode_izdelkov.setText(zalogaKodeXML);
9    method.addChild(tip_jedi);
10   method.addChild(rfid_kode_izdelkov);
11   return method;
12 }
```

V drugi vrstici funkcije se kreira objekt `OMFactory` (Object Model Factory), ki vsebuje metode za kreiranje ostalih objektov modela AXIOM, kot so `OMElement`, `OMNamespace`, `OMText`, `OMDocument`,... V tretji vrstici zapišemo ciljni imenski prostor. V vrsticah od štiri do šest kreiramo objekte za metodo *predlogZaTipJedi* in vhodna parametra spletne storitve *tipJedi* in *zalogaKodeXML*. V vrstici sedem in osem nastavimo vrednosti objektoma `tip_jedi` in `rfid_kode_izdelkov` in jima v vrstici devet in deset določimo, da pripadata

objektu metode *predlogZaTipJedi*. V vrstici enajst vrnemo odjemalcu referenco na objekt tipa `OMElement`, v katerem so metoda *predlogZaTipJedi* in vhodni parametri predstavljeni v obliki modela AXIOM.

### 4.5.6. Namestitev spletne storitve

Ko je spletna storitev napisana, jo je potrebno namestiti oz. postaviti na strežnik.

Za delovanje spletne storitve wsPoisciRecepte mora biti na strežniku Tomcat nameščen Axis2 in sicer distribucija za spletno aplikacijo, ki je v obliki datoteke WAR (Web Archive). Namestimo ga tako, da datoteko axis2.war skopiramo v mapo "$CATALINA_HOME/webapps", kjer je "$CATALINA_HOME" mapa, v kateri je nameščen strežnik Tomcat. Po namestitvi je potrebno Tomcat ponovno zagnati. Ob tem se v mapi *webapps* ustvari nova mapa z imenom *axis2* in s tem je Axis2 nameščen.

Eclipse za Java EE, v katerem sem razvijal spletno storitev, ima vgrajen enostaven, a zelo zmogljiv sistem za generiranje in preizkušanje spletnih storitev. Eclipse lahko namreč sam ustvari datoteko wsdl, odjemalca, proxy, spletno storitev postavi na strežnik in omogoči preizkus storitve preko že napisanih odjemalcev v JSP ali orodja raziskovalec spletnih storitev (ang. web service explorer). To testno okolje je odlično za preizkušanje spletnih storitev, vendar spletna storitev deluje le, dokler se Eclipse izvaja, kar predstavlja nepotrebno uporabo računalniških virov.

Da bo spletna storitev tekla na strežniku Tomcat, ki se samostojno izvaja, je potrebno kreirati arhivsko datoteko AAR (Axis Archive) in jo shraniti v podmapo strežnika. Datoteko sem ustvaril z orodjem Axis2 Code Generator. V orodju sem nastavil naslednje parametre:

- nastavil sem pot do mape z datotekami .class,
- izbral sem dokument WSDL, ki sem ga že ustvaril v prejšnjih korakih,
- dodal sem vmesnik JDBC kot zunanjo knjižnico,
- izbral sem samodejno kreiranje datoteke services.xml,
- določil sem ime datoteke spletne storitve in
- določil sem mapo, kamor naj se datoteka shrani (datoteko moramo shraniti v mapo "$CATALINA_HOME/webapps/axis2/WEB-INF/services").

Z določitvijo naštetih parametrov sem ustvaril arhivsko datoteko .aar spletne storitve in jo shranil na strežnik TOMCAT. Spletna storitev je s tem nameščena.

### 4.5.7. Preizkus spletne storitve

Spletna storitev *wsPoisciRecepte* dobi kot vhodna parametra tip jedi in podatke o izdelkih v obliki XML. Tako na primer klicanje metode *predlogZaTipJedi* spletne storitve z vrednostima parametrov "Juha" za tip_jedi in zapis XML za rfid_kode_izdelkov na sliki 40 vrne rezultat, ki je prikazan na sliki 41.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<seznamRFIDKodIzdelkov>
  <RFIDKodaIzdelka>0200010001002E43C901680168000C030000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>0200010006019D3D1902BC02BC03E8020000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>020001000800D64399003202BC03E8020000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>010001000701CAB5BA00280032010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>010001000A0010A9AC00320064010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>010002000C01BBE8A901F401F4010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>010002000D0098EC7901900320010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>0100020010002E1CB9001400F0010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>0100030013004E27D800820082010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>0100040017007BC789012C01F4010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>0100040017019CC7E904B004B0010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>010004001901CB79090050008C010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>010004001A016FC47900460050010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>010004001F00B7980900DC00DC010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>010004002000F4A109026C026C010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>010004002101228ED900280032010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>0100040023002FA359012C01F4010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>0100050026001034790 2EE03E8010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>01000300140132BB78018601F4010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>010002000E0010A9A801F401F4010000</RFIDKodaIzdelka>
  <RFIDKodaIzdelka>0200010001009A24F8007800960005030000</RFIDKodaIzdelka>
</seznamRFIDKodIzdelkov>
```


*Slika 40. Vrednost vhodnega parametra spletne storitve rfid_kode_izdelkov*

```
<recepti>
    <recept>
        <imeRecepta>Cvetačna juha</imeRecepta>
        <tipJedi>Juha</tipJedi>
        <seznamSestavin>
            <sestavina>
                <imeSestavine>Cvetača</imeSestavine>
                <kolicina>500 g</kolicina>
                <naZalogi>manjka</naZalogi>
            </sestavina>
            <sestavina>
                <imeSestavine>Jušna kocka</imeSestavine>
                <kolicina></kolicina>
                <naZalogi>manjka</naZalogi>
            </sestavina>
            <sestavina>
                <imeSestavine>Poper</imeSestavine>
                <kolicina></kolicina>
                <naZalogi></naZalogi>
            </sestavina>
            <sestavina>
                <imeSestavine>Sol</imeSestavine>
                <kolicina></kolicina>
                <naZalogi></naZalogi>
            </sestavina>
            <sestavina>
                <imeSestavine>Peteršilj</imeSestavine>
                <kolicina></kolicina>
                <naZalogi></naZalogi>
            </sestavina>
        </seznamSestavin>
        <priprava>Cvetačo razrežemo in dobro operemo. V malo vode jo
kuhamo 15 min.

Cvetačo nato odcedimo in zavremo 1,2 l juhe z jušno kocko. Juho začinimo
s soljo in poprom. Dodamo cvetačo in še enkrat na hitro prevremo.
Potresemo s peteršiljem.</priprava>
        <slika>cvetacna-juha.jpg</slika>
    </recept>
</recepti>
```

*Slika 41. Recept za juho, ki ga predlaga spletna storitev na podlagi poslanih izdelkov.*

# 5. Sklepne ugotovitve

Izdelana aplikacija za sistem inteligentnega hladilnika je pregledna in enostavna za uporabo. Uporabniški vmesnik je prilagojen uporabi z zaslonom na dotik, saj so gumbi dovolj veliki, da je navigacija preprosta. Vse implementirane funkcije so bile testirane in delujejo v skladu z zastavljenimi cilji.

Aplikacija prikazuje zalogo izdelkov, ki se nahajajo v bližini čitalnika RFID. Ko izdelek odmaknemo od čitalnika, se izdelek zbriše iz zaloge. Glede na zalogo se prikažejo možni recepti, ki jih lahko naredimo.

Testiranje aplikacije je potekalo v dveh fazah. V prvi fazi smo zalogo simulirali tako, da smo podatke o izdelkih na zalogi sami vnesli v bazo. V drugi fazi pa smo podatke o izdelkih zapisali na značke RFID. Kadar je bila značka v dometu čitalnika, so se podatki o izdelku zapisali v bazo in izdelek je postal del zaloge.

Prototip strojne in programske opreme inteligentnega hladilnika je bil razvit od samega začetka, saj je postavljena izvedba šele njegova prva različica. Temu primerne so bile tudi težave, s katerimi smo se srečevali. Pri tehtanju je bilo treba izdelke čisto približati čitalniku, ker je njegov domet le 10 cm. Ugotavljanje zalog ni bilo vedno zanesljivo, ker je bilo v okolju prisotno veliko šuma. Branje značke na istem mestu je bilo odvisno od lege značke glede na anteno. Oba čitalnika sta bila testna in sta se obnašala slabše kot sorodni komercialni produkti na trgu. Kovinski materiali v območju značk so vplivali na komunikacijo s čitalnikom.

Ker je sistem v začetni fazi razvoja in vsa uporabljena tehnologija še ni zrela, so možne še številne izboljšave, kot na primer:

- uporaba zanesljivejših čitalnikov z večjim dometom ali večjim številom anten;
- uporaba digitalne tehtnice z vgrajenim čitalnikom RFID;
- tehtanje, implementirano na policah, kjer so zloženi izdelki;
- razširitev vsebine:
    - povečanje števila izdelkov, receptov in tipov jedi;
    - razširitev podatkov o izdelkih – sestavine, energijska vrednost, primerna temperatura hranjenja izdelka;
    - podrobnejša kategoriziracija izdelkov, npr. po proizvajalcih; iste vrste izdelkov po lastnostih, po katerih se razlikujejo – npr. mleko z 0.5, 1.6 in 3.2% maščobe ipd.
- uporabniku prijaznejše vnašanje vsebine s pomočjo vnosnih obrazcev;
- analiza uporabnikovih navad glede na zgodovino zaloge in izbranih receptov;
- povezava aplikacije s spletno trgovino, tako da bi lahko izdelke, ki jih dodamo na nakupovalni listek, naročili preko spleta;

- dodajanje nastavitev, ki uporabniku omogočajo določanje količin izdelkov, ki naj bodo vedno na zalogi (ko je izdelka manj, se avtomatsko doda na nakupovalni listek ali kupi v spletni trgovini);

Vse naštete izboljšave, tako kot koncept inteligentnega hladilnika, temeljijo na pomembni predpostavki, da bodo v prihodnosti prehrambni izdelki opremljeni z značko RFID. Čas bo pokazal, če bo temu res tako in kdaj bomo takšen hladilnik ugledali tudi v trgovini.

# 6. Viri


[1] H. Bhatt, B. Glover, *RFID Essentials*, O'Reilly, 2006, pogl. 1.

[2] D. E. Brown, *RFID Implementation*, New York: McGraw – Hill, 2007, pogl. 2, 3, 5.

[3] E. Cerami, *Web Service Essentials*, O'Reilly, 2002.

[4] T. M. Connoly, C. E. Begg, *Database Systems, A Practical Approach to Design, implementation and Management*, Harlow: Addison Wesley, 2005, pogl. 20.

[5] N. Dai, L. Mandel, A. Ryman, *Eclipse Web Tools Platform: Developing Java Web Applications*, Addison-Wesley Professional, 2007

[6] R. Monson-Haefel, *J2EE Web Services*, Boston: Addison Wesley, 2003

[7] T. Erl, *Service-Oriented Architecture: Concepts, Technology and Design*, Prentice Hall, 2005

[8] M. Hasanuzzaman, R. Saidur, H. H. Masjuki, *Investigation of Energy Consumption and Energy Savings of Refrigerator-Freezer During Open and Closed Door Condition*, 2008
Dostopno na: http://www.ansijournals.com/jas/2008/1822-1831.pdf

[9] RFID Journal: The History of RFID Technology. Dostopno na: http://www.rfidjournal.com/article/view/1338/1/129

[10] The EPCglobal Network Demonstration. Dostopno na: http://www.epcglobalinc.org/about/media_centre/EPCglobal_Network_Demo.pdf

[11] Describe REST Web Services with WSDL 2.0. Dostopno na: http://www.ibm.com/developerworks/webservices/library/ws-restwsdl/

[12] What's new in WSDL 2.0. Dostopno na: http://webservices.xml.com/pub/a/ws/2004/05/19/wsdl2.html

[13] WSDL 2.0, A New Hope? or The Empire Strikes Back? Dostopno na: http://www.acs.org.au/vic/socsig/ACSSOCSIG-Talk-Aug%202006.pdf

[14] Web Services Description Language (WSDL) Version 2.0 Part1: Core Language. Dostopno na: http://www.w3.org/TR/wsdl

[15] Cover Pages: OASIS Consortium Members Approve Version 3 as an OASIS Standard. Dostopno na: http://xml.coverpages.org/ni2005-02-02-a.html

[16] Web Services – Carnegie Mellon Software Engineering Institute. Dostopno na: http://www.sei.cmu.edu/isis/guide/technologies/web-services.htm

[17] Java Architecture for XML Binding. Dostopno na: http://java.sun.com/developer/technicalArticles/WebServices/jaxb/

[18] The Python Web Services Developer: Messaging technologies compared. Dostopno na: http://www-128.ibm.com/developerworks/library/ws-pyth9/

[19] The State of Web Services. Dostopno na: http://resources.zdnet.co.uk/articles/features/0,1000002000,2137913,00.htm

[20] Web Services Glossary. Dostopno na: http://www.w3.org/TR/ws-gloss/

[21] PostgreSQL: About. Dostopno na: http://www.postgresql.org/about/

[22] About Java Technology. Dostopno na: http://www.sun.com/java/about/

[23] FAQ about Java. Dostopno na: http://java.sun.com/products/jdk/faq.html

[24] Domača stran razvojnega okolja Eclipse. Dostopno na: http://www.eclipse.org

[25] Axis2. Dostopno na: http://ws.apache.org/axis2/index.html

[26] Axis2 plugins for Eclipse. Dostopno na: http://ws.apache.org/axis2/tools/index.html

[27] XJC dodatek za Eclipse. Dostopno na: https://jaxb-workshop.dev.java.net/