UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Razvoj mobilne aplikacije za evidentiranje prometnih nesreč v Republiki Sloveniji

(Developing mobile application for recording traffic accidents in the Republic of Slovenia)

Ime in priimek: Alen Redek
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Jernej Vičič

Koper, september 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Ključna dokumentacijska informacija

Ime in PRIIMEK: Alen REDEK

Naslov zaključne naloge: Razvoj mobilne aplikacije za evidentiranje prometnih nesreč v Republiki Sloveniji

Kraj: Koper

Leto: 2014

Število listov: 51 Število slik: 7 Število tabel: 3

Število prilog: 3 Število strani prilog: 9 Število referenc: 16

Mentor: doc. dr. Jernej Vičič

Ključne besede: prometne, nesreče, MVC, mobilna, aplikacija

Izvleček:
V zaključni nalogi bo predstavljen razvoj mobilne aplikacije za vnašanje ter evidentiranje prometnih nesreč v Republiki Sloveniji neposredno v podatkovno bazo, ki je bila plod lanske zaključne naloge Andreja Godca z naslovom "Izdelava aplikacije za prikazovanje nevarnih cestnih odsekov v Republiki Sloveniji". Glavni cilj aplikacije bi bil predvsem v geografskem smislu točnejši ter nedvoumen opis prometne nesreče. Ob izdelavi spletne aplikacije za prikaz nevarnih cestnih odsekov smo namreč naleteli na veliko število nepopolnih podatkov, ki so onemogočali učinkovito geolociranje prizorišč prometnih nesreč. Za rešitev omenjenega problema bo odgovorna glavna funkcionalnost mobilne aplikacije, ki bo s pomočjo trenutne lokacije naprave enolično in natančno določila kraj prometne nezgode. Na ta način bo pristojnim organom prihranjenega veliko časa ter truda za geografsko umestitev prometne nesreče, saj je le-ta trenutno določena s pomočjo petih opisnih atributov v podatkovni bazi. Prizorišče prometne nesreče bo tako na razmeroma enostaven način določeno z veliko natančnostjo v upanju na več uporabnih podatkovnih vpisov, ki bodo v realnem času tudi umeščeni na zemljevidu ter pripomogli h zmanjšanju števila prometnih nesreč na naših cestah.

Key words documentation

Name and SURNAME: Alen REDEK

Title of final project paper: Developing mobile application for recording traffic accidents in the Republic of Slovenia

Place: Koper

Year: 2014

Number of pages: 51 Number of figures: 7 Number of tables: 3

Number of appendices: 3 Number of appendix pages: 9 Number of references: 16

Mentor: Assist. Prof. Jernej Vičič, PhD

Keywords: traffic, accidents, MVC, mobile, application

Abstract:
The final thesis will present the development of a mobile application for recording traffic accidents in the Republic of Slovenia directly into the database that was created last years by Andrej Godec for his thesis entitled: "Developing application for displaying dangerous road sections in the Republic of Slovenia". The main goal of the application would be to accurately pinpoint the location of a traffic accident and to provide understandable and unambiguous data about it. In developing the application, we encountered a large number of incomplete and incorrect data, which rendered the accurate localisation of the accident more difficult. The main function of the application will be to solve the aforementioned issue. It will determine the exact location of a traffic accident using the device's coordinates. Using this application, the competent authorities will save time and effort previously wasted on ascertaining the exact location of an event. The software will perform this part of the task instead of using the five attributes describing the geographic location of a traffic accident from the database. The localisation of an accident will be simplified as well as more accurate. A site of a traffic accident will thus be more easily and accurately determined. Hopefully, real time map integration will contribute to lowering the number of traffic accidents on our roads.

Zahvala

Rad bi se zahvalil mentorju zaključne naloge doc. dr. Jerneju Vičiču ter ostalim pedagoškim delavcem na UP FAMNIT, ki so mi omogočili kvalitetno izobraževanje.

Posebne zahvale si zasluži tudi dr. Branko Bogunovič za ves trud in tehnično podporo, ki mi jo je nudil med pisanjem zaključne naloge.

Na koncu bi se rad zahvalil še mojim staršem, teti in partnerki, ker so od samega začetka verjeli vame, ter za vso njihovo potrpežljivost v času mojega študija.

Hvala vam.

Kazalo vsebine

Seznam tabel

Seznam slik

Seznam prilog

Seznam kratic

BSD	Berkeley Software Distribution (Distribucija programske opreme Berkeley)
CSS	Cascading Style Sheets (Kaskadne stilske podloge)
ExtJS	Razvojno okolje JavaScript za izdelavo interaktivnih spletnih aplikacij
HMVC	Hierarchical MVC (Hierarhični MVC)
HTML	Hyper Text Markup Language (Jezik za označevanje nadbesedila)
iOS	Applov operacijski sistem za mobilne naprave
JavaScript	Skriptni jezik
Mac OSX	Applov operacijski sistem za računalnike Macintosh
MVA	Model-View-Adapter (Model-Pogled-Vmesnik)
MVC	Model-View-Controller (Model-Pogled-Upravitelj)
MVP	Model-View-Presenter (Model-Pogled-Prikazovalec)
MVVM	Model-View-View-Model (Model-Pogled-Pogled-Model)
OL	OpenLayers (Odprtokodna knjižnica zemljevidov)
OSM	OpenStreetMap (Odprtokodni ulični zemljevid)
Python	Programski jezik
SQL	Structured Query Language (Strukturirani povpraševalni jezik)
UML	Unified Modeling Language (Poenoteni jezik modeliranja)
UNIX	Večopravilni in večuporabniški računalniški operacijski sistem

1 UVOD

Danes si življenja brez tehnoloških naprav pravzaprav sploh ne moremo več predstavljati. V bistvu so tako globoko zakoreninjene v naši podzavesti, da so postale del našega vsakdana pri opravljanju raznoraznih opravil. Mednje prav gotovo sodijo tudi vse mobilne naprave skupaj s tabličnimi računalniki, ki so s svojimi aplikacijami praktično postali nepogrešljiv pripomoček številnih potrošnikov. Na tej točki bi radi izpostavili pomembno vprašanje, ki se poraja v mislih marsikateremu uporabniku. Ali bi lahko vso to tehnologijo uporabili tudi za reševanje človeških življenj? In zakaj se ne bi lotili reševanja življenj že s preprečitvijo oziroma z zmanjšanjem števila prometnih nesreč na naših cestah?

1.1 Motivacija

Pri izdelavi spletne aplikacije za prikazovanje nevarnih cestnih odsekov, ki je bila plod lanske diplomske naloge Andreja Godca, smo izhajali iz podatkov slovenske policije o prometnih nesrečah. Upoštevane so bile nesreče med leti 2007 ter 2011, kjer je bilo zapisov v podatkovni bazi več kot sto osemdeset tisoč, vendar jih je po filtraciji ostalo le še dobrih dvajset tisoč [5]. Kljub temu da so bile pretežno izločene prometne nesreče, ki niso terjale hujših posledic oziroma so se zgodile na manj prometnih cestah, je ta razlika še vedno prevelika. Motivacijo za izdelavo zaključne naloge smo našli ravno v veliki razliki med temi številkami, saj je bilo treba kljub vsemu izločiti veliko podatkovnih zapisov zaradi nepopolnosti ter dvoumnosti. Posledično je to pomenilo, da je bila aplikacija za prikazovanje nevarnih cestnih odsekov zgrajena na relativno skromnem številu prometnih nesreč. Da bi v prihodnje pri klasifikaciji cestnih odsekov lahko upoštevali večino, če ne kar vseh prometnih nesreč, želimo v ta namen pristojnim organom ponuditi v uporabo mobilno aplikacijo, s katero bodo lahko ključne podatke o prometni nesreči na razmeroma enostaven način vnesli neposredno v podatkovno bazo, ki trenutno služi prikazovanju nevarnih cestnih odsekov. Z redno in pravilno uporabo aplikacije bi sčasoma izpopolnili obstoječo klasifikacijo kritičnih predelov.

1.2 Cilji

Smernice zaključne naloge predstavljajo tako izdelavo mobilne aplikacije za evidentiranje prometnih nesreč, kot tudi konsistentno dokumentacijo, ki bo opisovala razvoj, uporabo ter morebitno nadgradnjo aplikacije. Osnova za izdelavo aplikacije je poznavanje osnovnih potreb končnih uporabnikov, v našem primeru bodo to pristojni organi. Grafični vmesnik same aplikacije bo razvit na podlagi podatkovne baze, ki jo trenutno uporabljajo za vnašanje podatkov o prometnih nesrečah. Velik poudarek bo tudi na sami lokaciji nezgode, saj želimo prometno nesrečo geolocirati z največjo možno natančnostjo. S tem bi pridobili uporabne ter natančne podatke, s katerimi bo možno nevarne cestne odseke še natančneje prikazati na zemljevidu.

1.3 Struktura zaključne naloge

V poglavjih, ki si sledijo, bomo zastavljeno problematiko poskušali rešiti z razvojem mobilne aplikacije. Začeli bomo s študijo izvedljivosti optimalne rešitve danega problema. Zatem se bomo z analizo ter definicijo potreb posvetili teoretičnemu delu razvoja, ki bo ključen za končni uspeh. Teoriji bo sledil še praktičen del, kjer bomo poskušali preko načrtovanja in z uporabo preverjenih metod implementirati aplikacijo. Zaključili bomo s testiranjem, kjer bomo odkrivali morebitne napake v delovanju ter skušali analizirati odpravo le-teh. V sklepni besedi bomo še podali svoje stališče o uporabi ter o potencialnih posledicah uporabe aplikacije.

2 DEFINICIJA PROBLEMA

2.1 Opis problema

Pri razvoju spletne aplikacije za prikazovanje nevarnih cestnih odsekov smo naleteli na številne težave. Med drugim smo se soočali z vrsto nepopolnih ter neuporabnih podatkov. Da smo dokončali razvoj aplikacije, smo bili primorani izločiti večino podatkovnih vpisov o prometnih nesrečah. Obstoječo aplikacijo za prikazovanje nevarnih cestnih odsekov [5] želimo izboljšati do te mere, da bi imeli natančnejši, predvsem pa verodostojnejši vpogled na kritične cestne odseke. V ta namen želimo v bodoče pri klasifikaciji cestnih odsekov po nevarnosti uporabiti vsaj večino prometnih nesreč. S tem bi pristojnim organom prihranili čas ter trud, ki ga sicer vsakodnevno vlagajo v podobne dejavnosti, vsi ostali udeleženci v prometu pa bi imeli jasnejši vpogled v najnevarnejše cestne predele na slovenskih cestah.

2.2 Obstoječe rešitve

Po temeljitem pregledu vseh morebitnih alternativ ter rešitev smo prišli do spoznanja, da podobne aplikacije, s pomočjo katere bi prometne nesreče vnašali neposredno v podatkovno bazo, ni. Pristojni organi oziroma policijski uslužbenci, ki sodelujejo na kraju nesreče, še vedno uporabljajo list papirja ter svinčnik, s katerim opišejo tehnične lastnosti prometne nesreče. Postopoma vsako nesrečo vpišejo v svojo podatkovno bazo ter podatke enkrat letno objavijo na svoji spletni strani [14]. Iz teh podatkov smo izhajali tudi pri izdelavi spletne aplikacije za prikazovanje nevarnih cestnih predelov.

3 ŠTUDIJA IZVEDLJIVOSTI

3.1 Koristi projekta

Projekt vključuje razvoj mobilne aplikacije za evidentiranje prometnih nesreč, ki bo namenjena zelo majhnemu številu končnih uporabnikov. Tukaj gre predvsem za pristojne organe, ki so prisotni na prizoriščih prometnih nesreč. Uporaba aplikacije bi jim prihranila ogromno časa ter truda, ki ga trenutno porabijo za evidentiranje prometnih nesreč. Vendar bi kljub temu imeli posredno korist tudi vsi ostali uporabniki, predvsem vozniki na slovenskih cestah ter avtocestah, saj bi imeli s pravilno ter redno uporabo aplikacije jasnejši vpogled v stanje o nevarnosti naših cest. Obstoječa spletna aplikacija bi se namreč vsakodnevno ažurirala z novimi prometnimi nesrečami. Na ta način bi sčasoma upoštevala dovolj veliko število prometnih nesreč, s čimer bi dopolnili klasifikacijo nevarnih cestnih odsekov v upanju, da bi to pripomoglo h zmanjšanju števila prometnih nesreč.

3.2 Izvedljivost

Zametki projekta izhajajo iz lanskega leta, ko smo razvijali aplikacijo za prikazovanje nevarnih cestnih odsekov. Po koncu projekta smo misli usmerili v ta projekt, intenzivno delo pa se je začelo šele letos spomladi. Ob upoštevanju vseh časovnih rokov bo projekt izveden pravočasno, in sicer v doglednem času štirih mesecev. Ker so denarna sredstva omejena študentu primerno, bomo uporabljali izključno odprtokodno programsko opremo. Pri projektu bo sodeloval tudi somentor lanske zaključne naloge Andreja Godca [5] dr. Branko Bogunovič.

Za uspešnost projekta je ključnega pomena poznavanje ter razumevanje obstoječe podatkovne baze prometnih nesreč, ki jo uporablja slovenska policija. Na podlagi atributov bo izdelan grafični vmesnik s padajočimi seznami, s katerimi bo mogoče podrobno opisati okoliščine prometne nesreče. Velik poudarek bo namenjen geografski lokaciji nesreče, saj bo v aplikacijo vključen tudi namenski zemljevid, ki bo omogočal natančno umestitev nezgode na pripadajočem cestnem odseku.

4 ANALIZA POTREB IN DEFINICIJA ZAHTEV

Pri gradnji programskega produkta je prav gotovo pomembno, da že v začetnih fazah prepoznamo vse morebitne ovire, ki bi lahko na koncu negativno vplivale na uporabniško izkušnjo. Napačna, nepopolna, nekonsistentna ali dvoumna specifikacija zahtev je namreč pogosto vzrok za neuspeh projektov in kasnejše spore. V ta namen se bomo osredotočili na uporabnikove potrebe in zahteve, pri čemer bomo uporabili objektno orientiran pristop z UML diagrami.

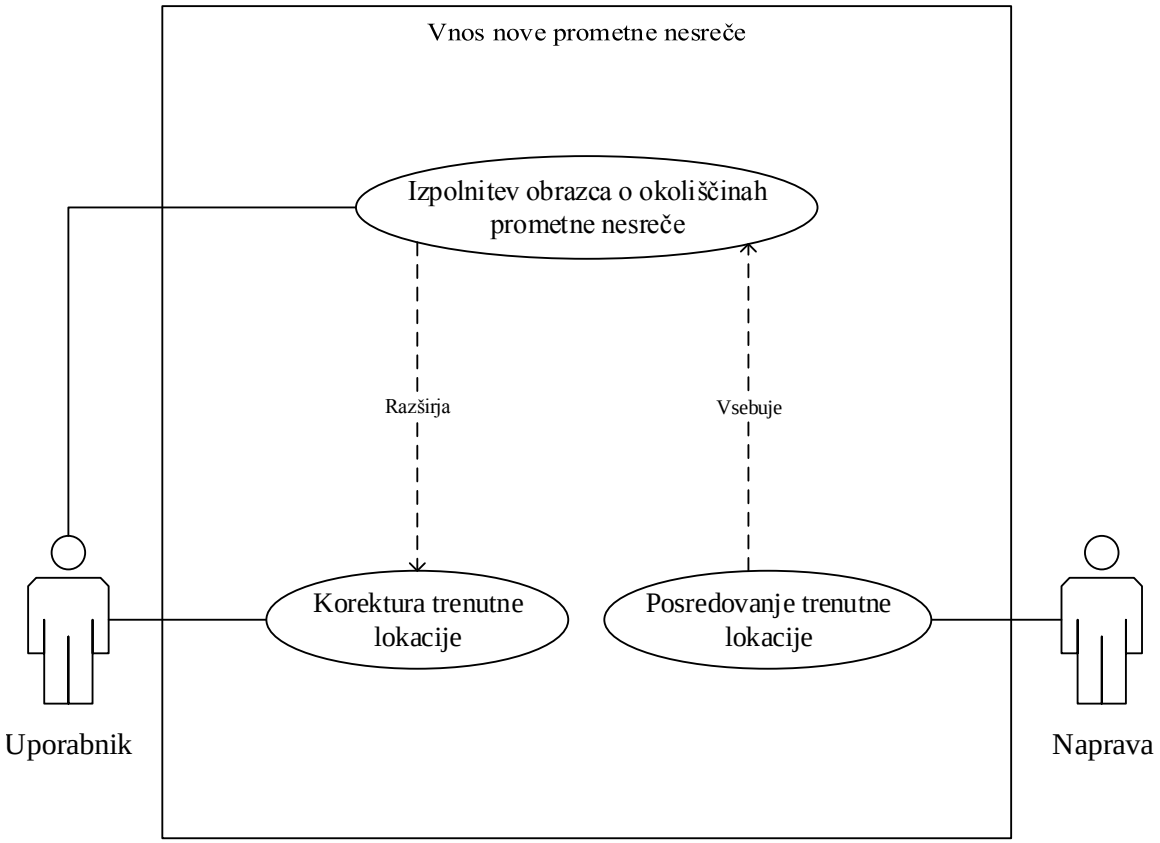
UML ("Unified Modeling Language") ali poenoteni jezik modeliranja je družina grafičnih oznak, podprtih z meta modelom, ki pomagajo pri opisovanju in oblikovanju programskih sistemov, zlasti programske opreme, ki se jo gradi s pomočjo objektno usmerjenega pristopa. UML diagrame delimo glede na njihov namen v dve skupini [8]:

- Strukturni (razredni, komponentni, objektni in paketni diagram)
- Vedenjski (aktivnostni, sekvenčni, komunikacijski, časovni diagram ter diagram stanj in diagram primera uporabe)

Uporaba UML diagramov je pri načrtovanju programskega produkta ključnega pomena za temeljito razumevanje uporabnikovih zahtev ter poznavanje komponent sistema, ki služijo zadovoljevanju uporabnikovih zahtev. Ker v fazi analize ter zbiranja potreb sodelujejo tudi končni uporabniki, je komunikacija glede končnih rešitev ter funkcionalnosti programskega produkta občutno olajšana, če poteka preko UML diagramov. Predstavljajo namreč komunikacijski kanal med strokovnim znanjem razvijalcev programske opreme ter laičnim poznavanjem problematike končnih uporabnikov, ki bodo aplikacijo tudi uporabljali [4].
Namen analize potreb in definiranja zahtev je v celoti opisati obnašanje sistema. Osredotočili se bomo na funkcijske zahteve, kjer bomo obravnavali vse interakcije uporabnika s sistemom, ter ostale (nefunkcijske) zahteve, ki lahko omejujejo načrtovanje ali izvedbo sistema.

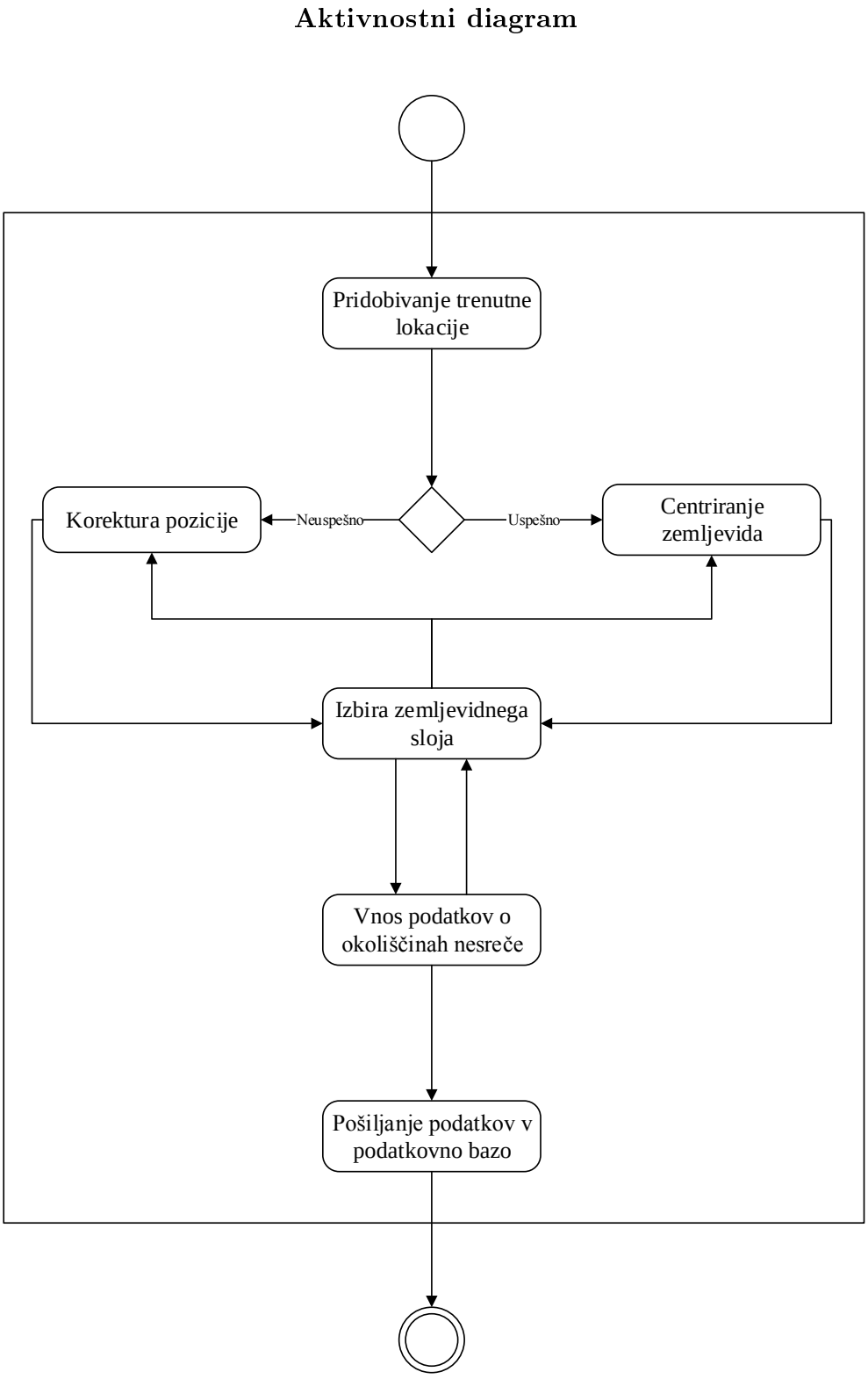
4.1 Funkcijske zahteve

Diagram primera uporabe



Slika 4.1: Diagram primera uporabe za vnos nove prometne nesreče

Slika 4.1 predstavlja diagram primera uporabe za vnašanje nove prometne nesreče v podatkovno bazo s pomočjo namenske aplikacije. Uporabnik ni kdorkoli, temveč le oseba, ki ima pristojnosti na prizorišču prometne nesreče. V našem primeru gre za policijskega uslužbenca. Naprava delno sodeluje tudi kot akter, saj odigra ključno vlogo pri geografski umestitvi prometne nesreče. Trenutno lokacijo namreč pridobi samodejno ter jo preko aplikacije posreduje uporabniku. Ta ima nato možnost še natančneje določiti lokacijo nesreče.

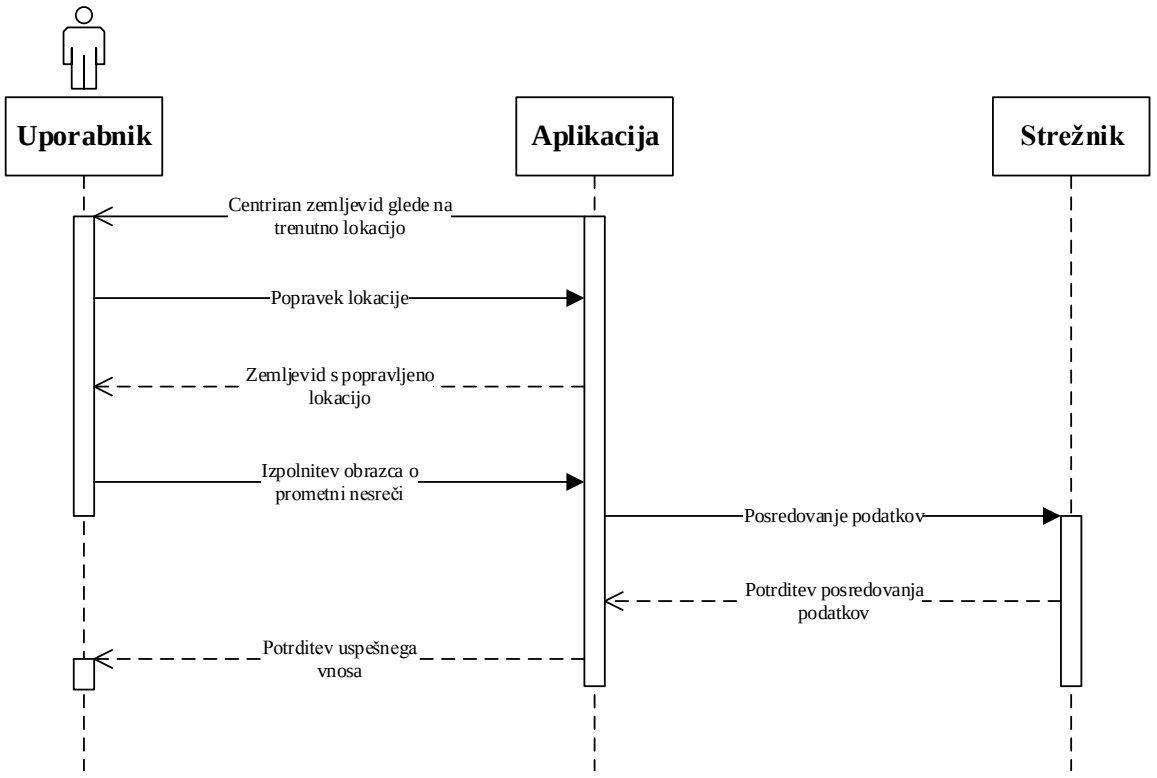


Slika 4.2: Aktivnostni diagram za dodajanje nove prometne nesreče

Na sliki 4.2 se nahaja aktivnostni diagram, ki prikazuje potek vnašanja nove prometne nesreče v podatkovno bazo z uporabo mobilne aplikacije.

1. Ob zagonu aplikacije se sproži pridobivanje trenutne lokacije.
2. Uporabnik oceni natančnost pridobljene lokacije.
   2.1. V kolikor lokacija ni bila natančno določena, jo ročno popravimo.
   2.2. V nasprotnem primeru nadaljujemo z uporabo aplikacije.
3. Na voljo imamo še različne zemljevidne sloje, ki jih lahko menjujemo med uporabo aplikacije.
4. Zatem izpolnimo obrazec o okoliščinah prometne nesreče.
5. Ko so vsi podatki izpolnjeni, jih s klikom na gumb *Potrdi* posredujemo podatkovni bazi.

Sekvenčni diagram



Slika 4.3: Sekvenčni diagram za opis poteka uporabe aplikacije

Slika 4.3 ponazarja komunikacijo v kronološkem zaporedju med uporabnikom, aplikacijo ter strežnikom. Ko uporabnik izpolni vse podatke o nesreči, jih aplikacija nato posreduje strežniku, kjer se zapišejo v podatkovno bazo.

4.2 Nefunkcijske zahteve

Glede na karakteristike lahko nefunkcijske zahteve razdelimo v dve skupini [2]:

- Obratovalne lastnosti (varnost, uporabljivost, zahtevana prepustnost sistema)
- Evolucijske lastnosti (možnost testiranja, zmožnost vzdrževanja, razširljivost, nadgradljivost)

Z zgoraj omenjeno delitvijo nefunkcijskih zahtev želimo problem razgraditi na manjše komponente, katere obravnavamo ločeno in neodvisno. Uporabniki namreč pogosto ne poznajo dejanskih potreb in ne vedo, kako lahko računalniški sistem izboljša njihovo uporabniško izkušnjo.

4.2.1 Obratovalne lastnosti

Varnost

Ker razvijamo aplikacijo za specifičen nabor končnih uporabnikov, v našem primeru gre za policijske uslužbence, je varnost ena ključnih lastnosti, kateri moramo nameniti več pozornosti. Nepravilna uporaba aplikacije ter njena zloraba lahko negativno vplivata na končno klasifikacijo nevarnih cestnih odsekov, kar bi lahko posledično odločalo o življenju in smrti na slovenskih cestah. Zato želimo zagotoviti kredibilnost podatkov z avtentikacijo uradne osebe tik pred pošiljanjem podatkov na strežnik.

Uporabljivost

Aplikacija je bila že od samega začetka načrtovana za enostavno uporabo z minimalističnim videzom. S tem smo želeli ustreči uporabnikovim zahtevam glede preprostosti uporabe. V ta namen je trenutna lokacija pridobljena samodejno takoj po zagonu aplikacije, obrazec, ki opisuje okoliščine prometne nesreče, je nato mogoče izpolniti pretežno z uporabo padajočih seznamov.

Zahtevana prepustnost sistema

Kljub temu da gre za aplikacijo, ne smemo izključiti dejstva, da je ne bo istočasno uporabljalo več uporabnikov. Lahko se namreč zgodi, da bo aplikacijo sočasno uporabljalo več uporabnikov na različnih napravah hkrati. Ker je glavna funkcionalnost aplikacije pošiljanje podatkov v podatkovno bazo na strežniku, bo treba poskrbeti, da bo strežnik dovolj zmogljiv za hranjenje ter procesiranje množice podatkov, saj bo teh z vsakodnevno uporabo aplikacije čedalje več.

4.2.2 Evolucijske lastnosti

Možnost testiranja

Že samo dejstvo, da je testiranje ena izmed faz programskega inženirstva, pove veliko. Testiranja ne smemo zanemarjati pod nobenim pogojem, saj lahko z njegovo pomočjo odkrijemo marsikatero spregledano napako v fazi analize ter kasneje načrtovanja. Ker želimo v polni meri zadovoljiti pričakovanja uporabnikov, se bomo v ta namen posebej posvetili tudi testiranju v kasnejših fazah projekta.

Zmožnost vzdrževanja

V današnjih časih, ko tehnološki razvoj raste z nepredstavljivo hitrostjo, je zelo pomembno, da temu primerno tudi redno vzdržujemo našo aplikacijo. Številne raziskave so namreč pokazale, da je vzdrževanje, tako časovno kot tudi finančno, najdražja faza programskega inženirstva [2]. Zato je ključnega pomena, da morebitna vzdrževanja predvidimo že v fazi načrtovanja ter temu primerno tudi zasnujemo našo aplikacijo. Zgrajena bo namreč na podlagi arhitekture MVC ("Model-View-Controller") oziroma po slovensko "Model-Pogled-Upravitelj", pri čemer so podatki, grafični vmesnik ter kontrolna logika v ozadju popolnoma ločeni ter neodvisni med seboj.

Razširljivost

Dodatne funkcionalnosti doprinesejo aplikaciji dodano vrednost ter s tem pripomorejo h pozitivni uporabniški izkušnji. Iz tega razloga mora biti programski produkt odprt za razne razširljivosti, katere bodo zadovoljile uporabnikove potrebe. V ta namen bomo v aplikacijo kot dodatne funkcionalnosti vključili tudi možnost izbire med različnimi zemljevidnimi sloji, popravek trenutne geografske lokacije ter možnost večanja oziroma manjšanja zemljevida. Vse dodatne funkcionalnosti si bomo tudi podrobneje pogledali v fazi načrtovanja ter kasneje tudi v fazi izvedbe.

Nadgradljivost

Prej omenjena arhitektura MVC bo v prihodnosti omogočala tudi morebitno nadgradnjo aplikacije, saj s posegom v katero izmed komponent ne bomo vplivali na ostale komponente. Že sama arhitekturna zasnova stremi h konstantnemu prilagajanju spreminjajočim se tehnologijam ter potrebam uporabnikov. Z željo po nenehnih izboljšavah bomo konstantno spremljali možnost potencialne nadgradnje, ki bi doprinesla h boljši uporabljivosti aplikacije.

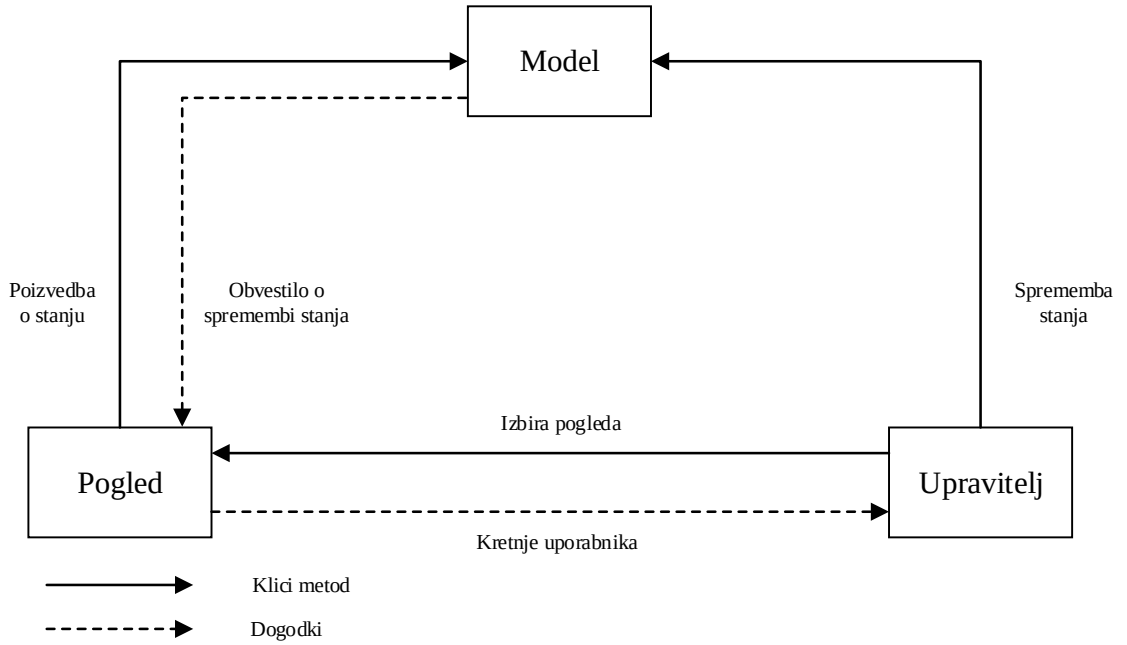
5 NAČRTOVANJE

V fazi analize smo skušali prepoznati vse uporabnikove zahteve ter potrebe. Postopoma smo jih uvrstili med osnovne funkcionalnosti ter vse skupaj zaokrožili v celoto, ki bo tvorila končno aplikacijo. Sedaj je na vrsti načrtovanje programskega produkta, kjer bomo skušali z raznimi tehnologijami ter prijemi pripraviti učinkovito podlago za kasnejšo implementacijo. Predstavili bomo programsko opremo, s katero bomo realizirali delujočo aplikacijo po vzorcu MVC. Slednje pomeni, da bomo ločeno obravnavali podatke, katere bo procesirala aplikacija, grafični vmesnik, s katerim bo operiral končni uporabnik, ter na koncu še kontrolno logiko, ki bo poskrbela za izvajanje osnovnih funkcionalnosti aplikacije.

5.1 Arhitektura MVC

MVC ("Model-View-Controller") je dobro poznana ter priljubljena arhitektura med razvijalci programske opreme za razvijanje razširljivih aplikacij. Medtem ko omenjeno arhitekturo že dolgo uporabljamo na strani strežnikov, se je s kompleksnejšimi aplikacijami pojavila potreba po uporabi MVC arhitekture tudi na strani odjemalcev. Če aplikacija namreč uporablja grafični vmesnik, s katerim uporabnik izvaja različne interakcije s sistemom v ozadju, je prav MVC arhitektura najprimernejša. Vzorec MVC je najboljša možna arhitektura, ko gre za razvoj aplikacije, kjer je predstavitveno logiko moč ločiti od poslovne logike, ki procesira podatke. Določene MVC arhitekture se lahko bistveno razlikujejo od tradicionalne, odvisno od potreb danega sistema. Tako je na primer klasičen MVC vzorec v zadnjih letih dobil različne paralelne alternative kot so HMVC, MVA, MVP, MVVM ter ostale, ki omogočajo prilagoditev na posamezne zahteve sistema [6].

MVC ni oblikovalski vzorec, temveč arhitekturni vzorec, ki opisuje na kakšen način strukturiramo našo aplikacijo in interakcije z uporabnikom za vsak posamezen del v strukturi programskega produkta. Dano programsko opremo razdeli namreč na tri med seboj neodvisne dele, ki jih po slovensko imenujemo model, pogled in upravitelj. Na ta način dosežemo, da je interna obdelava podatkov popolnoma ločena od načina, kako te podatke sprejmemo ali predstavimo uporabniku [6].



Slika 5.1: Komponente MVC arhitekture

Slika 5.1 prikazuje celoten pogled na arhitekturo MVC ter hkrati ponazarja, kako posamezne komponente komunicirajo med seboj z namenom doseganja skupnih ciljev [6].

V nadaljevanju si bomo podrobneje ogledali MVC arhitekturo ter jo skušali aplicirati s Sencha Touch. Gre za razvojno okolje, ki temelji na omenjeni arhitekturi.

5.1.1 Model (Model)

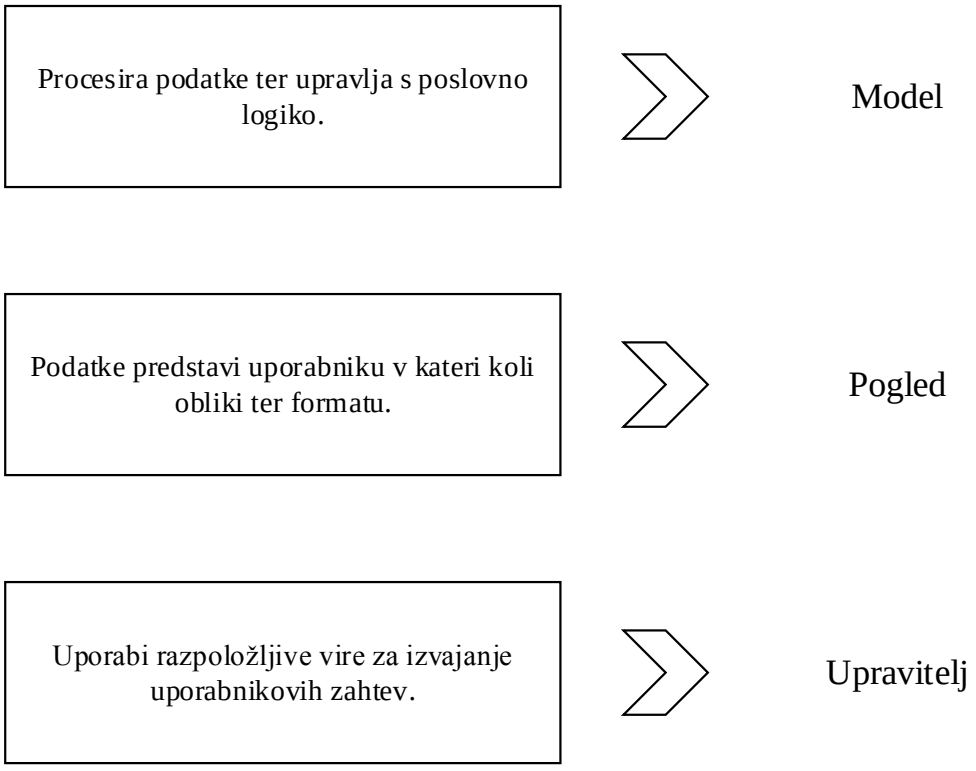
Model predstavlja podatke ter pravila, ki veljajo zanje. Vsebuje najpomembnejši del logike naše aplikacije, ki se nanaša na problem, katerega želimo rešiti z njenim razvojem. V vsakem programskem produktu so vse komponente modelirane s pomočjo podatkov, ki jih obdelujemo na takšen ali drugačen način. Za aplikacijo so objekti iz realnega sveta nič drugega kot le podatki, kateri morajo biti procesirani v skladu z določenimi pravili.
Model posreduje podatke upravitelju, kateri jih kasneje predstavi uporabniku. Ne glede na to, kako bomo podatke predstavili uporabniku, bo podatkovni model vselej ostal enak. Iz tega razloga lahko izbiramo med katerimi koli razpoložljivimi pogledi za njihovo predstavitev [6].

5.1.2 Pogled (View)

Grafični vmesnik posreduje podatke uporabniku v kateri koli podprti obliki in formatu. Omogoča namreč predstavitev podatkov, pridobljenih od modela, na več različnih načinov. Na podlagi uporabnikovih kretenj ter ukazov se krmilnik odloči, kateri pogled bo uporabil. Vse interakcije uporabnika s sistemom potekajo ravno preko grafičnega vmesnika, zato je zelo pomembno, da je ta nedvoumen ter enostaven za uporabo [6].

5.1.3 Upravitelj (Controller)

Krmilnik upravlja z zahtevami uporabnika tako, da sprejme vse njegove zahteve in za njihovo izvedbo uporabi vire, ki so na voljo. Njegova glavna naloga je klicanje in koordiniranje razpoložljivih virov ter predmetov, ki so potrebni za uspešno izvedbo željenega dogodka. Za izvajanje dejavnosti krmilnik običajno izbere ustrezen model ter temu primerno tudi prilagodi grafični vmesnik [6].



Slika 5.2: Osnovne funkcije komponent MVC arhitekture

Slika 5.2 ponazarja razdelitev aplikacije na tri osnovne komponente. Vsaka izmed njih je odgovorna za izpolnjevanje različnih nalog.

5.2 Uporabljena programska oprema

Pri razvoju projekta smo se osredotočili na odprtokodne rešitve. Pri tem smo skušali izkoristiti vse prednosti, ki jih prinaša uporaba odprtokodne programske opreme. Kljub temu, da so uporabljena razvojna orodja popolnoma brezplačna ter na voljo za prenos prek spleta, bo končen videz aplikacije popolnoma profesionalen ter obogaten s paleto različnih funkcionalnosti, ki jih prinaša širok spekter odprtokodne programske opreme.

5.2.1 Sencha Touch

Sencha Touch je vsestransko razvojno okolje, ki temelji na tehnologiji HTML5. Uporablja skriptni jezik JavaScript v kombinaciji s slogovnim jezikom CSS, ki omogočata izdelavo zmogljivih ter nadvse prilagodljivih mobilnih aplikacij. Tako narejene aplikacije lahko uporabljamo kot običajne spletne strani, lahko pa jih prevedemo celo v mobilne aplikacije za naprave na dotik, saj so tako na videz kot tudi po načinu uporabe podobne avtohtonim aplikacijam [7].

Do aplikacije, izdelane na podlagi razvojnega okolja Sencha Touch, se lahko dostopa z uporabo najpopularnejših spletnih brskalnikov. Še več, aplikacijo je mogoče tudi zagnati na marsikateri napravi na dotik. Zadnja verzija (v času pisanja zaključne naloge) Sencha Touch 2.3.1 podpira WebKit, spletni brskalnik Internet Explorer 11 ter naslednje platforme:

- Android
- iOS
- BlackBerry
- Windows Phone 8

Vsaka od navedenih platform zahteva svojevrstno postavitev razvojnega okolja za zagon aplikacije na napravi s pripadajočo platformo [7].

Obstaja več dejavnikov, zakaj smo se odločili za uporabo razvojnega okolja Sencha Touch. Najpomembnejši razlog je zagotovo sočasen razvoj mobilne aplikacije za vse popularne platforme hkrati. Torej to pomeni, da aplikacije ni treba razvijati za vsako platformo posebej. Aplikacijo je mogoče nadgraditi do te mere, da samodejno prilagodi svoj videz glede na mobilno platformo. Integrirana arhitektura MVC omogoča preglednost in enostavno popravljanje kode. Poleg tega Sencha Touch ponuja obširno in zelo pregledno dokumentacijo.

5.2.2 Sencha Architect

Sencha Architect je vizualno razvojno orodje za Sencha Touch ter Ext JS. Na voljo je za platforme, kot so Mac, Windows ter Linux. Ena izmed ključnih prednosti uporabe programske opreme Sencha Architect je, da nam omogoča hitro ustvarjanje vmesnikov ter njihovo testiranje. V ozadju nam namreč "oblikovalec" ustvarja standardne JavaScript datoteke po vzorcu MVC, ki jih je kasneje mogoče urejati s poljubnim urejevalnikom besedil. Ta lastnost nam omogoča, da hitro sestavimo osnovne elemente naše aplikacije, medtem ko preostali, bolj specifičen del, napišemo na roke.
Program ponuja vse razrede in komponente, ki jih vsebuje Sencha Touch v svojih knjižnicah. V aplikacijo jih vključimo s preprosto metodo "povleci in spusti". Poleg tega nam ponuja tudi bogato dokumentacijo neposredno med uporabo programa [1].

Med razvojem naše aplikacije bomo za izdelavo grafičnega vmesnika uporabili Sencha Architect, saj omogoča oblikovanje končnega videza naše aplikacije na intuitiven način z možnostjo sprotnega spremljanja razvoja ter testiranja.

5.2.3 JavaScript

JavaScript je programski jezik svetovnega spletišča. Dandanes že velika večina modernih spletnih strani uporablja JavaScript. Prav tako ga prepoznajo vsi spletni brskalniki, s čimer je prisoten na vseh napravah, ki uporabljajo brskalnike, računalniki, igralne konzole, tablični računalniki ter nazadnje tudi pametni telefoni. Iz tega vidika je JavaScript najpogosteje uporabljen programski jezik vseh časov. JavaScript je poleg tehnologij HTML ter CSS sestavni del spletnih vsebin, saj s svojo dinamično zasnovo opredeljuje njihovo obnašanje. S posebnim pristopom "JavaScript end-to-end" (po slovensko "JavaScript od začetka do konca") je v zadnjem času postal tudi pomemben programski jezik spletnih strežnikov [3].

Posebnost JavaScript leži v njegovi zasnovi. Za razliko od ostalih objektno-usmerjenih programskih jezikov, kateri uporabljajo predmete, ki temeljijo na razredih, JavaScript uporablja predmete, ki so osnovani na podlagi prototipov. V praksi to pomeni, da ustvarimo objekt, ki bo videti kot vsi ostali objekti danega tipa, nato sporočimo JavaScript prevajalniku, da želimo še več takih objektov. Takšen pristop je enostavnejši ter hitrejši za programiranje, saj obstaja le ena instanca predmeta. Prototipe namreč definiramo sproti, ko jih potrebujemo. V tabeli 5.1 je prikazana primerjava med uporabo razredov in prototipov za ustvarjanje objektov [9].

Primer uporabe razreda	Primer uporabe prototipa

`public class Prisoner {`
`  public int sentence = 4;`
`  public int probation = 2;`
`  public string name = "Joe";`
`  public int id = 1234;`
`}`
`Prisoner prisoner = new Prisoner();`
	`var prisoner = {`
	`  sentence : 4,`
	`  probation : 2,`
	`  name : 'Joe',`
	`  id : 1234`
	`};`

Tabela 5.1: Primerjava med uporabo razredov in prototipov za ustvarjanje objektov

5.2.4 OpenLayers

OpenLayers je odprtokodna JavaScript knjižnica za izdelavo interaktivnih spletnih zemljevidov, ki jih je mogoče prikazati v večini modernejših spletnih brskalnikih. Kljub temu da omenjeno knjižnico uporabljajo izključno odjemalci na svojih brskalnikih, strežnik ne potrebuje nobene dodatne programske opreme oziroma posebnih nastavitev. Uporaba je mogoča celo brez kakršnih koli dodatnih prenosov s spleta. Knjižnico je prvotno razvilo podjetje Metacarta kot odgovor na popularne zemljevide Google Maps. Sčasoma se je po zaslugi marljivih razvijalcev prelevilo v nadvse priljubljeno razvojno okolje.
Knjižnica OpenLayers nam omogoča izgradnjo celotne kartografske aplikacije praktično iz nič. Ponuja namreč možnost, da svojim potrebam prilagodimo vsak detajl zemljevida, sloje, kontrolno logiko, dogodke in podobno. Prava moč knjižnice OpenLayers se skriva v sočasni uporabi različnih zemljevidov ter slojev, vključno z vsestranskim in nadvse zmogljivim vektorskim slojem [11].

Ena izmed ključnih funkcionalnosti naše aplikacije bo vključevala zemljevid, ki bo temeljil na zgoraj omenjeni knjižnici OpenLayers. Na voljo bo več različnih slojev, med katerimi bo lahko izbiral uporabnik:

- OpenStreetMap Standard
- OpenStreetMap Transport
- Bing Road

5.2.5 Python

Python je sodoben, visoko nivojski programski jezik, primeren za širok nabor programerskih opravil. Tehnično je opredeljen kot skriptni jezik, vendar se je v praksi že večkrat izkazalo, da je vsestranski programski jezik.
Python se nahaja vsepovsod – na spletnih sistemih, namiznih aplikacijah, v igrah, znanstvenih programih ter celo v višjih nivojih nekaterih operacijskih sistemov.
Podpira različne programerske tehnike, od enostavnega postopkovnega programiranja ter vse tja do objektno usmerjenega in funkcijskega programiranja [12].

Python je primeren za vse vrste programiranja, od razvoja velikih in kompleksnih sistemov do pisanja kratkih skript. Pri slednji lastnosti smo prepoznali njegov potencial, saj bomo njegovo vsestranskost in hitrost uporabili za izdelavo različnih skript, ki bodo podatke prenašale iz mobilne aplikacije v podatkovno bazo na strežniku.

5.2.6 PostgreSQL

PostgreSQL je odprtokodni sistem za upravljanje objektno relacijskih podatkovnih baz, ki je bil razvit na Univerzi Berkeley v Kaliforniji. PostgreSQL je prvotno spadal pod licenco BSD, medtem ko je sedaj izvorna koda dostopna pod licenco PostgreSQL License. Kljub vsemu je še vedno odprtokodna programska oprema, kar pomeni, da je njena uporaba, spreminjanje ter distribucija brezplačna. Za sabo ima vrsto let aktivnega razvoja ter izboljšav, s čimer si je PostgreSQL pridobil močan ugled, kar se tiče stabilnosti, integritete podatkov in korektnosti. Kljub temu da je bil PostgreSQL prvotno razvit izključno za UNIX sisteme, danes podpira tudi ostale platforme (Mac OS X, Solaris, Windows in druge).
PostgreSQL je zanimiv za uporabnike predvsem zaradi številnih funkcionalnosti ter razširitev. Tako na primer poleg jezika SQL ("Structured Query Language"), ki je značilen za podatkovne baze, podpira množico ostalih programskih jezikov za pisanje dodatnih funkcij, kot so Python, Java, Perl, C in ostali. Omeniti še velja razširitev PostGIS, s katero je hranjenje ter procesiranje geografskih podatkov v standardni obliki sploh mogoče [10].

Kot je bilo že večkrat omenjeno, je ta projekt nadaljevanje lanske zaključne naloge Andreja Godca [5], ki je pri razvoju aplikacije za prikazovanje nevarnih cestnih odsekov uporabil podatkovno bazo PostgreSQL. Zaradi tega bomo obstoječo podatkovno bazo uporabili tudi pri razvoju mobilne aplikacije za evidentiranje prometnih nesreč, v katero bomo shranjevali nove prometne nesreče.

6 IZVEDBA

V fazi izvedbe bomo na podlagi zbranih zahtev ter potreb z načrtovalskimi orodji predstavili potek implementacije. Kot se za aplikacijo, ki temelji na arhitekturi MVC spodobi, bomo potek implementacije predstavili v treh delih. Najprej bomo definirali podatkovni model, nato bomo oblikovali grafični vmesnik, za konec sledi še implementacija kontrolne logike, ki upravlja delovanje posameznih komponent aplikacije.

6.1 Podatkovni model

Model predstavlja podatke ter pravila, po katerih so predstavljeni uporabniku. Kljub temu da uporabnik nima neposrednega stika z modelom, ga neprestano koristi med uporabo aplikacije.
Sencha Touch za predstavitev podatkov predvideva uporabo dveh komponent [7]:

- Model – definira pravila ter format podatkov
- Store – predstavlja dejanske podatke

Glede na datotečno hierarhijo arhitekture MVC definiramo nove modele v direktoriju *app/model/*, medtem ko "skladišča" pripravimo v direktoriju *app/store/* [7].
S podatki se ukvarjata dva tipa datotek:

- .js – datoteka JavaScript, s katero definiramo modele ter skladišča
- .json – notacija JavaScript objektov, v katerih hranimo podatke

Kot bomo videli kasneje pri oblikovanju grafičnega vmesnika, smo obrazec za opis okoliščin prometne nesreče razvili pretežno z uporabo padajočih seznamov. Podatke, ki jih ponujajo, smo shranili v *.json* datoteke, ki jih nato z uporabo skladišč ponudimo v uporabo uporabniku.
Posamezna vrstica *.json* datoteke je sestavljena iz dveh vrednosti:

```
fields: [
  { name: 'value',
    type: 'string'
  },
  {
```

```
        name: 'text',
        type: 'string'
      }
  ]
```

Atribut *value* hrani vrednost, ki se bo zapisala v podatkovno bazo, medtem ko se vrednost atributa *text* prikaže uporabniku na zaslonu [16].

Kot je bilo že omenjeno, smo se pri izdelavi obrazca za opis okoliščin prometne nesreče zgledovali po že obstoječi podatkovni bazi policije. Kot je razvidno iz tabele 6.1, smo obstoječe geografske atribute nadomestili z atributoma "geografska širina" ter "geografska dolžina", ki ju priskrbi mobilna naprava. Uporaba novih, primernejših geografskih atributov za določitev točne lokacije prometne nesreče je bil namreč primarni vzrok za izdelavo aplikacije.

Vsi atributi	Uporabljeni atributi	Dodatni atributi
Številka zadeve	✓	
Datum nesreče	✓	
Ura nesreče	✓	
Upravna enota	✓	
Prizorišče nesreče	✓	
Kategorija ceste	✓	
Oznaka ceste ali šifra naselja	✗	
Tekst ceste ali naselja	✗	Geografska širina
Oznaka odseka ali šifra ulice	✗	
Tekst odseka ali ulice	✗	Geografska dolžina
Stacionaža ali hišna številka	✗	
Opis kraja dogodka	✓	
Klasifikacija nesreče	✓	
Stanje vozišča	✓	
Stanje prometa	✓	
Vrsta vozišča	✓	
Tip nesreče	✓	
Vremenske okoliščine	✓	
Vzrok nesreče	✓	

Tabela 6.1: Atributi, ki jih uporablja aplikacija

Obrazec je nato nastal na podlagi šifranta, ki opisuje posamezne atribute ter njihove vrednosti [14]. Poleg vhodnih podatkov bo aplikacija generirala tudi izhodne. Ko bo uporabnik izpolnil vse podatke ter pritisnil na gumb *Potrdi*, se bodo podatki najprej shranili v lokalni spomin naprave, šele nato bodo poslani v podatkovno bazo na strežniku.

```
requires: [
  'Nesreca.model.Nesreca',
  'Ext.data.proxy.LocalStorage'
],
config: {
  model: 'Nesreca.model.Nesreca',
  storeId: 'NesreceStore',
  proxy: {
    type: 'localstorage',
    id: 'NesreceProxy'
  }
}
```

Opomba 6.1. Z zgornjo kodo definiramo skladišče, kamor lokalno shranimo podatke, preden jih pošljemo na strežnik.

6.2 Grafični vmesnik

Grafični vmesnik predstavlja prvi stik uporabnika z aplikacijo, zato smo dali velik poudarek prav na oblikovanje končnega videza aplikacije. Držali smo se osnovnih načel razvoja uporabljive aplikacije, kot so minimalizem, preprostost, enostavnost ter samoumevnost. Uporabo aplikacije smo skušali približati uporabniku na popolnoma elementaren nivo, kjer ima vse, kar potrebuje, na dosegu prsta.
Končni videz aplikacije je rezultat zbranih zahtev ter potreb končnega uporabnika. Pri tem smo skušali vključiti vse uporabnikove zahteve v končni videz aplikacije v obliki komponent. Končni uporabnik namreč s celotnim sistemom komunicira prav preko grafičnega vmesnika ter tako posredno vključuje tudi ostale funkcijske zahteve.

Glede na to da je aplikacija zgrajena po vzorcu MVC, je temu primerno urejena tudi datotečna hierarhija. Vse datoteke, ki definirajo grafični vmesnik, se namreč nahajajo v direktoriju *app/view/* [7].

Grafični vmesnik je razdeljen na dva dela:

- Zemljevid
- Obrazec

Prvi se prikaže takoj po zagonu aplikacije in vsebuje zemljevid s točko, ki ponazarja trenutno lokacijo, medtem ko drugi del predstavlja obrazec za opis okoliščin prometne nesreče. Med pogledoma se lahko premikamo s pritiskom na gumb *Nova zadeva* ali na gumb *Nazaj*.

Slika 6.1: Zemljevid s točko, ki ponazarja trenutno lokacijo

Kot je prikazano na sliki 6.1, se ob zagonu aplikacije naloži zemljevid. Osnovan je na podlagi knjižnice OpenLayers, ki omogoča uporabo različnih zemljevidnih slojev. V našem primeru smo uporabniku omogočili izbiro med sledečimi sloji:

- OSM Standard
- OSM Transport
- Bing Road

Sloje lahko spreminjamo med uporabo aplikacije glede na željen prikaz cestnih odsekov.

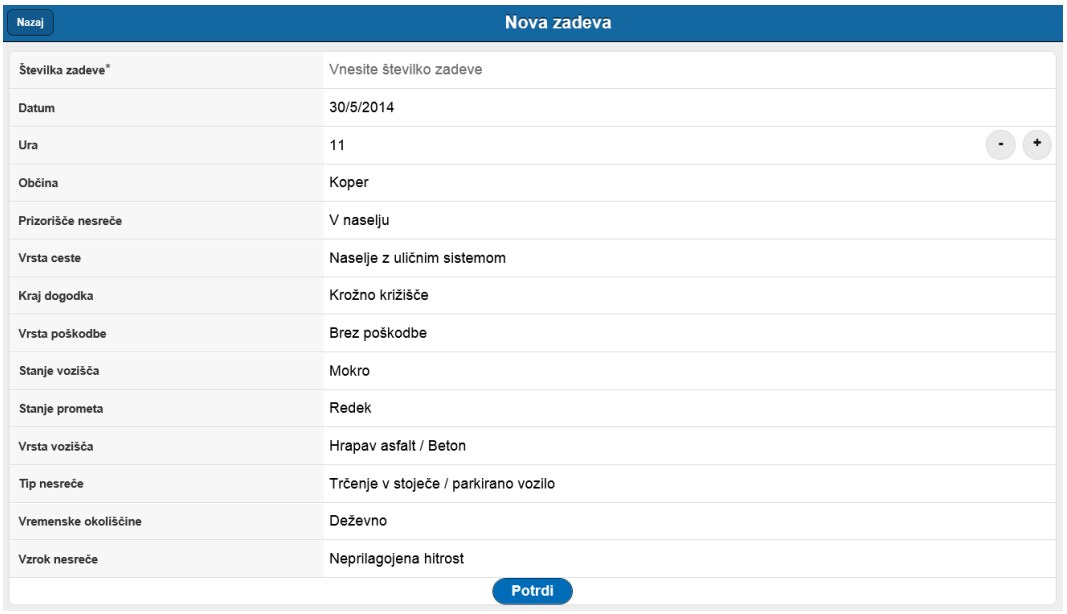
Poleg naštetih slojev smo uporabili še vektorski sloj, ki omogoča prikaz točke s trenutno lokacijo. Zanj smo se odločili, ker ponuja posebno funkcionalnost *DragFeature* [13]. Omenjena lastnost nam namreč omogoča, da lahko premikamo točko poljubno po zemljevidu. Na ta način lahko popravimo napako, do katere ponavadi pride pri določanju trenutne lokacije.

Pridobivanje trenutne geolokacije je v razvojnem okolju Sencha Touch poenostavljeno, saj zadostuje že uporaba posebne spremenljivke ob inicializaciji zemljevida [15]:

```
useCurrentLocation: true
```

Koordinate tako pridobljene lokacije smo uporabili tudi za postavitev točke na zemljevidu, ki označuje trenutno lokacijo.

Ob izdelavi obrazca za opis okoliščin prometne nesreče na sliki 6.2 smo uporabniku želeli prihraniti čim več časa ter truda pri izpolnjevanju omenjenega obrazca. V ta namen mu aplikacija ponuja uporabo padajočih seznamov, s pomočjo katerih lahko opiše posamezne okoliščine prometne nesreče. Obrazec temelji na obstoječi podatkovni bazi policije [14] in ponuja le tiste vrednosti atributov, ki jih tudi sami uporabljajo.



Slika 6.2: Obrazec za opis okoliščin prometne nesreče

Obrazec sestoji iz treh komponent, ki so na voljo uporabniku za uporabo:

- Atributi, ki jih je treba ročno izpolniti
- Atributi, ki uporabljajo padajoče sezname
- Gumb *Potrdi*

Številko zadeve, datum ter uro mora uporabnik izpolniti sam, medtem ko padajoči seznami pridobijo vrednosti za prikaz iz datotek *.json*. Slednje so bile kreirane na podlagi šifranta policije, s pomočjo katerega lahko razumemo vrednosti atributov v podatkovni bazi [14].

Na koncu imamo še gumb *Potrdi*, ki vse izpolnjene podatke pošlje v podatkovno bazo na strežniku. Poleg izpolnjenega obrazca se podatkovni bazi posredujejo tudi geografski podatki naprave, natančneje geografska širina ter dolžina.

6.3 Kontrolna logika

Uporabnikova interakcija s sistemom poteka preko grafičnega vmesnika, vendar vse ukaze ter dejanja v ozadju prevzame upravitelj. Ta nato v sodelovanju z modelom uporabniku odgovori z ustrezno obdelanimi podatki. Taka ureditev nam namreč omogoča, da poslovno logiko popolnoma ločimo od podatkov ter uporabniškega vmesnika aplikacije.

Datotečna hierarhija razvojnega okolja Sencha Touch temelji na vzorcu MVC. Temu primerno se datoteke, ki opravljajo poslovno logiko, nahajajo v direktoriju *app/controller* [7]. Datoteke so tipa *.js*, kar ni nič nenavadnega, glede na to da Sencha Touch uporablja JavaScript. Upravitelj, ki krmili delovanje aplikacije, je lahko tudi razdeljen v več datotek. S tem bi pridobili na preglednosti pri kasnejših popravkih.

Delovanje naše aplikacije bi lahko razdelili na dva dela:

- Upravljanje zemljevida
- Pošiljanje podatkov

Zemljevid je bil zgrajen na podlagi knjižnice OL, ki omogoča uporabo različnih slojev. V našem primeru smo uporabniku ponudili možnost izbire med temi sloji s pomočjo razreda *LayersSwitcher* [11]:

```
map.addControl(new OpenLayers.Control.LayerSwitcher());
map.addLayers([bingmap,transportmap,vectorlayer]);
```

Poleg menjave slojev smo predvideli še eno pomembno funkcionalnost, to je možnost premikanja točke, ki označuje našo trenutno lokacijo. Točka je del vektorskega sloja, ki omogoča integracijo raznih elementov na zemljevidu [13]. Po premiku si točka zapomni svoje nove koordinate, ki jih nato uporabimo za popravek samodejno pridobljene lokacije.

```
marker = new OpenLayers.Feature.Vector(point, null);
vectorlayer.addFeatures([marker]);

drag = new OpenLayers.Control.DragFeature(vectorlayer, {
  autoActivate: true
});
map.addControl(drag);
```

Opomba 6.2. Zgornja demonstracija predstavlja le del celotne kode.

Drugo pomembno funkcionalnost aplikacije predstavlja pošiljanje podatkov v podatkovno bazo na strežniku. Ko uporabnik izpolni obrazec ter pritisne gumb *Potrdi*, kontrolna logika poskrbi tudi za posredovanje podatkov strežniku. Pošiljanje podatkov se izvede v več fazah:

1. validacija izpolnjenega obrazca;
2. shranjevanje izpolnjenih podatkov;
   2.1. zapis podatkov v lokalni spomin naprave;
   2.2. zapis podatkov v *.json* datoteko;
3. avtentikacija s podatkovno bazo na strežniku;
4. zapis podatkov v podatkovno bazo.

Pred shranjevanjem podatkov moramo najprej poskrbeti za njihovo korektnost, saj ne želimo zapisovati nepopolnih podatkov v podatkovno bazo. Za to poskrbi namenska funkcija takoj po pritisku na gumb *Potrdi*, ki preveri, ali katero polje vsebuje vrednost *null* oziroma ničelno vrednost. V kolikor metoda odkrije podobne pomanjkljivosti, o tem obvesti uporabnika s pojavnim sporočilom na zaslonu naprave. Šele po tem, ko uporabnik ustrezno izpolni obrazec za opis okoliščin prometne nesreče, se nato vrednosti izpolnjenega obrazca na podlagi šifranta shranijo v lokalni spomin naprave skupaj z geografsko širino in dolžino trenutne lokacije. Namensko skladišče zatem uporabi JsonP Proxy za zapis shranjenih podatkov v *.json* datoteko. Podatki so na ta način pripravljeni za pošiljanje.
Pred pošiljanjem podatkov se mora uporabnik identificirati z uporabniškim imenom in geslom, s čimer poskrbimo za verodostojnost podatkov ter hkrati ohranjamo želeni varnostni nivo podatkovne baze. Uspešnost avtorizacije preverja posebna Python skripta *checkPass.cgi* na strežniku, kjer se nahaja tudi podatkovna baza. Šele po uspešni avtorizaciji se nato podatki tudi zapišejo v podatkovno bazo PostgreSQL s pomočjo Python skripte *writeJson.cgi*.

7 TESTIRANJE

Za uspešnost programskega produkta je ključnega pomena tudi intenzivno in učinkovito testiranje, kjer lahko odkrijemo številne pomanjkljivosti ter napake še pred predajo aplikacije končnim uporabnikom.

Testiranje bo izvedeno v dveh fazah. Najprej bomo sestavili načrt testiranja ter predvideli obnašanje kritičnih delov sistema. Zatem sledi preizkus obnašanja posameznih komponent aplikacije na podlagi prej izdelanega plana testiranja.

7.1 Načrt testiranja

Pred samo izvedbo testiranja je na podlagi potreb uporabnikov dobro vedeti, kakšne rezultate lahko pričakujemo od posameznih komponent sistema. V ta namen smo izdelali tabelo 7.1, ki nazorno prikazuje komponente, ki jih bomo testirali, ter njihov pričakovan rezultat.

Testirane komponente	Pričakovan rezultat
Pridobivanje trenutne lokacije	Natančnost do 10 metrov
Nalaganje zemljevida	Nalaganje do 5 sekund
Menjava zemljevidnega sloja	Nemotena zamenjava
Premikanje točke	Premik na željeno pozicijo
Padajoči seznami	Prikaz ustreznih vrednosti
Uporaba gumbov	Izpolnjevanje svojih nalog
Gumb *Potrdi*	Priprava podatkov za pošiljanje
Obnašanje v različnih brskalnikih	Brez odstopanja

Tabela 7.1: Testirane komponente in pričakovani rezultati

Testiranja se bomo lotili z uporabno modernejših spletnih brskalnikov (Google Chrome 36, Mozilla Firefox 31, Internet Explorer 11 in Safari 5.1.7), ki so v času pisanja zaključne naloge najnovejše verzije za operacijski sistem Windows 8.1.

Aplikacija je zaenkrat dostopna le z uporabo spletnih brskalnikov, saj želimo pred imigracijo na mobilne naprave oziroma tablične računalnike aplikacijo temeljito testirati.

7.2 Najdene napake in popravki

Vse elemente testiranja smo preizkusili na različnih brskalnikih. Večjih odstopanj nismo opazili, omeniti moramo le brskalnik Safari, ki ni uspel prikazati zemljevida. Težava se najverjetneje nahaja v zastarelosti brskalnika, saj za operacijski sistem Windows 8.1 v zadnjih dveh letih ni bilo večje posodobitve omenjenega brskalnika. Na novejših verzijah brskalnika Safari, preizkušenih na Applovih napravah, omenjenega problema namreč nismo zasledili.

Pridobivanje trenutne lokacije preko brezžičnega domačega omrežja je potekalo nemoteno in hitro. Anomalije je bilo moč opaziti med uporabo mobilnega omrežja, ko sta na trenutke odpovedala tako Chrome kot tudi Firefox, medtem ko se je Internet Explorer vselej pravilno odzval. Z zemljevidom ter ostalimi elementi, povezanimi z njim, ni bilo nobenih težav. Zemljevid se je naložil v doglednem času, prav tako ostali zemljevidni sloji. Točko, ki označuje trenutno lokacijo, smo brez težav premikali po celotni površini zemljevida.

Sestavne grafične komponente aplikacije so delovale brez večjih posebnosti. Gumbi so uspešno opravljali svoje funkcije, prav tako so brezhibno delovali tudi padajoči seznami. Vselej so prikazali pričakovane vrednosti, izbor posamezne vrednosti se je tudi izpisal na ustreznem mestu v obrazcu za opis okoliščin prometne nesreče.

Po kliku gumba *Potrdi* se pričakuje, da aplikacija pripravi vse geografske podatke ter vrednosti izpolnjenega obrazca. Zatem se podatki shranijo v internem spominu aplikacije v obliki podatkovnega zapisa. Težava nastopi, ko želimo shraniti šumnike. Nekateri atributi iz podatkovne baze policije uporabljajo namreč šumnike. Eden izmed takšnih je na primer atribut “Čelno trčenje”, ki na podlagi šifranta uporablja oznako “ČT”, ter atribut “Železniški prehod”, ki uporablja oznako “Ž” [14]. Odkrite pomanjkljivosti bomo skušali odpraviti na strežniku, kjer bomo ponovno rekonstruirali šumnike.

8 ZAKLJUČEK IN NADALJNJE DELO

S tem poglavjem zaključujemo pregled razvoja mobilne aplikacije za evidentiranje prometnih nesreč v Republiki Sloveniji. Z omenjeno aplikacijo želimo pristojnim organom poenostaviti delo na terenu ter jim hkrati prihraniti nekaj časa, ki bi ga lahko efektivno uporabili za pomoč ljudem v stiski. Obenem smo želeli izboljšati spletno aplikacijo za prikaz nevarnih cestnih odsekov, ki je nastala v okviru zaključne naloge Andreja Godca [5]. Z redno in pravilno uporabo aplikacije za evidentiranje prometnih nesreč bi dobili jasnejšo sliko kritičnih predelov slovenskih cestišč. S tem bi morda uspeli prepričati pristojne organe o posodobitvi oziroma sanaciji najnevarnejših predelov cestnega omrežja v upanju, da bi s tem zmanjšali število prometnih nesreč. V tujini so na vidiku najrazličnejši projekti, ki predvidevajo popolno digitalizacijo cest. Eden izmed takih projektov predvideva uporabo prometne signalizacije neposredno na cestnih površinah, s čimer bi bil voznik neprestano na tekočem o najaktualnejših cestno-prometnih razmerah.

Aplikacija za vnašanje novih prometnih nesreč predstavlja nadaljevanje spletne aplikacije za prikazovanje nevarnih cestnih odsekov, katere avtor je Andrej Godec [5]. Z novonastalo aplikacijo smo želeli odpraviti nekatere pomanjkljivosti omenjene aplikacije. Z redno uporabo aplikacije za vnašanje prometnih nesreč bi namreč lahko vsakodnevno vnašali nove podatke o prometnih nesrečah v podatkovno bazo. S tem bi bila aplikacija za prikazovanje nevarnih cestnih odsekov neprestano ažurirana z najaktualnejšimi podatki. Poleg tega smo želeli tudi začrtati nove smernice tovrstnim aplikacijam. Na tem področju ostaja namreč še veliko maneverskega prostora pri razvoju podobnih programskih orodij. Ena izmed takšnih orodij bi lahko bila mobilna aplikacija, ki bi na mobilnih napravah dejansko prikazovala vse nevarne cestne odseke na naših cestah ter o tem predhodno obvestila voznika. Ta bi nato prihajajoči nevarni cestni odsek odpeljal previdneje, s čimer bi se lahko izognil morebitni prometni nesreči. “Kdor reši eno življenje, reši cel svet.”

Literatura

[1] J. E. Clark in B. P. Johnson, *Creating Mobile Apps with Sencha Touch 2*, Packt Publishing, 2013. *(Citirano na strani 15.)*

[2] A. Dennis, *Systems Analysis and Design, Fifth Edition*, Wiley, 2012. *(Citirano na straneh 9 in 10.)*

[3] D. Flanagan, *JavaScript Pocket Reference, Third Edition*, O'Reilly Media, 2012. *(Citirano na strani 15.)*

[4] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition*, Addison-Wesley Professional, 2003. *(Citirano na strani 5.)*

[5] A. Godec, *Izdelava aplikacije za prikazovanje nevarnih cestnih odsekov v Republiki Sloveniji: zaključna naloga* , Univerza na Primorskem, Fakulteta za matematiko, naravoslovje in informacijske tehnologije, 2013. *(Citirano na straneh 1, 3, 4, 17 in 27.)*

[6] A. Kumar, *Sencha MVC Architecture*, Packt Publishing, 2012. *(Citirano na straneh 11, 12 in 13.)*

[7] A. Kumar, *Sencha Touch Cookbook, Second Edition*, Packt Publishing, 2013. *(Citirano na straneh 14, 18, 20 in 23.)*

[8] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, Addison-Wesley Professional, 2004. *(Citirano na strani 5.)*

[9] M. S. Mikowski in J. C. Powell, *Single Page Web Applications: JavaScript end-to-end*, Manning Publications, 2013. *(Citirano na strani 15.)*

[10] R. Obe in L. Hsu, *PostgreSQL: Up and Running*, O'Reilly Media, 2012. *(Citirano na strani 17.)*

[11] A. S. Perez, *OpenLayers Cookbook*, Packt Publishing, 2012. *(Citirano na straneh 16 in 23.)*

[12] E. Westra, *Python Geospatial Development*, Packt Publishing, 2010. *(Citirano na strani 17.)*

Viri

[13] OpenLayers `http://openlayers.org/` 2014 *(Citirano na straneh 21 in 23.)*

[14] Policija Republike Slovenije `http://www.policija.si` 2014 *(Citirano na straneh 3, 19, 22 in 26.)*

[15] Sencha Architect `http://docs.sencha.com/architect/3/` 2014 *(Citirano na strani 21.)*

[16] Sencha Touch Documentation `http://docs-origin.sencha.com/touch/2.3.1/` 2014 *(Citirano na strani 19.)*

Priloge

Priloga A: Upravljanje zemljevida

Priloga vsebuje del kode upravitelja glede na arhitekturo MVC, ki skrbi za izvajanje funkcij nad zemljevidom.

```
Ext.define('Nesreca.controller.MapView', {
  extend: 'Ext.app.Controller',

  config: {
    refs: {
      mainView: {
        selector: 'mainview',
        xtype: 'Ext.navigation.View'
      },
      mapPanel: {
        selector: 'mainview #mapPanel',
        xtype: 'Ext.Panel'
      },
      openLayersMap: {
        selector: 'mainview #openlayersmap',
        xtype: 'Ext.ux.OpenLayersMap'
      }
    },
    control: {
      "openLayersMap": {
        maprender: 'onMapRender'
      }
    }
  },

  onMapRender: function(component, map, layer) {
    var EPSG4326 = new OpenLayers.Projection("EPSG:4326");
    var EPSG900913 = new OpenLayers.Projection("EPSG:900913");
    var position = new OpenLayers.LonLat(component._geo._longitude,
        component._geo._latitude).transform( EPSG4326, EPSG900913);
    map.setCenter(position, 15);


```

```
33      // Layer switcher
34      map.addControl(new OpenLayers.Control.LayerSwitcher());
35
36      // OSM Standard
37      map.baseLayer.name = 'OpenStreetMap Standard';
38
39      // OSM Transport map layer
40      var transportmap = new OpenLayers.Layer.OSM("OpenStreetMap
            Transport",
41      ["http://a.tile.opencyclemap.org/transport/${z}/${x}/${y}.png",
42      "http://b.tile.opencyclemap.org/transport/${z}/${x}/${y}.png",
43      "http://c.tile.opencyclemap.org/transport/${z}/${x}/${y}.png"]
44      );
45
46      // Bing Layer
47      var bingmap = new OpenLayers.Layer.Bing({
48        name: "Bing Road",
49        type: "Road",
50        key: "apiKey",
51      });
52
53      // Vector layer
54      var vectorlayer = new OpenLayers.Layer.Vector("Vector Layer");
55
56      // Add layers to map
57      map.addLayers([transportmap,bingmap,vectorlayer]);
58
59      // Point object which represents draggable marker
60      var point = new OpenLayers.Geometry.Point(position.lon,position
            .lat);
61      marker = new OpenLayers.Feature.Vector(point, null, {
62        externalGraphic: "resources/icons/target.png",
63        graphicWidth: 48,
64        graphicHeight: 48,
65        fillOpacity: 1
66      });
67      vectorlayer.addFeatures([marker]);
68
69    // Drag Feature
70      drag = new OpenLayers.Control.DragFeature(vectorlayer, {
71        autoActivate: true,
72      });
73      map.addControl(drag);
74    }
```

Priloga B: Shranjevanje podatkov

Priloga vsebuje del kode upravitelja glede na arhitekturo MVC, ki po pritisku gumba *Potrdi* v lokalni spomin shrani atribute izpolnjenega obrazca.

```
Ext.define('Nesreca.controller.SaveData', {
  extend: 'Ext.app.Controller',

  config: {
    models: [
      'Nesreca',
    ],
    stores: [
      'NesreceStore',
      'LOVC',
      'PRKD',
      'PRPO',
      'PRPV',
      'PRSP',
      'PRSV',
      'PRTN',
      'PRVR',
      'PRVZ'
    ],
    refs: {
      novaNesrecaForm: 'mainview #nesrecaPanel'
    },
    control: {
      "button#potrdiButton": {
        tap: 'onPotrdiButtonTap'
      },

  onPotrdiButtonTap: function(button, e, eOpts) {

    // Get the form and its values
    var form = this.getNovaNesrecaForm(),
        values = form.getValues();

```

```
34      // Find the map
35      var map = this.getOpenLayersMap();
36
37      // Build the location
38      var currentPosition = new OpenLayers.LonLat(map._geo._longitude
            , map._geo._latitude);
39
40      // Get the store
41      var store = Ext.getStore('NesreceStore');
42
43      // Adjust date format
44      var date = Ext.Date.format(values.datum, 'j/n/Y');
45
46      // Add this to the store
47      store.add({
48        zadeva: values.zadeva,
49        datum: date,
50        ura: values.ura,
51        LOOB: values.LOOB,
52        naselje: values.naselje,
53        LOVC: values.LOVC,
54        PRKD: values.PRKD,
55        PRPO: values.PRPO,
56        PRPV: values.PRPV,
57        PRSP: values.PRSP,
58        PRSV: values.PRSV,
59        PRTN: values.PRTN,
60        PRVR: values.PRVR,
61        PRVZ: values.PRVZ,
62        latitude: currentPosition.lat,
63        longitude: currentPosition.lon
64      });
65      store.sync();
66    }
```

Priloga C: Šifrant

Obrazec s katerim opišemo okoliščine prometne nesreče je nastal na podlagi šifranta, ki ga trenutno uporabljajo pristojni organi za opis prometne nesreče v svoji podatkovni bazi.

1. Številka zadeve – enolični identifikator prometne nesreče
2. Datum nesreče določen v formatu dan/mesec/leto
3. Ura nesreče omogoča izbiro celoštevilskih vrednosti od 0 do 23
4. Upravna enota dogodka nesreče ponuja na izbiro množico občin – LOOB
    - 5501 $\Longleftrightarrow$ Ajdovščina
    - 5502 $\Longleftrightarrow$ Brežice
    - 5503 $\Longleftrightarrow$ Celje
    - 5504 $\Longleftrightarrow$ Cerknica
    - 5505 $\Longleftrightarrow$ Črnomelj
    - 5506 $\Longleftrightarrow$ Domžale
    - 5507 $\Longleftrightarrow$ Dravograd
    - 5508 $\Longleftrightarrow$ Gornja Radgona
    - 5509 $\Longleftrightarrow$ Grosuplje
    - 5510 $\Longleftrightarrow$ Hrastnik
    - 5511 $\Longleftrightarrow$ Idrija
    - 5512 $\Longleftrightarrow$ Ilirska Bistrica
    - 5513 $\Longleftrightarrow$ Izola
    - 5514 $\Longleftrightarrow$ Jesenice
    - 5515 $\Longleftrightarrow$ Kamnik
    - 5516 $\Longleftrightarrow$ Kočevje
    - 5517 $\Longleftrightarrow$ Koper
    - 5518 $\Longleftrightarrow$ Kranj
    - 5519 $\Longleftrightarrow$ Krško
    - 5520 $\Longleftrightarrow$ Laško
    - 5521 $\Longleftrightarrow$ Lenart
    - 5522 $\Longleftrightarrow$ Lendava
    - 5523 $\Longleftrightarrow$ Litija
    - 5524 $\Longleftrightarrow$ Ljubljana
    - 5525 $\Longleftrightarrow$ Ljubljana Center

- 5526 $\Longleftrightarrow$ Ljubljana Moste Polje
- 5527 $\Longleftrightarrow$ Ljubljana Šiška
- 5528 $\Longleftrightarrow$ Ljubljana Vič Rudnik
- 5529 $\Longleftrightarrow$ Ljutomer
- 5530 $\Longleftrightarrow$ Logatec
- 5531 $\Longleftrightarrow$ Maribor
- 5534 $\Longleftrightarrow$ Metlika
- 5535 $\Longleftrightarrow$ Mozirje
- 5536 $\Longleftrightarrow$ Murska Sobota
- 5537 $\Longleftrightarrow$ Nova Gorica
- 5538 $\Longleftrightarrow$ Novo Mesto
- 5539 $\Longleftrightarrow$ Ormož
- 5540 $\Longleftrightarrow$ Piran
- 5541 $\Longleftrightarrow$ Postojna
- 5542 $\Longleftrightarrow$ Ptuj
- 5543 $\Longleftrightarrow$ Radlje ob Dravi
- 5544 $\Longleftrightarrow$ Radovljica
- 5545 $\Longleftrightarrow$ Ravne na Koroškem
- 5546 $\Longleftrightarrow$ Ribnica
- 5547 $\Longleftrightarrow$ Sevnica
- 5548 $\Longleftrightarrow$ Sežana
- 5549 $\Longleftrightarrow$ Slovenj Gradec
- 5550 $\Longleftrightarrow$ Slovenska Bistrica
- 5551 $\Longleftrightarrow$ Slovenske Konjice
- 5552 $\Longleftrightarrow$ Šentjur pri Celju
- 5553 $\Longleftrightarrow$ Škofja Loka
- 5554 $\Longleftrightarrow$ Šmarje pri Jelšah
- 5555 $\Longleftrightarrow$ Tolmin
- 5556 $\Longleftrightarrow$ Trbovlje
- 5557 $\Longleftrightarrow$ Trebnje
- 5558 $\Longleftrightarrow$ Tržič
- 5559 $\Longleftrightarrow$ Velenje
- 5560 $\Longleftrightarrow$ Vrhnika
- 5561 $\Longleftrightarrow$ Zagorje ob Savi
- 5562 $\Longleftrightarrow$ Žalec
- 5564 $\Longleftrightarrow$ Maribor
- 5565 $\Longleftrightarrow$ Pesnica
- 5566 $\Longleftrightarrow$ Maribor – Pobrežje
- 5567 $\Longleftrightarrow$ Maribor – Rotovž
- 5568 $\Longleftrightarrow$ Ruše
- 5569 $\Longleftrightarrow$ Maribor – Tabor
- 5570 $\Longleftrightarrow$ Maribor – Tezno
- 5580 $\Longleftrightarrow$ Izdano v drugi državi
- 5598 $\Longleftrightarrow$ Mnz
- 5599 $\Longleftrightarrow$ Neznana občina

5. Prizorišče nesreče

   - 1 $\Longleftrightarrow$ v naselju
   - 0 $\Longleftrightarrow$ izven naselja

6. Kategorija ceste – LOVC

   - 0 $\Longleftrightarrow$ avtocesta
   - A $\Longleftrightarrow$ avtocesta – stara
   - H $\Longleftrightarrow$ hitra cesta
   - L $\Longleftrightarrow$ lokalna cesta
   - M $\Longleftrightarrow$ magistralna cesta – stara
   - N $\Longleftrightarrow$ naselje z uličnim sistemom
   - R $\Longleftrightarrow$ regionalna cesta – stara
   - T $\Longleftrightarrow$ turistična cesta
   - V $\Longleftrightarrow$ naselje brez uličnega sistema
   - 1 $\Longleftrightarrow$ glavna cesta I. reda
   - 2 $\Longleftrightarrow$ glavna cesta II. reda
   - 3 $\Longleftrightarrow$ regionalna cesta I. reda
   - 4 $\Longleftrightarrow$ regionalna cesta II. reda
   - 5 $\Longleftrightarrow$ regionalna cesta III. reda

7. Opis kraja dogodka – PRKD

   - Ž $\Longleftrightarrow$ železniški prehod
   - A $\Longleftrightarrow$ avtobusna postaja
   - C $\Longleftrightarrow$ cesta
   - E $\Longleftrightarrow$ železniško postajališče
   - K $\Longleftrightarrow$ kolesarska steza ali pločnik
   - M $\Longleftrightarrow$ krožno križišče
   - N $\Longleftrightarrow$ naravno okolje
   - O $\Longleftrightarrow$ naravovarstveno območje
   - P $\Longleftrightarrow$ parkirni prostor
   - R $\Longleftrightarrow$ križišče
   - T $\Longleftrightarrow$ predor
   - V $\Longleftrightarrow$ vlak
   - Z $\Longleftrightarrow$ prehod za pešce

8. Poškodba osebe in klasifikacija nesreče – PRPO

   - B $\Longleftrightarrow$ brez poškodbe
   - U $\Longleftrightarrow$ brez poškodbe – UZ
   - P $\Longleftrightarrow$ sled poškodbe
   - L $\Longleftrightarrow$ lažja telesna poškodba
   - H $\Longleftrightarrow$ hujša telesna poškodba
   - S $\Longleftrightarrow$ smrt

9. Stanje vozišča v času nesreče – PRPV

    - BL $\Longleftrightarrow$ blatno
    - MO $\Longleftrightarrow$ mokro
    - OS $\Longleftrightarrow$ ostalo
    - PN $\Longleftrightarrow$ poledenelo – neposipano
    - PP $\Longleftrightarrow$ poledenelo – posipano
    - SL $\Longleftrightarrow$ snežno – pluženo
    - SN $\Longleftrightarrow$ snežno – nepluženo
    - SP $\Longleftrightarrow$ spolzko
    - SU $\Longleftrightarrow$ suho

10. Stanje prometa v času nesreče – PRSP

    - E $\Longleftrightarrow$ neznano
    - G $\Longleftrightarrow$ gost
    - N $\Longleftrightarrow$ normalen
    - R $\Longleftrightarrow$ redek
    - Z $\Longleftrightarrow$ zastoji

11. Vrsta vozišča – PRSV

    - AH $\Longleftrightarrow$ hrapav asfalt / beton
    - AN $\Longleftrightarrow$ nevaren asfalt / beton
    - AZ $\Longleftrightarrow$ zglajen asfalt / beton
    - MA $\Longleftrightarrow$ makadam
    - OS $\Longleftrightarrow$ ostalo

12. Tip prometne nesreče – PRTN

    - ČT $\Longleftrightarrow$ čelno trčenje
    - BT $\Longleftrightarrow$ bočno trčenje
    - NT $\Longleftrightarrow$ naletno trčenje
    - OP $\Longleftrightarrow$ oplaženje
    - OS $\Longleftrightarrow$ ostalo
    - PP $\Longleftrightarrow$ povoženje pešca
    - PR $\Longleftrightarrow$ prevrnitev vozila
    - PZ $\Longleftrightarrow$ povoženje živali
    - TO $\Longleftrightarrow$ trčenje v objekt
    - TV $\Longleftrightarrow$ trčenje v stoječe / parkirano vozilo

13. Vremenske okoliščine v času nesreče – PRVR

    - D $\Longleftrightarrow$ deževno
    - J $\Longleftrightarrow$ jasno
    - M $\Longleftrightarrow$ megla
    - N $\Longleftrightarrow$ neznano
    - O $\Longleftrightarrow$ oblačno
    - S $\Longleftrightarrow$ sneg

- T $\Longleftrightarrow$ toča
- V $\Longleftrightarrow$ veter

14. Vzrok prometne nesreč – PRVZ

- CE $\Longleftrightarrow$ nepravilnosti na cesti
- HI $\Longleftrightarrow$ neprilagojena hitrost
- NP $\Longleftrightarrow$ nepravilnosti pešca
- OS $\Longleftrightarrow$ ostalo
- PD $\Longleftrightarrow$ neupoštevanje pravil o prednosti
- PR $\Longleftrightarrow$ nepravilno prehitevanje
- PV $\Longleftrightarrow$ premiki z vozilom
- SV $\Longleftrightarrow$ nepravilna stran / smer vožnje
- TO $\Longleftrightarrow$ nepravilnosti na tovoru
- VO $\Longleftrightarrow$ nepravilnosti na vozilu
- VR $\Longleftrightarrow$ neustrezna varnostna razdalja