UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Peter Dolenec

Sistem za nadzor zaloge z uporabo pametnega telefona za skeniranje črtne kode izdelkov

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Damjan Vavpotič

Ljubljana, 2012

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: 00210/2012
Datum: 02.04.2012

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

Kandidat: **PETER DOLENEC**

Naslov: **SISTEM ZA NADZOR ZALOGE Z UPORABO PAMETNEGA TELEFONA ZA SKENIRANJE ČRTNE KODE IZDELKOV**

INVENTORY CONTROL SYSTEM WITH USE OF A SMARTPHONE FOR BARCODE SCANNING

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V okviru diplomske naloge izdelajte delujoč prototip informacijskega sistema, ki bo s pomočjo skeniranja črtne kode na izdelkih omogočal nadzor zaloge izdelkov, kar bo osnova za izdelavo naročil za dobavitelja. Posebnost sistema naj bo, da za skeniranje namesto dragih namenskih naprav uporablja pameten telefon. Prototip naj poleg aplikacije za pameten telefon vključuje tudi strežnik s katerim se bo pameten telefon povezal in na katerem bodo shranjeni podatki o izdelkih in njihovi zalogi. Pripravite tudi analizo morebitnih podobnih že obstoječih aplikacij ter predstavite posebnosti vaše aplikacije. Naloga naj vključuje predstavitev vseh ključnih tehnologij, ki bodo uporabljene za izdelavo sistema.

Mentor:

doc. dr. Damjan Vavpotič

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU
diplomskega dela

Spodaj podpisani Peter Dolenec

z vpisno številko 63050025

sem avtor diplomskega dela z naslovom:

Sistem za nadzor zaloge z uporabo pametnega telefona za skeniranje črtne kode izdelkov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Damjana Vavpotiča
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., ang.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«

V Ljubljani, dne 21. 9. 2012 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju doc. dr. Damjanu Vavpotiču za vso pomoč in nasvete pri izdelavi diplomskega dela.
Prav tako se zahvaljujem vsem, ki so mi stali ob strani in me podpirali v času študija.

Kazalo

Seznam uporabljenih kratic in simbolov

- EDGE – Enhanced Data rates for Global Evolution; izboljšanje vrednosti hitrosti prenosa za globalni napredek
- UMTS – Universal Mobile Telecommunications System; univerzalni sistem za mobilno komunikacijo
- iOS – Operation System; operacijski sistem
- URL – Uniform Resource Locator; naslov spletnih strani v svetovnem spletu
- HTML – Hypertext Markup Language; označevalni jezik za oblikovanje večpredstavnostnih dokumentov, ki omogoča povezave znotraj dokumenta ali med dokumenti
- PHP – PHP Hypertext Preprocessor; jezik za razvoj dinamičnih spletnih strani
- ASP – Active Server Page; aktivno kreiranje spletnih strani
- JSON – JavaScript Object Notation; označevalni jezik
- XML – Extensible Markup Language; razširljiv označevalni jezik
- GCC – GNU Compiler Collection; prevajalniška zbirka GNU
- SDK – Software Development Kit; zbirke knjižnic za razvoj aplikacij
- API – Application Program Interface; programski vmesnik
- UML – Unified Modeling Language; poenoteni jezik za modeliranje

Povzetek

V diplomskem delu je predstavljen informacijski sistem, ki omogoča nadzor zaloge s pomočjo skeniranja črtne kode izdelka. Informacijski sistem je sestavljen iz dveh strežnikov ter iz mobilne aplikacije, ki je narejena za iOS platformo. Aplikacija temelji na programskem jeziku Objective C in je bila izdelana v razvojnem okolju Xcode.

V diplomskem delu je najprej predstavljen mobilni telefon, ki omogoča uporabo te aplikacije, ter operacijski sistem, katerega uporablja. Predstavljeno je tudi razvojno okolje ter posamezne tehnologije, ki so bile uporabljene za izdelavo aplikacije.

Glavni del diplomskega dela opisuje izdelavo aplikacije. Predstavlja sestavne dele aplikacije ter opisuje vse funkcionalnosti, ki jih aplikacija omogoča. V zadnjem delu pa sta opisana tudi oba spletna strežnika, ki sta za samo delovanje aplikacije zelo pomembna, saj brez njiju aplikacija ne bi mogla prikazati drugega kot samo prebrano šifro brez ostalih podatkov.

Ključne besede:

iOS, aplikacija, črtne kode, izdelava naročil, spletni strežnik

Abstract

The diploma presents the information system that allows you to monitor stock by scanning the barcode of the product. Information system consists of two servers and the mobile application that is designed for iOS platform. The application is based on programming language Objective C and was developed in Xcode development environment.

The diploma starts with the description of devices that allow execution of the application and the operating system that they use. Presented are also a development environment and the particular techniques that have been used for making application.

The main part describes the development of the application. It mainly presents components of the application and describes all the functionality that the application provides. Final part is devoted to both web servers that are very important. Without those servers the application could not show any other data than just code.

Key words:

iOS, application, barcode, orders creating, web server

1 Uvod

Danes je nakupovanje s pomočjo spletnih trgovin že zelo razširjeno. Tako tudi marsikateri obrtnik za naročanje potrebnega blaga uporablja spletno trgovino svojega dobavitelja. Vsakodnevno naročanje preko spleta pa je lahko tudi zelo zamudno, zato nekateri še vedno raje pošiljajo naročila preko elektronske pošte ali pa preko faksa, kar pa dobaviteljem povzroča dodatno delo, saj morajo vsa naročila ročno prepisati v svoj informacijski sistem. Na tak način se poveča tudi možnost za kakšno napako, kar pa prinaša nezadovoljstvo tako naročniku kot tudi dobavitelju.

Obstaja tudi možnost, da bi kupci uporabljali črtne kode ter laserske skenerje črtnih kod. Vendar je taka rešitev lahko precejšen finančni zalogaj, hkrati pa to prinese tudi dodatne težave, saj je v večini primerov potrebno imeti tudi posebno programsko opremo za izdelavo in pošiljanje naročil.

Druga možnost pa je, da za zajem črtnih kod uporabljajo pametne mobilne telefone. Na tržišču obstaja veliko aplikacij, ki znajo prebrati črtno kodo ter prikazati podatke. Kljub temu jih je precej generičnih, saj podatke iščejo na raznih spletnih brskalnikih ali pa so namenjene izdelavi nakupovalnih seznamov. Nekaj aplikacij pa je narejenih posebej za uporabo v skladiščih in so povezane direktno z informacijskim sistemom, tako da lahko prikazujejo tudi stanje zaloge, vendar pa za svoje delovanje potrebujejo neprekinjeno, hitro povezavo z informacijskim sistemom, kar pa je lahko v skladišču tudi težava. Nekatere od teh aplikacij pa poleg tega potrebujejo tudi posebne laserske skenerje.

Naša zamisel je bila izdelati aplikacijo, ki bi za zajem črtne kode uporabljala vgrajeno kamero, podatki o artiklih pa bi bili se prenašali iz spletne trgovine dobavitelja in bi se shranjevali na mobilni telefon, kar bi omogočalo tudi dostop do podatkov brez internetne povezave. Aplikacijo bi bilo mogoče povezati tudi z internim informacijskim sistemom, kar bi omogočalo prikaz zaloge vsakega artikla. Na podlagi podatka o zalogi bi se uporabnik lažje odločil, kolikšno količino bo naročil. Aplikacija bi omogočala tudi izdelavo naročila, katerega bi lahko uporabnik poslal takrat, ko ima internetno povezavo, hkrati pa bi vsako naročilo imelo nek status, ki bi uporabniku povedal, v kateri fazi se nahaja.

Pametni mobilni telefoni danes uporabljajo predvsem dva operacijska sistema, iOS in Android. Odločili smo se, da bomo naredili aplikacijo za iOS operacijski sistem.

Uvodnemu poglavju sledi poglavje, v katerem na kratko predstavi iPhone, operacijski sistem iOS ter življenjski cikel aplikacije in razvojna orodja na strežniški strani. Tretje poglavje predstavlja orodja in tehnologije, s katerimi je bil razvit celoten projekt. Prav tako predstavlja dodatne knjižnice, katere smo potrebovali za razvoj projekta. Četrto poglavje opisuje posamezne dele razvite aplikacije ter opisuje njihove funkcionalnosti. Zaključne misli pa so podane v petem poglavju.

2 Razvojna platforma

2.1 iPhone

iPhone (slika 1) je pametni mobilni telefon podjetja Apple in je bil prvič predstavljen leta 2007. Za delovanje uporablja operacijski sistem iOS.

Za prenos podatkov iz spleta uporablja tehnologiji EDGE in UMTS, omogoča pa tudi priklop na brezžično omrežje Wi-Fi.

Za vnos podatkov uporablja virtualno tipkovnico in na dotik občutljiv ekran. Ker je tipkovnica virtualna, se lahko prilagaja vsaki aplikaciji oziroma potrebi primerno (npr. če je potrebno vnašati samo številke, tipkovnica prikaže samo številke). Poleg tega pa ima tudi gumb domov, ki omogoča izhod iz trenutno aktivne aplikacije oziroma prehod na domači zaslon (na namizje).

Slika 1: iPhone 4S

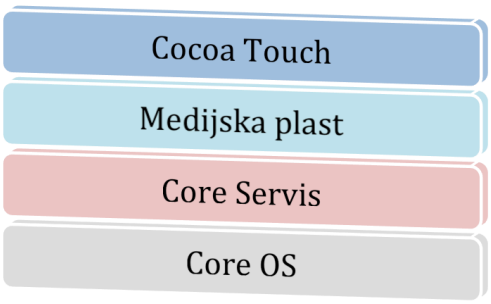
2.2 iOS

iOS je mobilni operacijski sistem, ki ga je razvilo podjetje Apple. Prvič je bil predstavljen skupaj s prvo generacijo iPhone-a leta 2007. Sprva ga je uporabljal samo iPhone, sedaj pa ga uporabljajo tudi iPod touch, iPad ter Apple TV.

iOS izvira iz operacijskega sistema OS X (operacijski sistem za Apple računalnike), ki pa temelji na Unix operacijskem sistemu. Uporabniški vmesnik je bil razvit posebej za več-dotične zaslone, kar pomeni, da so ikone ter gumbi dovolj veliki, da jih lahko pritisnemo s prstom.

Četrta generacija je prinesla največjo spremembo, ker je bila prva, ki je omogočala večopravilnost, tako da so se aplikacije lahko izvajale tudi v ozadju. Peta generacija pa je med drugim prinesla integracijo s spletno storitvijo iCloud ter center za obvestila.

iOS sestavljajo štiri abstraktne plasti, kot je prikazano na sliki 2. V spodnjih plasteh sistema so temeljne storitve in tehnologije, na katerih slonijo vse aplikacije, na višjih plasteh pa so bolj zapletene storitve in tehnologije.



Slika 2: Plasti iOS

Plast Cocoa Touch je vrhnja plast v iOS operacijskem sistemu in vsebuje ogrodja, ki jih razvijalci aplikacij največkrat uporabljajo. Napisana je v programskem jeziku Objective C. Temelji na standardnem sistemu Mac OS X Cocoa API (kot je na voljo na Apple namiznih in prenosnih računalnikih) in je bila prilagojena specifičnim potrebam na mobilnih napravah. Definira osnovno aplikacijsko infrastrukturo in podporo tehnologijam, kot so večopravilnost, večdotičnost, opozorila, kamera in mnoge druge visokonivojske sistemske storitve.

Vloga medijske plasti je, da zagotovi operacijskemu sistemu avdio, video, animacije in grafične zmogljivosti. Tako kot druge plasti je tudi medijska plast sestavljena iz več ogrodij, ki se lahko uporabljajo pri razvoju iOS aplikacij.

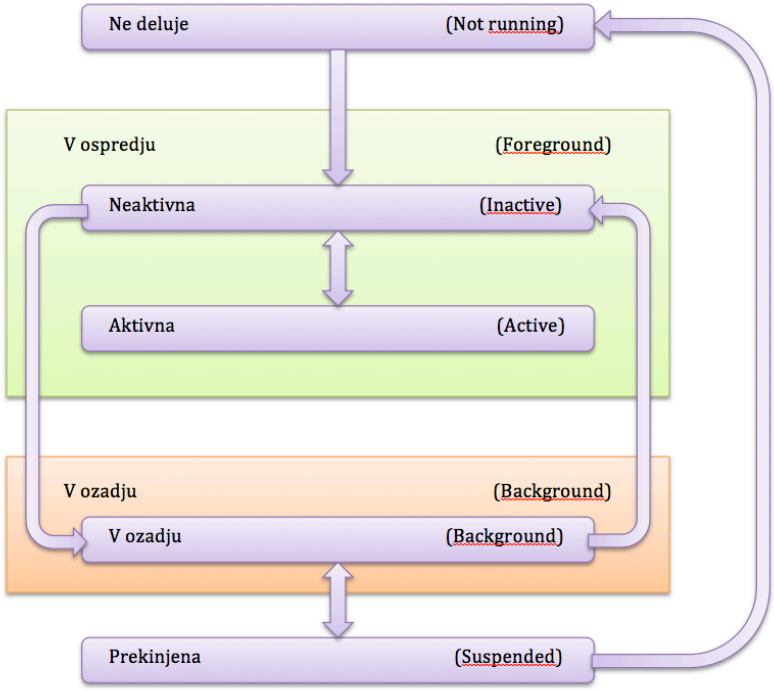
Core Service plast vsebuje temeljne sistemske storitve, kot so omrežje, dostop do datotek, SQLite, nitenje, nastavitve in mnoge druge.

Plast Core OS je spodnja plast v iOS operacijskem sistemu in se nahaja neposredno nad strojno opremo naprave. Zagotavlja različne storitve, vključno z nizkostopenjskim upravljanjem omrežja, dostopom do zunanjih naprav, varnosti ter običajnih storitev operacijskih sistemov, kot so upravljanje s pomnilnikom, datotečnim sistemom, nitmi ter upravljanje s porabo električne energije.

V operacijskem sistemu iOS imamo na voljo več različnih načinov za hranjenje podatkov. Razlikujejo se predvsem po tem, za kakšno količino podatkov so primerni. Podrobneje so predstavljeni v naslednjem poglavju, kjer so predstavljene tehnologije, katere smo uporabili za izdelavo aplikacije.

2.3 Življenjski cikel aplikacije

Aplikacija se v vsakem trenutku nahaja v enem izmed petih stanj. Sistem premika aplikacijo iz enega stanja v drugega glede na dogodke, ki se dogajajo v celotnem sistemu. Slika 3 prikazuje prehode aplikacije iz enega stanja v drugega.



Slika 3: Življenjski cikel aplikacije

V stanju ne deluje (ang. not running) je aplikacija pred prvim zagonom ali pa je bila aplikacija zaključena s strani uporabnika oziroma operacijskega sistema.

Aplikacija je v osredju, kadar je prikazana na ekranu. Hkrati je lahko v ospredju samo ena aplikacija. Če je aplikacija v ospredju, se lahko nahaja v dveh različnih stanjih: kot aktivna ali neaktivna.

Medtem ko je aplikacija v ospredju in je aktivna (ang. active) ter upravlja s trenutnimi dogodki (npr. interakcija z uporabnikom), pa je neaktivna (ang. inactive) takrat, ko ne prejema

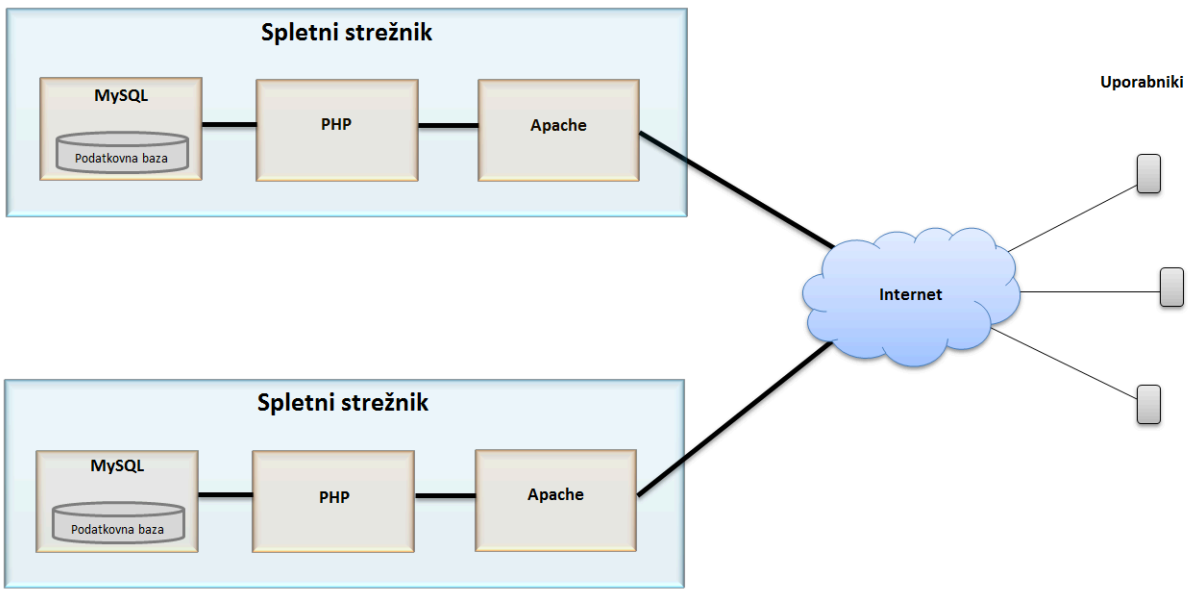
ali upravlja nobenih dogodkov. Običajno je neaktivna samo za kratek čas, ko prehaja med stanji.

Aplikacija tipično preide v ozadje (ang. background), ko uporabnik zažene drugo aplikacijo. Aplikacije v ozadju se lahko še vedno izvršujejo, če zahtevajo dodaten čas za dokončanje izvajanja, ali pa, ker so zaprosile za delovanje v ozadju. Od operacijskega sistema iOS 4 dalje pa je mogoče aplikacijo zagnati tudi direktno v ozadje.

Aplikacija preide v prekinjeno (ang. suspended) stanje, ko je premaknjena v ozadje in se zaključi izvrševanje. V tem stanju se aplikacija še vedno ohranja v pomnilniku, vendar s tem ne obremenjuje niti procesorja niti ne dodaja dodatnih obremenitev za baterijo. To stanje omogoča aplikaciji, ob zahtevi uporabnika, hiter prehod v ospredje in nadaljevanje izvajanja ter s tem zagotavlja skoraj takojšnje preklapljanje med aplikacijami. Operacijski sistem lahko aplikacije v tem stanju kadar koli zaustavi in odstrani iz pomnilnika. Običajno se to zgodi takrat, kadar pride do pomanjkanja prostora v pomnilniku.

2.4 Strežniški del

Strežnika sta potrebna za pravilno delovanje aplikacije, saj so na enem shranjeni podatki o artiklih ter uporabnikih, hkrati pa se na strežniku tudi obdelajo prejeta naročila, na drugem pa podatki o zalogi. Na Apache strežniku se izvajajo PHP skripte, ki s pomočjo MySQL podatkovnih baz skrbijo za pravilno obdelavo in shranjevanje podatkov. Zgradba ter povezave med strežnikoma in mobilno aplikacijo je prikazana na sliki 4.



Slika 4: Arhitektura strežnikov

2.4.1 Apache

Apache spletni strežnik (pogosto samo Apache) je igral pomembno vlogo pri širjenju svetovnega spleta. Leta 2009 je postal prvi spletni strežnik, ki je presegel število 100 milijonov spletnih strani. Od aprila 1996 je Apache najbolj priljubljen spletni strežnik. Marca 2012 je bilo ocenjeno, da je na njemu postavljenih 57,46 % aktivnih spletnih strani ter da 65,24 % strežnikov na vseh platformah uporablja Apache.

Apache spletni strežnik podpira več različnih operacijskih sistemov. Mi smo ga uporabili na operacijskem sistemu OS X Mountain Lion (10.8). Zadnja stabilna verzija Apache strežnika je 2.4.3 in je bila izdana 21. avgusta 2012.

2.4.2 MySQL

MySQL je najpogosteje uporabljen odprtokodni sistem za upravljanje s podatkovnimi bazami, ki se uporablja kot strežnik, ki omogoča več uporabnikom hkrati dostopati do podatkov v podatkovnih bazah. MySQL je priljubljena izbira za upravljanje s podatkovnimi bazami pri spletnih aplikacijah.

MySQL podpira več različnih platform. Zadnja stabilna verzija je 5.5.27 in je bila izdana 2. avgusta 2012.

2.4.3 PHP

PHP je razširjen odprtokodni programski jezik, ki se uporablja za strežniške uporabe oziroma za razvoj dinamičnih spletnih strani. PHP skripta je združena s HTML dokumentom, katerega spletni strežnik interpretira in s pomočjo PHP procesnega modula generira končno spletno stran. Lahko ga primerjamo z Microsoftovim jezikom ASP, VBScript ter JScript.

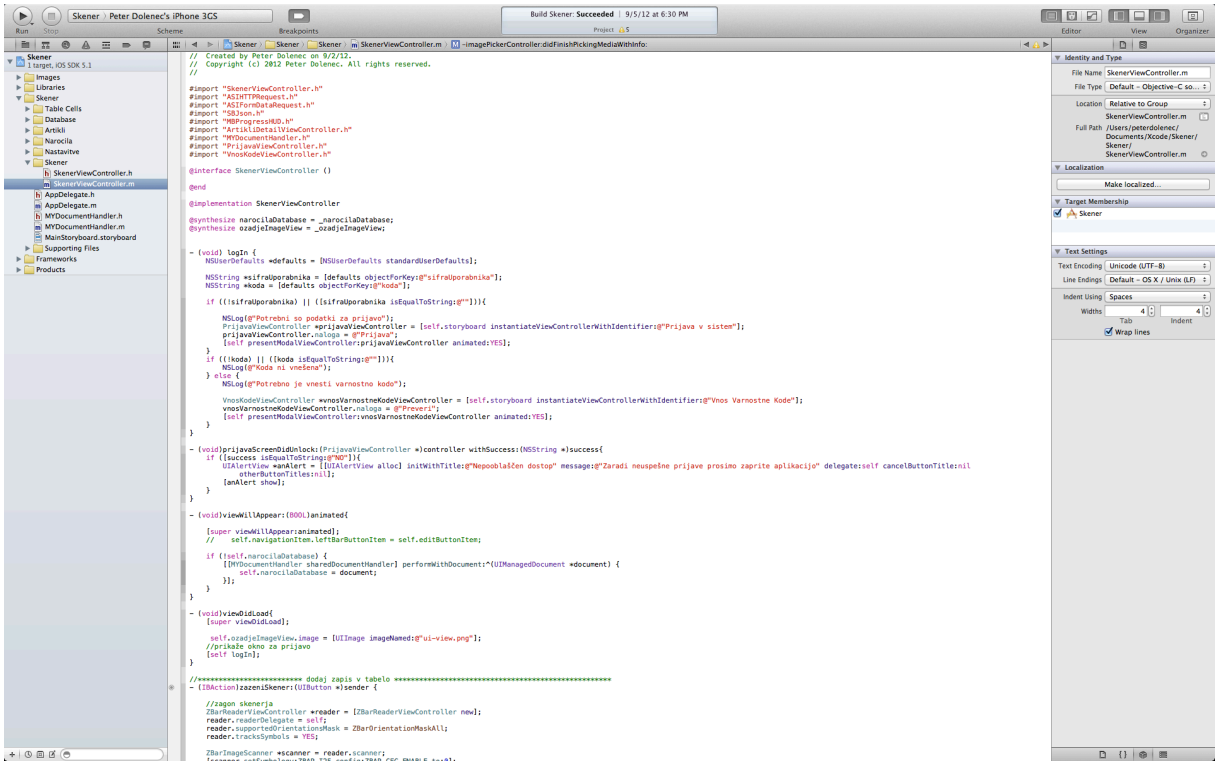
PHP podpira več različnih platform. Zadnja stabilna verzija je 5.4.6 in je bila izdana 15. avgusta 2012.

3 Uporabljena orodja in tehnologije

Za razvoj aplikacije smo uporabili Xcode skupaj z iOS SDK-jem. Za programiranje smo uporabili programski jezik Objective C. Za delovanje programa pa smo uporabili tudi nekaj knjižnic drugih razvijalcev.

3.1 Xcode

Xcode je razvojno okolje, ki za delovanje potrebuje operacijski sistem OS X in omogoča razvoj aplikacij tako za OS X platformo kot tudi za iOS platformo. Razvilo ga je podjetje Apple.



Slika 5: Osnovno okno programa Xcode

Xcode (slika 5) razvojno okolje združuje vsa orodja, ki so potrebna za razvoj aplikacije. Omogoča tako pisanje kode in razhroščevanje kot tudi izdelavo uporabniškega vmesnika. Poleg tega je vgrajen tudi Apple LLVM prevajalnik, ki sproti med pisanjem programske kode preverja sintakso. Če zazna napako, jo označi ter ponudi predlog, kako jo popraviti, med samim pisanjem kode pa tudi pomaga s predlogi. To pride prav predvsem pri uporabi dolgih imen spremenljivk, ker naj bi bila sama programska koda čim bolj podobna normalnemu angleškemu jeziku, kar pa lahko pripelje do precej dolgih imen spremenljivk.

Poleg tega so priložena tudi orodja za analizo samega programa. Tako lahko analiziramo delovanje programa in odkrijemo, če se kateri deli programske kode izvajajo predolgo časa in s tem upočasnjujejo delovanje samega programa, kar nam omogoča nadaljnjo optimizacijo.

Priložen pa je tudi simulator naprav (slika 6), ki lahko simulira tako iPhone kot tudi iPad. Tako lahko nekatere osnovne stvari testiramo kar na računalniku, hkrati pa nam je v pomoč tudi razhroščevalnik. Na žalost ne moremo simulirati uporabe kamere (lahko samo izderemo sliko iz knjižnice). Opazili pa smo tudi, da ni nujno, da aplikacija, ki dela na simulatorju, dela tudi na dejanski napravi. Ker ima računalnik drugačen procesor in več pomnilnika, pa lahko pride tudi do razlike v hitrosti izvajanja aplikacije.

Slika 6: iPhone simulator

3.2 Objective C

Objective C je visokonivojski objektno usmerjen programski jezik, ki je zasnovan na programskem jeziku C. Zasnovan je tako, da programskemu jeziku C omogoča popolno objektno programiranje. Njegovi dodatki k C-ju temeljijo večinoma na programskem jeziku Smaltalk, ki je bil eden prvih objektno usmerjenih programskih jezikov. Prvotno je bil jezik uporabljen za izdelavo operacijskega sistema NEXTSTEP, katerega je kasneje kupilo podjetje Apple in ga uporabilo v svojem operacijskem sistemu OS X. Ker je Objective C samo nadgradnja programskega jezika C, lahko v isti datoteki uporabljamo oba programska jezika.

Razlog za uporabo programskega jezika Objective C je, da ustvari objektno usmerjeno okolje, znotraj katerega izdelujemo programe. Objektno usmerjeno programiranje je programska paradigma, ki poizkuša omogočiti razvijalcem, da pri oblikovanju programske opreme razmišljajo o objektih in atributih namesto o spremenljivkah in funkcijah. Z drugimi besedami objektno usmerjeno programiranje uporablja abstrakcijo podatkov, enkapsulacijo, modularnost, polimorfizem in dedovanje. Eden izmed najboljših razlogov,zakaj uporabljati objektno usmerjeno programiranje je, ker na podoben način delujejo tudi možgani. Obstaja pa tudi veliko koristi, da lahko na podoben način abstraktno razvijamo programsko opremo.

Razredi (in s tem objekti) so sestavljeni iz metod in lastnosti. Pri drugih programskih jezikih bi lahko metode enačili s funkcijami in atribute s spremenljivkami.

3.3 Cocoa Touch

Cocoa Touch je ogrodje (ang. Framework) za izdelavo programov, ki so namenjeni iOS operacijskemu sistemu. Je zgornja abstraktna plast v iOS operacijskem sistemu. Temelji na Mac OS X Cocoa API in je ravno tako napisana s programskim jezikom Objective C. Cocoa Touch je posebej prilagojena iOS napravam z edinstvenim uporabniškim vmesnikom ter podporo za zaznavanje več hkratnih dotikov zaslona in posebne kretnje, ki se uporabljajo za interakcijo z objekti na ekranu. Pri izdelavi programov se uporablja arhitekturni stil model-pogled-nadzornik (ang. Model-View-Controller). Vsa orodja za razvoj aplikacij s pomočjo Cocoa Touch so vgrajena v iOS SDK.

3.4 Arhitekturni stil

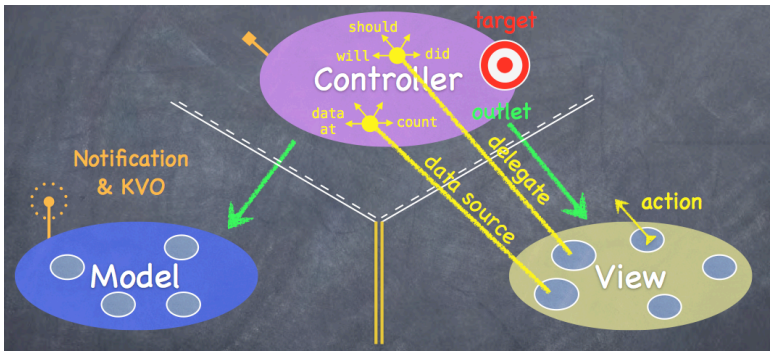
Pri izdelavi aplikacij za iOS uporabljamo arhitekturni stil model-pogled-nadzornik (ang. Model-View-Controller oz. MVC). Glavna značilnost tega arhitekturnega stila je razdelitev predstavitve, interakcije in podatkov na tri logične komponente (pogled, nadzornik, model).

- **Model** upravlja obnašanje sistema, podatke in pripadajoče operacije nad njimi. Model ne more pošiljati sporočil nadzorniku ali pogledu, vendar lahko pride do spremembe podatkov, takrat pa mora model obvestiti nadzornika, da je prišlo do spremembe. Zaradi tega model uporablja sistem radijske postaje in na tak način pošlje obvestilo nadzorniku, da je prišlo do spremembe podatkov, nato pa nadzornik preveri, do kakšne spremembe je prišlo ter temu ustrezno spremeni pogled.

- **Pogled** določa in upravlja predstavitev podatkov uporabniku (prikaz modela). Pogled mora biti čim bolj generičen. Uporablja navadne objekte (gumbe, drsnike itd.) katerim pa

nadzornik določi, kakšno funkcijo imajo (npr. nadzornik pove, kaj se bo zgodilo ob pritisku na gumb). Pogled ne more komunicirati z modelom, lahko pa komunicira z nadzornikom. Uporablja lahko več načinov komunikacije. Eden izmed načinov komunikacije je, da pogled obvesti nadzornika, da je prišlo do nekega dogodka (npr. pritiska na gumb). Za to uporabi način »target-action«. Včasih se mora pogled sinhronizirati z nadzornikom, za to uporabljata »delegate«. Ker pogled ni lastnik podatkov, katere prikazuje, uporablja poseben protokol za pridobivanje podatkov, ki jih pridobi od nadzornika, imenovan »data source«.

- **Nadzornik** upravlja interakcijo z uporabnikom in prenaša interakcijo modelu in pogledu ter tako povezuje model in pogled. Nadzornik lahko komunicira z modelom in tudi ve vse o njem. S pogledom komunicira s pomočjo posebnih lastnosti, ki določajo pogled, imenovanimi »outlet«. Sam način komunikacije prikazuje slika 7.



Slika 7: Prikaz komunikacije med modelom, nadzornikom in pogledom (vir: www.stanford.edu)

Taka arhitektura omogoča spremembo podatkov neodvisno od njihove predstavitve in obratno, podpira predstavitev istih podatkov na več načinov, kar prinese lažje vzdrževanje uporabniškega vmesnika. Prednosti se predvsem pokažejo, če delamo aplikacijo za iPhone in za iPad, saj model ostaja popolnoma enak, spremeniti je potrebno samo pogled ter malo prilagoditi nadzornik, saj so za iPad na voljo še nekateri drugi gradniki, ki pa niso na volji za iPhone. Če pa uporabljamo iste gradnike za obe napravi, pa samo prilagodimo pogled, da izkoristimo večji ekran.

3.5 Podatkovni sistem

V operacijskem sistemu iOS aplikacije uporabljajo vsaka svoj peskovnik. Tako so podatki zaščiteni pred nepooblaščenim dostopom iz drugih aplikacij, čeprav je mogoče s pomočjo URL sistema podatke ponuditi tudi drugim aplikacijam. Operacijski sistem iOS omogoča več različnih načinov za shranjevanje podatkov.

- **NSUserDefaults** je namenjen shranjevanju majhne količine podatkov, običajno shranjevanju uporabnikovih nastavitev. Shranjujemo lahko objekte, ki spadajo v razrede: NSData, NSString, NSNumber, NSDate, NSArray ter NSDictionary. Če želimo shraniti kakšno drugo obliko podatkov (npr. sliko), pa je potrebno te podatke oviti z enim od prej naštetih razredov (npr. sliko lahko ovijemo z razredom NSData).

- **SQLite** je sistem za upravljanje relacijskih podatkovnih baz. Je zelo hiter in zanesljiv, porabi malo pomnilnika, kar je na mobilnih napravah zelo pomembno, vendar ni najboljši pri upravljanju večjih objektov, kot so slike, glasba ali video.

- **Core Data** je objektno usmerjena podatkovna baza. Upravljanje s podatki poteka z objekti na višjem nivoju, kjer jih predstavljajo entitete ter relacije tako, da ne delamo z vrsticami in tabelami. Je tudi zelo optimizirana, kar pomeni zelo hitro izvajanje tudi na večji količini podatkov, omogoča pa tudi shranjevanje večjih objektov, kot so slike ali celo video posnetki, čeprav je za video posnetke bolje, če jih shranimo v datotečni sistem in v podatkovno bazo shranimo samo povezavo do njega.

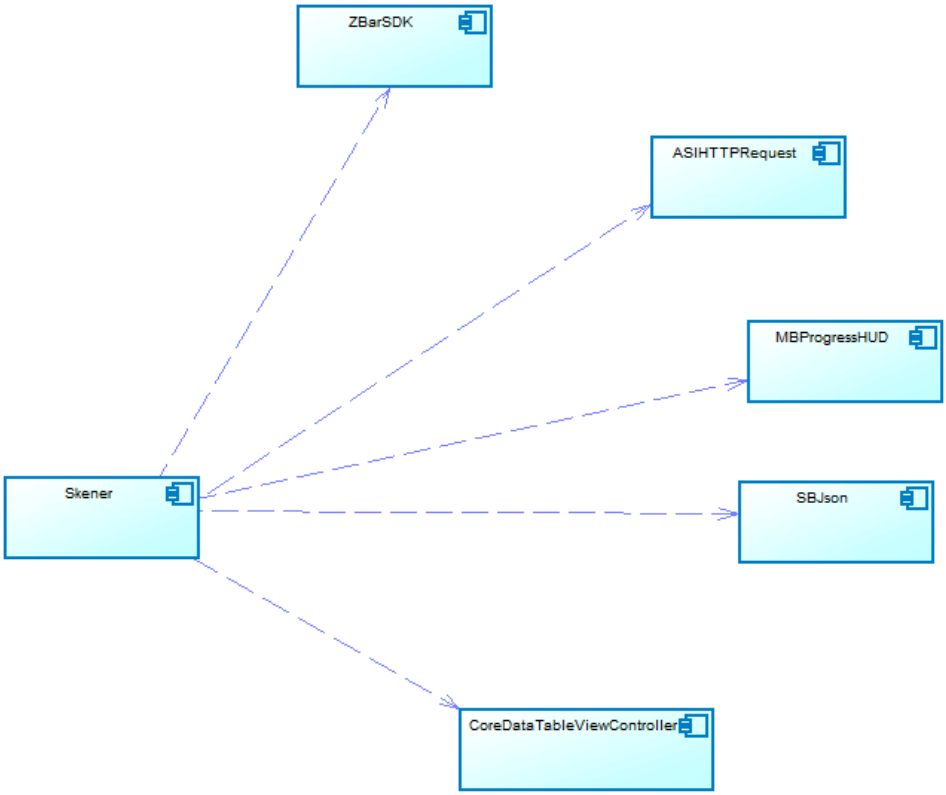
- **Datotečni sistem** je enak normalnemu Unix datotečnemu sistemu. Ravno tako so omejitve kot na Unix sistemu, tako da ne moremo videti in dostopati do vseh datotek. Aplikacija lahko shranjuje podatke samo znotraj svojega peskovnika, kar poskrbi za večjo varnost, saj druge aplikacije ne morejo dostopati do podatkov ter jih poškodovati, hkrati pa tudi omogoča, da se pri izbrisu aplikacije iz naprave odstranijo tudi vsi podatki znotraj peskovnika. Datotečni sistem je primeren predvsem za shranjevanje večje količine podatkov, kot so slike, glasba ter videoposnetki.

3.6 Kategorije

Pri izdelavi podatkovne baze (Core Data) nam Xcode na podlagi podatkov o entitetah in relacijah avtomatsko generira ustrezne podrazrede. Včasih pa želimo pri teh podrazredih dodati še nekaj svoje kode, nato pa spremenimo našo podatkovno shemo in pri ponovnem generiranju podrazredov se naša koda izgubi. Zato uporabljamo kategorije, s pomočjo katerih lahko dodajamo kodo razredu ali podrazredu brez spreminjanja originalnega razreda ali podrazreda. Kategorije se pogosto uporabljajo pri uporabi Core Data, vendar jih lahko uporabljamo za vse razrede, celo za tiste, do katerih niti nimamo dostopa. Tako lahko vsakemu razredu dodamo poljubne metode.

3.7 Uporabljene knjižnice

Za izdelavo programa smo uporabili tudi nekaj prosto dostopnih knjižnic iz spleta. Slika 8 prikazuje, kako so te knjižnice uporabljene znotraj programa. Ko uporabnik zažene skener, se najprej uporabi knjižnica ZBarSDK, ki zajame in analizira črtno kodo. Nato s pomočjo knjižnice ASIHTTPRequest pošljemo zahtevo na spletni strežnik, pri tem pa uporabimo tudi knjižnico MBProgressHUD, ki uporabniku prikaže animacijo prenosa podatkov. Ko prejmemo zahtevane podatke, uporabimo knjižnico SBJson, s pomočjo katere razčlenimo vrnjene podatke. Na koncu pa uporabimo še knjižnico CoreDataTableViewController, ki osveži prikazane podatke v tabeli artiklov.



Slika 8: Komponenti diagram knjižnic

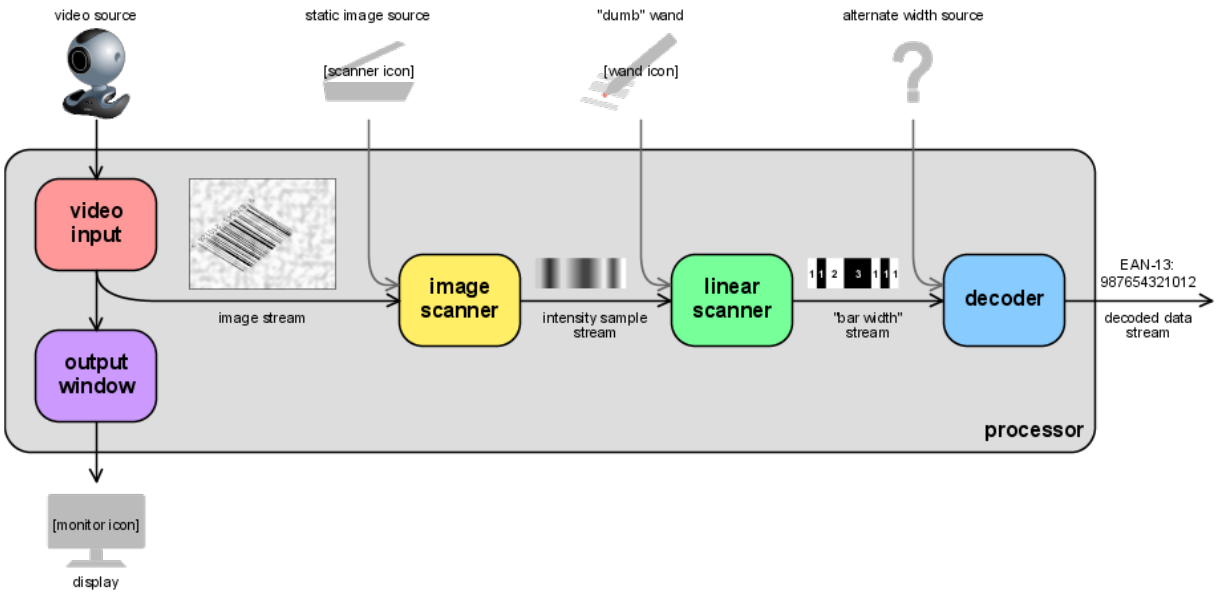
3.7.1 ZBarSDK

ZBarSDK je odprtokodna knjižnica in je ključna za izdelavo našega programa, saj omogoča zajemanje slike iz kamere ali pa iz knjižnice slik, nato pa to sliko tudi analizira ter vrne končen rezultat. Podpira več priljubljenih standardov črtnih kod, kot so: EAN-13/UPC-A, UPC-E, EAN-8 Code 128, Code 39, Interleaved 2 od 5 ter tudi QR kode. Knjižnica je v osnovi napisana v programskem jeziku C, vendar vsebuje vmesnik, tako da lahko uporabljamo programski jezik Objective C.

Običajen postopek za branje črtnih kod iz slike je, da na sliki, ki vsebuje črtno kodo, uporabimo digitalno procesiranje slike. To pomeni, da v več korakih uporabimo razne filtre ter tako izostrimo sliko, odstranimo šume in povečamo kontrast. Na tak način omogočimo zaznavanje robov ter oblik, kar nato analiziramo, določimo lokacijo simbolov ter orientacijo. Na koncu iz tega dobimo podatke. Vse to pa zahteva veliko procesorskega časa in prostora v pomnilniku, hkrati pa je sam postopek zelo občutljiv na parametre pri posameznih filtrih. Ker je potrebno veliko raznih nastavitev, pa je to lahko zelo težko razumljivo za končnega uporabnika.

ZBar knjižnica pa uporablja drugačen pristop, bolj podoben tistemu, ki ga uporabljajo laserski skenerji. Linearne črtne kode so načrtovane tako, da jih lahko dekodira preprost svetlobni senzor, ki potuje čez svetle in temne dele črtne kode. Implementacija ZBar skenerja uporablja linearni skener, ki skenira sliko in vsako piko obravnava kot vzorec iz enega svetlobnega senzorja. Tako so podatki skenirani, dekodiranji in sestavljeni skorajda v trenutku.

ZBar tako sestavi večplasten model. Procesiranje je razdeljeno na posamezne neodvisne plasti z dobro definiranimi vmesniki. Tako lahko plasti uporabimo skupaj ali pa kot posamezne plasti, ki so povezane z drugimi sistemi. Delovanje je prikazano na sliki 9.



Slika 9: Večplastno delovanje skenerja (vir: zbar.sourceforge.net)

3.7.2 SBJson

JSON je odprt standard, ki se uporablja za izmenjavo podatkov (običajno med klientom in strežnikom). Zasnovan je tako, da je lahko berljiv tudi za ljudi. JSON je neodvisen od

programskih jezikov, tako da se ga lahko uporablja skupaj z večino programskimi jeziki. Edino potrebno je za vsak programski jezik napisati razčlenjevalnik. Za komunikacijo med strežnikom in klientom lahko uporabljamo tudi XML, ki predstavlja alternativo JSON-u.

SBJson je okvir napisan za JSON (ang. JavaScript Object Notation) v Objective C. Z drugimi besedami to pomeni, da lahko JSON uporabljamo v programskem jeziku Objective C, hkrati pa poskrbi tudi za razčlenjevanje tako poslanih kot tudi prejetih sporočil.

3.7.3 ASIHTTPRequest

ASIHTTPRequest je enostavna knjižnica, ki je ovita okrog CFNetwork API, kar omogoča enostavnejšo komunikacijo s spletnim strežnikom. Napisan je v Objective C in ga lahko uporabljam tako v Mac OS X kot tudi v iOS aplikacijah. Primeren je za uporabo osnovnih HTTP zahtev ter interakcijo z REST spletnimi strežniki. Osnovne operacije na REST spletnem strežniku so: preberi, ustvari nov zapis, posodobi zapis, izbriši zapis (GET, POST, PUT, DELETE).

Primer programske kode, ki s pomočjo knjižnice ASIHTTPRequest pošlje podatke o artiklu na strežnik:

```
NSURL *url = [NSURL URLWithString:[defaults objectForKey:@"url2"]];
ASIFormDataRequest *request = [ASIFormDataRequest requestWithURL:url];
[request setPostValue:artikel.sifra forKey:@"sifra"];
[request setPostValue:artikel.naziv forKey:@"naziv"];
[request setPostValue:artikel.cena forKey:@"cena"];
[request setPostValue:artikel.enotaMere forKey:@"enotamere"];
[request setPostValue:artikel.opis forKey:@"opis"];
[request setDelegate:self];
[request startAsynchronous];
```

Najprej preberemo iz nastavitev uporabnika URL naslov strežnika, na katerega pošiljamo podatke, nato pa naredimo zahtevo, ki je naslovljena na ta naslov. Tej zahtevi moramo dodati še podatke o artiklu, katerega podatke pošiljamo na spletni strežnik. Na koncu moramo določiti še sebe kot delegata, nato pa samo še pošljemo zahtevo.

3.7.4 MBProgressHUD

MBProgressHUD je preprosta knjižnica, ki omogoča prikaz prosojnega obvestila (z ali brez besedila), ki uporabnika opozarja, da se neko delo izvaja v ozadju.

3.7.5 CoreDataTableViewController

CoreDataTableViewControler je razred, ki vsebuje predvsem samo kodo, kopirano iz dokumentacije NSFetchedResultsController. V bistvu razširja UITableViewController, tako da se prikazani podatki osvežijo takoj, ko pride do spremembe v podatkovni bazi.

4 Razvoj programa Skener

4.1 Ideja

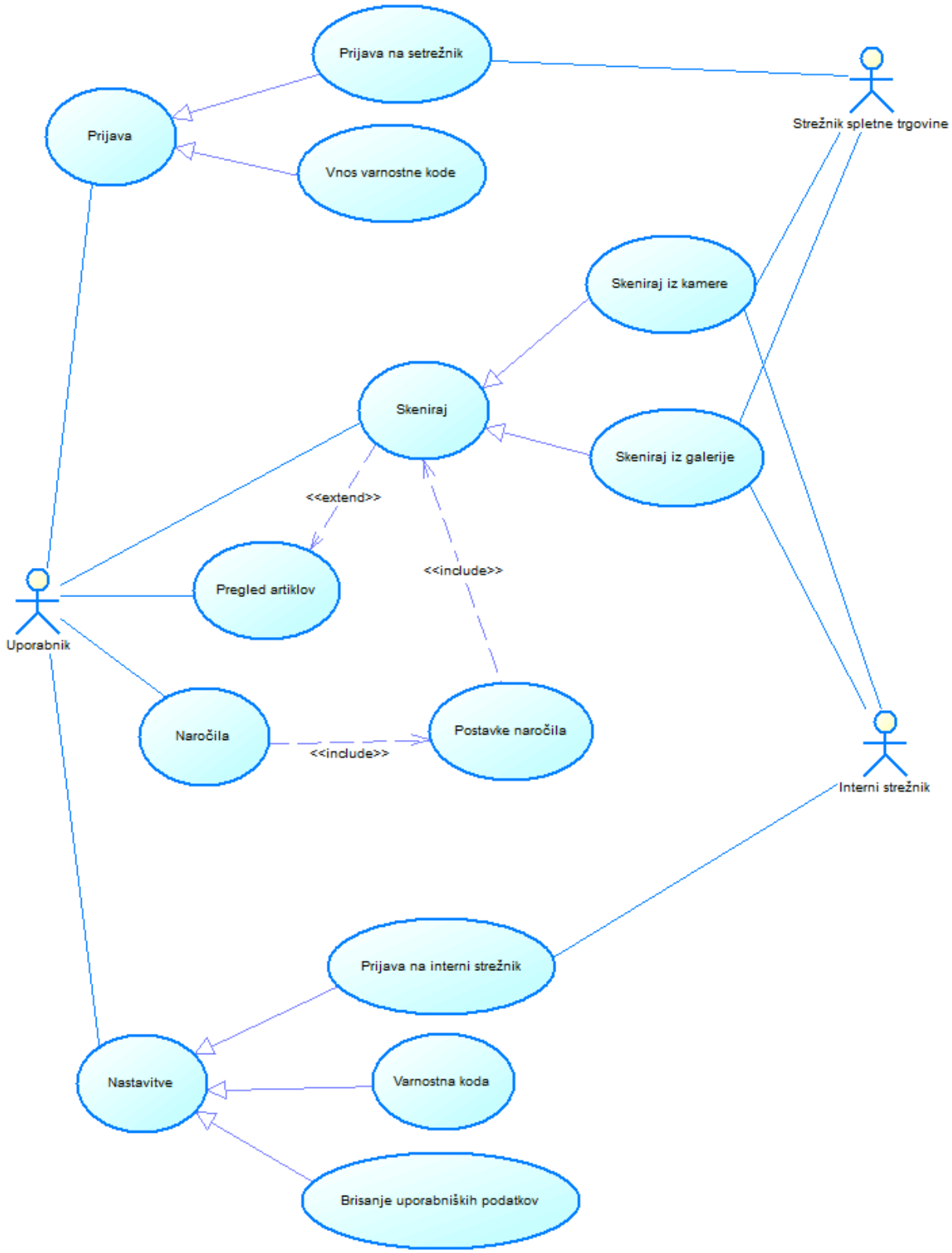
Na tržišču obstaja veliko aplikacij, ki omogočajo naročanje v spletnih trgovinah s pomočjo mobilnih telefonov. Nekatere izmed teh aplikacij omogočajo tudi izbiro artiklov s pomočjo skeniranja črtne kode. Vendar so te aplikacije namenjene predvsem navadnim uporabnikom (končnim potrošnikom) in ne bolj specifičnim, kot so obrtniki, ki bi radi imeli poleg ostalih podatkov tudi kakšen podatek iz svojega informacijskega sistema (npr. zaloga).

Na drugi strani pa obstajajo tudi mobilne aplikacije, ki so posebej narejene za uporabo skupaj z obstoječim informacijskim sistemom. Vendar te aplikacije nimajo direktnega dostopa do spletne trgovine in s tem dostopa do podatkov o vseh artiklih.

Naša ideja je bila, da bi združili obe vrsti aplikacij. Tako imamo na eni strani dostop do spletne trgovine in na drugi strani dostop do internega informacijskega sistema. Zasnovali smo aplikacijo, ki iz spletne trgovine pobira podatke o artiklih ter omogoča tudi pošiljanje naročil direktno v spletno trgovino. Poleg tega pa se aplikacija lahko poveže tudi z internim informacijskim sistemom in iz njega prebere podatek o zalogi. V primeru, da aplikacija zahteva podatke o artiklu, ki ga še ni v internem informacijskem sistemu, pa ga le-ta doda. Tako ni potrebno ročno vnašanje podatkov v računalnik, kar tudi zmanjša možnost za napake. Poleg tega pa smo želeli uporabniku omogočiti uporabo aplikacije tudi takrat, kadar nima internetne povezave. Za to smo uporabili tudi notranjo podatkovno bazo, ki hrani vse podatke o artiklih in o naročilih.

4.2 Načrtovanje

Pri načrtovanju smo si pomagali z UML (Unified Modeling Language) diagramom uporabe, katerega smo naredili s pomočjo orodja PowerDesigner. UML diagram je prikazan na sliki 10. Uporabnik se mora ob prvem zagonu aplikacije prijaviti z uporabniškim imenom in geslom, ki ju uporablja za prijavo v spletno trgovino. Nato je preusmerjen na prvo stran, kjer lahko izbira med skeniranjem iz kamere ter skeniranjem iz galerije slik. Uporabnik lahko kreira tudi naročila, v katera s pomočjo skenerja dodaja nove artikle. Aplikacija omogoča uporabniku tudi pregled artiklov, ki so shranjeni v podatkovni bazi. Pri artiklih je možno tudi dodajanje novih s pomočjo skenerja. V nastavitvah pa je možno vnesti podatke o internem informacijskem sistemu, vnos varnostne kode ter izbris uporabniških podatkov.

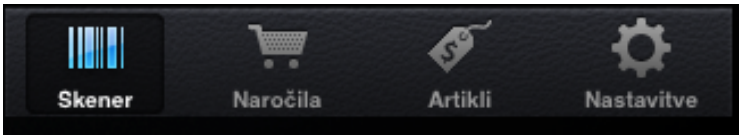


Slika 10: UML diagram uporabe mobilne aplikacije

Na podlagi tega diagrama smo se odločili, da je najboljše aplikacijo razdeliti na štiri logične dele:

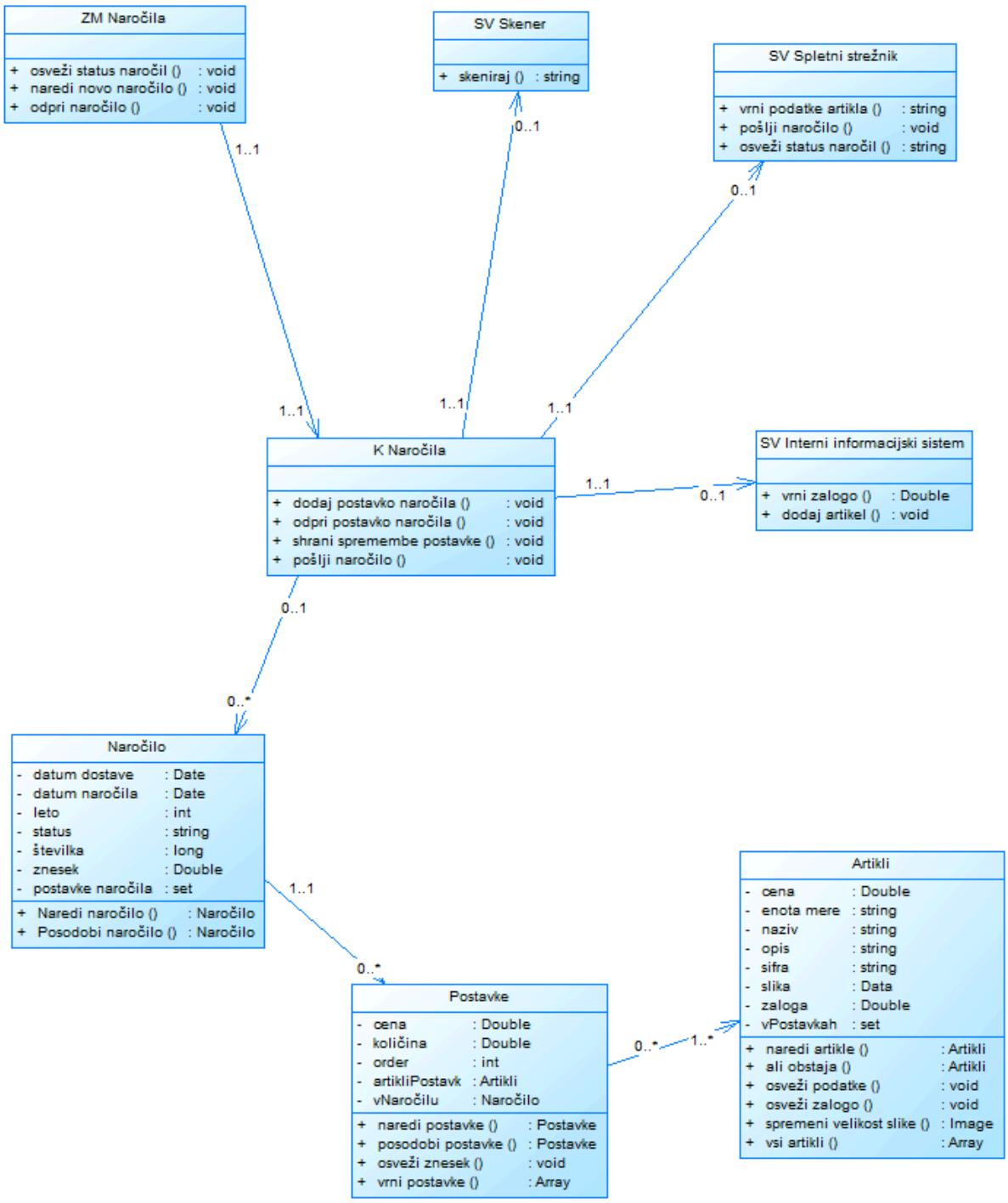
- **Skener** omogoča uporabniku hiter dostop do zagona skenerja in skeniranja črtne kode, kot rezultat pa prikaže podrobnosti skeniranega artikla.
- **Pregled artiklov** uporabniku prikazuje seznam vseh artiklov v podatkovni bazi.
- **Naročila** omogočajo izdelavo in pošiljanje naročil.
- **Nastavitve** pa omogočajo dostop do nastavitev programa.

Med posameznimi deli se lahko premikamo s pomočjo spodnje menijske vrstice (slika 11).



Slika 11: Spodnja menijska vrstica

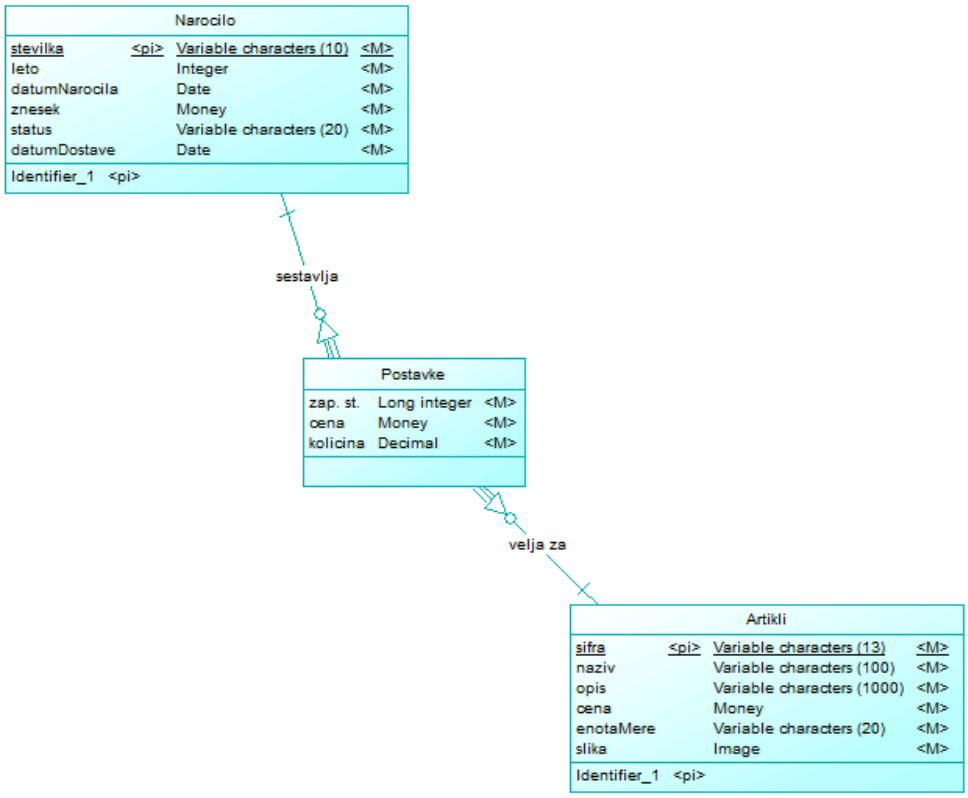
Izkazalo se je, da so nekateri deli aplikacije dokaj enostavni, drugi pa so precej zapleteni in pri delovanju komunicirajo z večjim številom razredov. Tako smo za tiste bolj kompleksne primere naredili tudi razredne diagrame. Eden izmed najbolj zapletenih je primer izdelave naročila (slika 12), ki komunicira tako s celotno podatkovno bazo, kot tudi z obema strežnikoma in tudi s skenerjem.



Slika 12: Primer razrednega diagrama naročila

4.3 Podatkovna baza

Odločili smo se za podatkovno bazo Core Data. Pri načrtovanju in izdelavi podatkovne baze nam je bil zelo v pomoč preprost uporabniški vmesnik za izdelavo podatkovne baze. Sestavili smo jo iz treh entitetnih tipov ter dveh relacij, kot je prikazano na sliki 13.



Slika 13: Prikaz entitet in relacij v podatkovni bazi

Iz podatkovne baze smo s pomočjo programa Xcode generirali podrazrede, s pomočjo katerih lahko dostopamo do podatkov, shranjenih v podatkovni bazi. Te podrazrede pa smo s pomočjo kategorij razširili in jim dodali še nekatere druge metode. Dodane metode so bile namenjene predvsem dodajanju zapisov ter osveževanju podatkov v podatkovni bazi.

Ker do podatkovne baze dostopamo iz več zavihkov, smo uporabili tako imenovani »Singlelton«. To pomeni, da smo uporabili poseben razred, ki skrbi za povezavo s podatkovno bazo. Ta razred ima eno metodo, katero kličemo iz razreda, v katerem želimo imeti dostop do podatkovne baze in nam vrne skupnega upravljavca datotek (ang. document handler).

V vsakem razredu, ki potrebuje dostop do podatkovne baze, imamo narejeno tako, da se ob prikazu okna na ekranu vzpostavi tudi povezava s podatkovno bazo. Pri tem moramo najprej

nastaviti metodo »setupFetchedResultsController«, ki s pomočjo knjižnice »CoreTableViewController« skrbita, da so prikazani osveženi podatki. Podatki se na ekranu osvežijo takoj, ko pride do kakšne spremembe v podatkovni bazi.

4.4 Uporabniški vmesnik

Za izdelavo uporabniškega vmesnika smo uporabili način, ki je bil prvič predstavljen skupaj z operacijskim sistemom iOS 5. Pri tem uporabljamo tako imenovan »Storyboard« na katerega dodamo vse poglede (ang. view), ki jih potrebujemo za izdelavo aplikacije. Poglede lahko dokaj enostavno povežemo enega z drugim, hkrati pa tudi vidimo vse povezave med pogledi. Vse to nam precej olajša razumevanje komunikacije in povezav med pogledi. Povezave, ki so narejene znotraj programske kode (npr. prikaz pogleda za vnos varnostne kode), se tukaj ne vidijo. Za izdelavo naše aplikacije smo uporabili 12 različnih pogledov (slika 14).

Slika 14: Storyboard celotne aplikacije

Pogledi dedujejo lastnosti svojih staršev. Tako je v tem primeru (slika 12) »Tab Bar Controller« oče vseh drugih pogledov, kar pomeni, da imajo vsi pogledi v svoji nogi že vgrajeno vrstico z zavihki. V vseh primerih uporabljamo tudi »Navigation Controller«, ki vsem svojim sinovom doda zgornjo navigacijsko vrstico in omogoča avtomatsko vračanje po hierarhiji navzgor. Pri vsakem pogledu je seveda možno v nastavitvah tudi izklopiti dedovanje.

Ostali elementi, kot so gumbi, napisi, vnosna besedila itd. so ravno tako kot pogledi na voljo v orodni vrstici in jih enostavno dodajamo na želena mesta. Vsak element je možno poljubno oblikovat s pomočjo nastavitev lastnosti, lahko pa tudi programsko tako, da se lahko

spreminjajo med samim izvajanjem programa. Tako smo naredili metodo, ki ob zagonu aplikacije vsem glavnim elementom nastavi obliko. Ker pa smo hoteli videz aplikacije bolj prilagoditi svojim željam, smo s pomočjo programa Adobe Photoshop izdelali posebne oblike za glavne elemente.

Metoda, ki ob zagonu aplikacije nastavi obliko glavnim elementom:

```
- (void)customizeiPhoneTheme{

   //navigation bar background
   UIImage *navBarImage = [UIImage imageNamed:@"nav-bar.png"];
   [[UINavigationBar appearance] setBackgroundImage:navBarImage
forBarMetrics:UIBarMetricsDefault];

   //navigation bar button
   UIImage *barButton = [[UIImage imageNamed:@"bar-button.png"]
resizableImageWithCapInsets:UIEdgeInsetsMake(5, 7, 5, 7)];
   [[UIBarButtonItem appearance] setBackgroundImage:barButton
forState:UIControlStateNormal barMetrics:UIBarMetricsDefault];

   //navigation bar back button
   UIImage *backButton = [[UIImage imageNamed:@"back.png"]
resizableImageWithCapInsets:UIEdgeInsetsMake(0, 15, 0, 6)];
   [[UIBarButtonItem appearance] setBackButtonBackgroundImage:backButton
forState:UIControlStateNormal barMetrics:UIBarMetricsDefault];

   //tab bar background
   UIImage *tabBarImage = [UIImage imageNamed:@"tab-bar.png"];
   [[UITabBar appearance] setBackgroundImage:tabBarImage];
   [[UITabBar appearance] setSelectionIndicatorImage:[UIImage imageNamed:@"tab-bar-
selection.png"]];

   //nastavi status bar style
   [[UIApplication sharedApplication] setStatusBarStyle:UIStatusBarStyleBlackTranslucent
animated:YES];
}
```

Metoda deluje tako, da glavnim elementom nastavi sliko za ozadje. Uporabljamo dva načina nastavljanja slike. Pri prvem načinu je slika enako velika kot prikazan element, tako da sliko samo nastavimo za ozadje. Tak primer je navigacijska vrstica. Za drugi način, ki pa ga uporabljamo pri elementih, ki lahko spreminjajo svojo velikost, pa uporabimo ponavljanje slike. Tako nastavimo kolikšen del slike je stacionaren (običajno začetek in konec) vmesni del pa se avtomatsko prilagaja glede na velikost elementa. Primer takega elementa so gumbi v navigacijski vrstici. Na koncu pa še spremenimo stil statusne vrstice, ki je po privzetih nastavitvah siva, vendar se ne ujema dobro z našo barvno shemo, zato smo uporabili transparentno črno barvo.

Ker je aplikacija mišljena za uporabo v skladišču, smo želeli, da bi z oblikovanjem aplikacije dosegli videz robustne, hkrati pa tudi udobne aplikacije.

Tako smo za zgornjo in spodnjo vrstico uporabili videz usnja. Usnje pride do izraza predvsem v zgornji vrstici, saj jo od ostalega dela ekrana loči bela črta, ki daje videz, kot da je zgornja vrstica prišita in zato negibna (slika 15).

Slika 15: Usnjena zgornja vrstica

Za prikaz podrobnosti pa smo se odločili, da uporabimo videz, ki spominja na kovino, kar predstavlja trpežnost in vzdržljivost (slika 16).

Slika 16: Kovinski videz vrstice v tabeli

Pri oblikovanju smo morali upoštevati tudi to, da imajo starejši modeli telefona manjšo ločljivost kot novejši. Zaradi tega smo najprej vse elemente naredili za iPhone z visoko ločljivostjo, nato pa vse elemente zmanjšali za 50 % in tako dobili tudi elemente za iPhone z manjšo ločljivostjo. Vse elemente je potrebno uvoziti v Xcode. Elementi, ki so narejeni za iPhone z visoko (retina) ločljivostjo, se od ostalih ločijo po zadnjem delu imena, kjer morajo imeti posebno oznako »@2x«, ki nakazuje, da so dvakrat večji od navadnih elementov. V programski kodi uporabljamo imena navadnih (manjših) elementov, nato pa se ob zagonu programa avtomatsko izbere, katere elemente bo program uporabil.

Poleg samega uporabniškega vmesnika smo naredili tudi ikono programa. Pri tem smo upoštevali, da mora biti čim bolj enostavna, z malo podrobnostmi, hkrati pa mora čim bolj predstavljati, kaj aplikacija dela oziroma za kaj se uporablja. Mora pa se tudi čim bolj razlikovati od ikon ostalih programov, saj tako uporabnik lažje in hitreje najde ikono na ekranu.

Slika 17: Ikona programa

Na podlagi tega smo naredili dokaj preprosto ikono, ki je razdeljena na dva dela. Zgoraj je narisano skladišče, kar ponazarja, da je aplikacija namenjena za uporabo v skladišču. Na spodnjem delu pa je narisana črtna koda, katero ravno skenira laserski skener, kar ponazarja, da aplikacija lahko skenira črtne kode. Za ozadje smo izbrali modro barvo, kar naredi dober kontrast med ozadjem in ospredjem, hkrati pa se modra barva tudi dobro povezuje z barvo pisave v aplikaciji, ki je na nekaterih mesti ravno tako modre barve (slika 17).

4.5 Zgradba programa

Kot je bilo že zapisano v poglavju 4.2, smo program razdelili na štiri dele. Tako si bomo sedaj malo podrobneje ogledali posamezne dele in funkcionalnosti.

4.5.1 Skener

To je prvo okno, katerega vidi uporabnik ob zagonu aplikacije (slika 18). Omogoča mu takojšen zagon skenerja. Na voljo pa ima dve možnosti, saj lahko zažene skener iz kamere, lahko pa tudi skenira slike iz galerije slik.

Slika 18: Osnovno okno za uporabo skenerja

Metoda za zagon skenerja:

```
- (IBAction)zazeniSkener:(UIButton *)sender {
    //zagon skenerja
    ZBarReaderViewController *reader = [ZBarReaderViewController new];
    reader.readerDelegate = self;
    reader.supportedOrientationsMask = ZBarOrientationMaskAll;
    reader.tracksSymbols = YES;

    ZBarImageScanner *scanner = reader.scanner;
    [scanner setSymbology:ZBAR_I25 config:ZBAR_CFG_ENABLE to:0];
    [self presentModalViewController:reader animated:YES];
}
```

Ob pritisku na gumb se kliče metodo za prikaz skenerja. Skenerju je potrebno nastaviti nekaj osnovnih nastavitev, kot so:

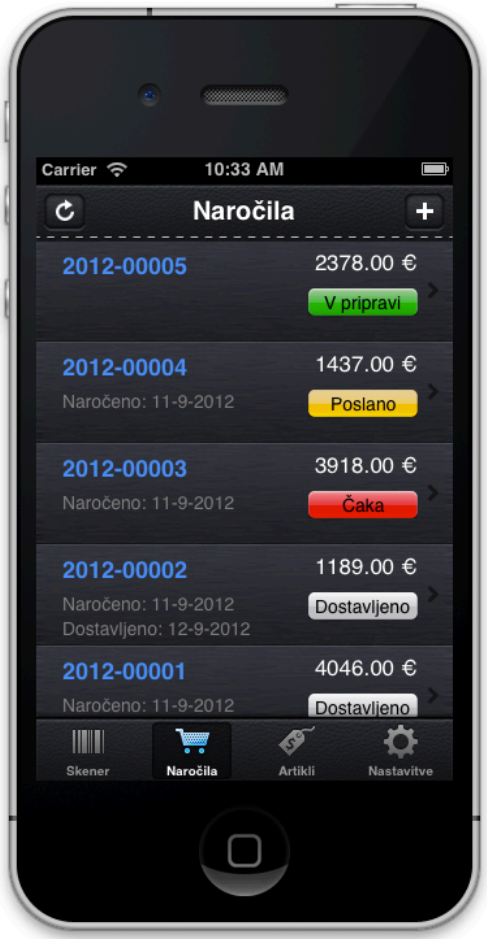
- sledenje simbolom (okrog črtne kode naredi kvadrat);

- katere orientacijske maske podpira (ali se prikaz na zaslonu prilagaja nagibu naprave);
- določiti je potrebno tudi katere vrste črtnih kod prepozna.

Ko skener prepozna črtno kodo, jo analizira in vrne NSString s šifro črtne kode. Program najprej preveri, če so podatki o artiklu že v podatkovni bazi, če se podatki o tem artiklu že nahajajo v podatkovni bazi in če jih je našel, jih tudi prikaže, drugače pa pošlje zahtevo na spletni strežnik. Če so vneseni podatki tudi za interni informacijski sistem in če ima naprava dostop do omrežja, potem ob vsakem skeniranju iz internega informacijskega sistema prenese tudi podatek o zalogi. Nato pa vse podatke shrani v lokalno bazo. Če je to prvo skeniranje artikla in še ni shranjen v internem informacijskem sistemu, potem se avtomatsko dodajo podatki o artiklu tudi v interni informacijski sistem. Če slučajno ni povezave s spletnim strežnikom, pa se prikaže obvestilo, ki uporabnika obvesti o težavi.

4.5.2 Naročila

Za prikaz naročil uporabljamo tabelo (slika 19), tako se ob pritisku na eno izmed vrstic prikažejo podrobnosti (postavke) naročila. V tabeli vidimo za vsako naročilo številko naročila, znesek in pa status. Če je naročilo že poslano, vidimo tudi datum naročila, ko pa je naročilo dostavljeno, pa vidimo tudi datum dostave.



Slika 19: Prikaz vseh naročil

Naročila je možno dodajati s pritiskom na ikono +. Pri tem se ustvari novo naročilo, hkrati pa se odpre novo okno s prazno tabelo. Tukaj pa s pritiskom na ikono + odpremo skener, s pomočjo katerega dodamo novo postavko. Po končanem skeniranju se prikaže novo okno s podatki o izbranem artiklu in poljem, v katerega vpišemo želeno količino. Količine ni potrebno vpisovati, saj lahko samo pritisnemo na gumb + ali –, odvisno od tega, ali želimo povečati ali zmanjšati količino. Hkrati s tem, ko spreminjamo količino, pa se tudi izračunava trenuten znesek. Ko je količina vnesena, pritisnemo na zgornji desni gumb Save in tako shranimo podatke in zapremo trenutno okno ter se vrnemo na postavke naročila.

Naročilo lahko vsebuje poljubno število artiklov (postavk), ne more pa vsebovati več enakih artiklov. V primeru, ko ponovno skeniramo enako črtno kodo, nam program prikaže podatke artikla, kateremu smo že določili količino. Na tak način preprečimo, da bi uporabnik ponesreči naročil en artikel večkrat.

Ko je naročilo pripravljeno za pošiljanje, potegnemo tabelo nekoliko navzdol, tako da se nad prvim artiklom prikaže zelen gumb »Pošlji naročilo«. S pritiskom na ta gumb se celotno

naročilo pošlje na spletni strežnik, kjer se nato obdela. Če je naročilo uspešno poslano, se zeleni gumb obarva rjavo, napis pa se spremeni na »Naročilo poslano«, hkrati pa se spremeni tudi status naročila v »Poslano«, doda pa se tudi datum, kdaj je bilo naročilo poslano.

Ko je naročilo enkrat poslano, se pri tem naročilu onemogoči gumb za dodajanje artiklov, prav tako pa se onemogoči spreminjanje naročene količine.

Ob pritisku na ikono se sproži posodobitev statusa ter datuma dostave naročila. Če je status »Čaka«, pomeni, da je naročilo pri dobavitelju, vendar je na čakanju (npr. dobavitelj artikla nima na zalogi). Status »Dostavljeno« pa pomeni, da je bilo naročeno blago že poslano, hkrati pa piše tudi datum dostave.

4.5.3 Artikli

Tukaj so prikazani vsi artikli, ki so shranjeni v podatkovni bazi. To so vsi artikli, ki so bili do sedaj skenirani na tej napravi. V tabeli vidimo sliko artikla, naziv, ceno ter zalogo (slika 20). Ob pritisku na enega izmed njih se prikaže pogled, ki prikazuje podrobnejše podatke o izbranem artiklu.

Artikle dodajamo s pritiskom na ikono . Postopek je podoben kot pri naročilih, s to razliko, da tukaj ne moremo vnašati količine.

Ob pritisku na ikono pa se sproži posodobitev podatkov pri vseh artiklih v podatkovni bazi. Zaradi potrebe po prenosu večje količine podatkov lahko osveževanje traja nekoliko dalj časa, zato uporabniku prikažemo animacijo, ki nakazuje delovanje v ozadju.

Slika 20: Prikaz vseh artiklov v podatkovni bazi

Slike pri artiklih imajo lahko različne ločljivosti. Tako so lahko nekatere slike s precej veliko ločljivostjo, kar pa se pozna pri samem delovanju aplikacije. Zaradi tega lahko pri premikanju po tabeli pride do učinka zatikanja in počasne odzivnosti. To se pojavi predvsem pri starejših napravah, ki imajo počasnejše procesorje in manj pomnilnika. Zaradi tega smo se odločili, da bomo sliko prikazovali v drugi niti. To pomeni, da premikanje po tabeli poteka tekoče, brez zatikanja, slika pa se prikaže (osveži) z manjšim zamikom.

Del programske kode, ki za obdelavo slike uporablja drugo nit:

```
dispatch_queue_t resizeQ = dispatch_queue_create("resize image", NULL);
dispatch_async(resizeQ, ^{
   UIImage *image = [Artikli postProcessImage:[UIImage imageWithData:artikli.slika]];
   [self.narocilaDatabase.managedObjectContext performBlock: ^{
      cell.thubnailImageView.image = image;
   }];
});
```

Najprej stopimo v čakalno vrsto, v kateri kličemo metodo za zmanjšanje slike, nato pa moramo stopiti nazaj v glavno nit ter tukaj prikazati sliko. Na tak način smo v drugi niti sprožili obdelavo slike in s tem razbremenili glavno nit, kar se pozna predvsem pri bolj tekočem uporabljanju programa.

4.5.4 Nastavitve

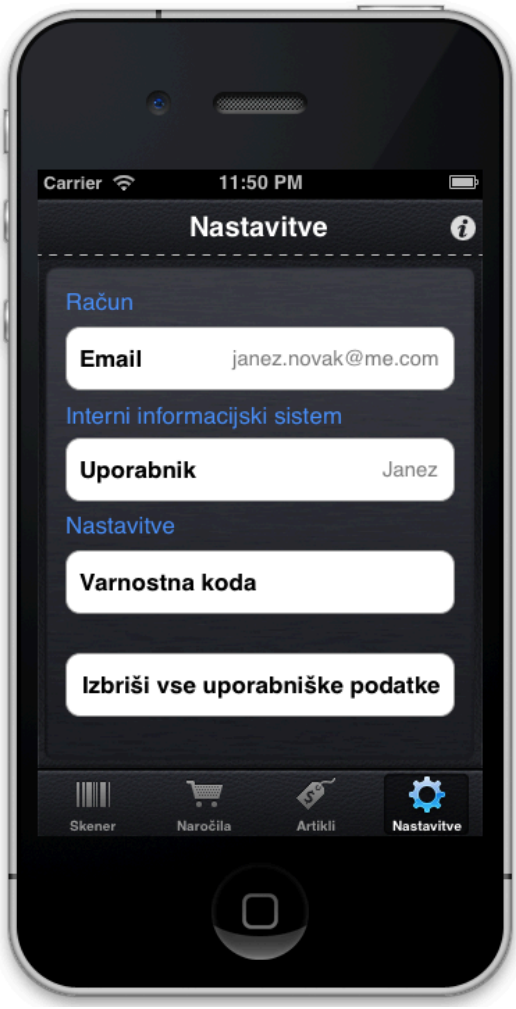
V nastavitvah se uporabnik lahko prijavi tudi na interni informacijski sistem. Za prijavo mora vpisati spletni naslov ter svoje uporabniško ime in geslo. Na voljo pa imamo tudi nastavitev varnostne kode ter brisanje uporabniških podatkov (slika 21).

Za shranjevanje nastavitev uporabljamo NSUserDefaults, ker je potrebno shraniti relativno malo podatkov.

Del programske kode, ki shrani uporabnikove podatke za dostop do internega informacijskega sistema:

```
NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
[defaults setObject:emailTextField.text forKey:@"uporabnik"];
[defaults setObject:gesloTextField.text forKey:@"geslo2"];
[defaults synchronize];
```

Najprej vzpostavimo povezavo z datoteko, v kateri so shranjene nastavitve. Nato dodamo podatke, katere želimo shraniti, pri tem pa moramo paziti, da uporabimo pravilne ključe, ker drugače ne bomo mogli prebrati teh nastavitev. Na koncu še shranimo.



Slika 21: Prikaz nastavitev

Prijava v interni informacijski sistem ni obvezna, vendar v primeru, če uporabnik ni prijavljen, aplikacija ne more dostopati do podatkov o zalogi, lahko pa uporabnik uporablja vse ostale funkcionalnosti programa.

Varnostna koda ni obvezna, je pa priporočljiva, saj preprečuje dostop do programa nepooblaščenim osebam. Varnostna koda je dokaj preprosta, saj je dolžine štiri znakov, znaki pa so samo številke. Varnostno kodo je mogoče vklopiti ali pa izklopiti, mogoče pa jo je tudi zamenjati (slika 22). Na voljo je tudi opcija brisanja podatkov ob neuspeli prijavi. To pomeni, da če nekdo več kot desetkrat zaporedoma vpiše napačno kodo, se vsi uporabniški podatki izbrišejo. Po izbrisu podatkov mora uporabnik vpisati svoje uporabniško ime ter geslo, ki ju uporablja za prijavo v spletno trgovino.

Slika 22: Možnosti varnostne kode

Če želi uporabnik izbrisati svoje podatke, se zaradi varnosti pojavi okno za prijavo in vpisati mora svoje uporabniško ime in geslo.

4.6 Spletna strežnika

Aplikacija za pravilno delovanje potrebuje nek vir podatkov ter neko mesto za shranjevanje naročil. Zato smo naredili tudi preprost spletni strežnik, ki hrani podatke o artiklih in uporabnikih. Omogoča pa tudi shranjevanje naročil. Potrebujemo pa tudi še drug strežnik, ki simulira uporabo internega informacijskega sistema. Ta drugi strežnik omogoča shranjevanje artiklov, vračanje zaloge ter prijavo v sistem.

Razviti strežniški aplikaciji služita kot dokaz delovanja aplikacije in nista bili bistvo tega diplomskega dela.

Za delovanje obeh spletnih strežnikov smo uporabili Apache spletni strežnik, za delovanje podatkovnih baz MySQL, za izdelavo pa programski jezik PHP.

Del kode za povezavo s podatkovno bazo:

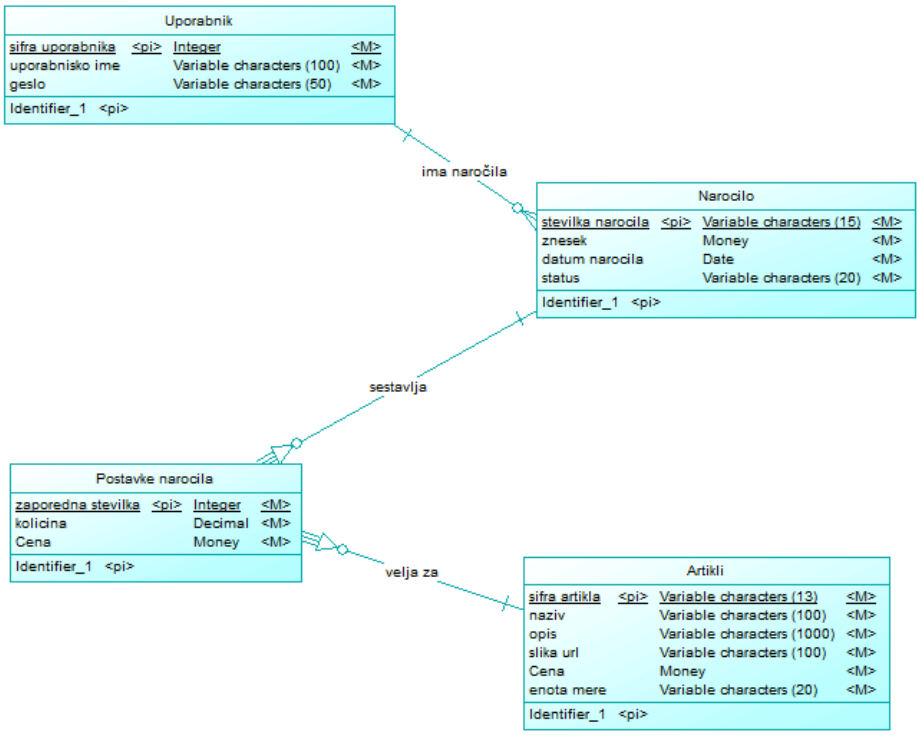
```
$localhost="127.0.0.1";
$username="root";
$password="root";
$database="localServer";

$con = mysql_connect($localhost, $username, $password);
if (!$con) {
        die('Could not connect: ' . mysql_error());
}
$db_selected = mysql_select_db($database,$con);
```

4.6.1 Spletni strežnik 1

Ta spletni strežnik simulira delovanje spletne trgovine in hrani podatke o artiklih, uporabnikih ter naročilih. Tako ima implementirane naslednje funkcionalnosti:

- preverjanje, ali so uporabnikovi podatki za prijavo pravilni. Če so, vrne šifro uporabnika;
- ob prejetju šifre poišče, če artikel s to šifro obstaja. Če obstaja, vrne vse podatke tega artikla (naziv, opis, cena, enoto mere ter sliko):
- ob prejetju zahteve za osvežitev podatkov artiklov vrne tabelo vseh podatkov o artiklih;
- sprejem in obdelavo naročil, kar pomeni, da v tabelo Naročila shrani podatke o naročilu, v tabelo Postavke naročila pa shrani vse postavke tega naročila;
- osveževanje statusa naročila vrne tabelo s statusi vseh naročil, v primeru, ko je status »Dostavljeno«, pa vrne tudi datum dostave.

Osnovna zgradba podatkovne baze na spletnem strežniku je prikazana na sliki 23.



Slika 23: Zgradba podatkovne baze na strežniku

4.6.2 Spletni strežnik 2

Ta spletni strežnik simulira interni informacijski sistem in hrani podatke o artiklih in uporabnikih, ki imajo dovoljenje za dostop. Implementirane ima tri funkcionalnosti:

- preverjanje uporabnikovih podatkov za dostop;
- dodajanje podatkov o artiklu v tabelo Artikli;
- vračanje zaloge za iskan artikel.

Del kode, ki v tabelo Artikli vnese podatke novega artikla:

```
if (isset($_POST["vnesi"]))
{                       // v tabelo artikli vnese podatke novega artikla
        mysql_query("INSERT INTO artikli (sifra, naziv, opis, cena, enota_mere, zaloga)
                        VALUES          // podatke prebere iz glave zahteve
                        ('$_POST[sifra]', '$_POST[naziv]', '$_POST[opis]', '$_POST[cena]',
'$_POST[enotamere]', '0')");
        mysql_close($con);      // zapre povezavo s podatkovno bazo
        sendResponse(201);      // vrne statusno kodo, ki označuje uspešno dodan zapis
        return true;
}
```

5 Sklepne ugotovitve

Končni rezultat diplomskega dela je delujoč prototip informacijskega sistema, ki obsega dva spletna strežnika ter mobilno aplikacijo, ki omogoča izdelavo naročil s pomočjo skenerja črtnih kod.

Moj cilj je bil izdelati aplikacijo, ki bi omogočala čim lažjo izdelavo naročil, pri tem pa bi za vir podatkov uporabljala dva spletna strežnika. Z aplikacijo bi lahko naročila poslali direktno v spletno trgovino, pri tem pa bi za izbiro artiklov uporabili skener črtnih kod.

Na začetku sem imel veliko težav, saj je bilo to moje prvo srečanje s programom Xcode ter s programskim jezikom Objective-C. Na srečo na spletu obstaja veliko gradiva, predvsem pa mi je bil v pomoč iTunesU, kjer so na voljo videoposnetki celotnega semestra predmeta CS193p iz univerze Stanford, kjer poučujejo izdelavo aplikacij za iOS operacijski sistem.

S programom Xcode in programskim jezikom Objective-C sem se naučil precej novega, saj sem spoznal precej novih gradnikov in načinov uporabe ter na koncu dosegel zastavljeni cilj.

Literatura

[1] (2012) iPhone. Dostopno na:
http://en.wikipedia.org/wiki/IPhone

[2] (2012) iOS. Dostopno na:
http://en.wikipedia.org/wiki/Ios

[3] (2012) Apache HTTP strežnik. Dostopno na;
http://en.wikipedia.org/wiki/Apache_HTTP_Server

[4] (2012) PHP. Dostopno na:
http://en.wikipedia.org/wiki/Php

[5] (2012) MySQL. Dostopno na:
http://en.wikipedia.org/wiki/Mysql

[6] (2012) Xcode. Dostopno na:
https://developer.apple.com/technologies/tools/whats-new.html

[7] (2012) Objective C. Dostopno na:
http://en.wikipedia.org/wiki/Objective_c

[8] (2012) Objective C. Dostopno na:
http://mobile.tutsplus.com/tutorials/iphone/learn-objective-c-day-1/

[9] (2012) Cocoa. Dostopno na:
https://developer.apple.com/technologies/mac/cocoa.html

[10] (2012) Data Management. Dostopno na:
https://developer.apple.com/technologies/mac/data-management.html

[11] (2012) ZBarSDK. Dostopno na:
http://zbar.sourceforge.net/iphone/sdkdoc/

[12] (2012) SBJson. Dostopno na:
http://stig.github.com/json-framework/

[13] (2012) MBProgressHUD. Dostopno na:
https://github.com/jdg/MBProgressHUD

[14] (2012) CoreDataTableViewController. Dostopno na:
http://www.stanford.edu/class/cs193p/cgi-bin/drupal/downloads-2011-fall

[15] Apple Inc. The Objective-C Programming Language, 2010

[16] Apple Inc. iOS Technology Overview, 2010