Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Janez Čadež

Izdelava hibridnih mobilnih aplikacij z ogrodjem Ionic

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Aleš Smrdel

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Izdelava hibridnih mobilnih aplikacij z ogrodjem Ionic (*ang. Creating hybrid mobile applications with Ionic framework*).

Tematika naloge:

Mobilne naprave postajajo vse pomembnejši segment računalništva, s tem pa se razvija tudi segment mobilnih aplikacij. Pri razvoju mobilnih aplikacij lahko izbiramo med različnimi vrstami aplikacij: domorodnimi, spletnimi in hibridnimi.

Vaša naloga v okviru diplome je, da najprej raziščete in opišite različne vrste aplikacij ter naštejte prednosti oziroma slabosti, ki jih prinašajo različne vrste aplikacij, v nadaljevanju pa nato tudi razvijete aplikacijo za sledenje časa. Na področju mobilnih naprav pa obstaja tudi več različnih platform, zato je zaželeno, da razvijete aplikacijo, ki jo bo mogoče izvajati na različnih platformah.

V ta namen raziščite različne tehnologije, ki se lahko uporabljajo pri razvoju hibridnih aplikacij, in izberite eno izmed teh tehnologij, s pomočjo katere boste razvili hibridno mobilno aplikacijo. Na primeru razvoja te aplikacije pa tudi predstavite korake, ki so potrebni za razvoj hibridne aplikacije, predstavite pa tudi korake, ki so potrebni, zato da lahko tovrstno aplikacijo prevedemo za vsaj dve mobilni platformi in jo na teh platformah tudi namestimo in izvedemo.

Zahvaljujem se mentorju doc. dr. Alešu Smrdelu za pomoč, nasvete in odzivnost pri izdelavi diplomskega dela. Zahvaljujem se tudi svojim sodelavcem iz podjetja Špica International sistemi za avtomatsko identifikacijo d.o.o., družini, prijateljem in vsem, ki so me podpirali na moji študijski poti.

Svojim domačim.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
SaaS	Software as a Service	programska oprema kot storitev
NPM	Node Packet Manager	upravnik NodeJS paketov
CLI	Command-line Interface	ukazni vmesnik
API	Application Programming Interface	aplikacijski programski vmesnik
REST	Representational State Transfer	predstavitveni prenos stanja
JSON	JavaScript Object Notation	objektna notacija JavaScript
HTTP	HyperText Transfer Protocol	protokol za prenos hiperteksta
UTC	Coordinated Universal Time	univerzalni koordinirani čas
OAuth	Open Standard for Authorization	odprti standard za avtorizacijo
SDK	Software Development Kit	zbirka orodij za razvoj
USB	Universal Serial Bus	univerzalno serijsko vodilo
SEO	Search Engine Optimization	optimizacija iskalnika
AoT	Ahead of Time Compilation	predčasno prevajanje

Povzetek

Naslov: Izdelava hibridnih mobilnih aplikacij z ogrodjem Ionic

Avtor: Janez Čadež

Glavni cilj diplomske naloge je predstaviti hibridne mobilne aplikacije in prikazati postopek razvoja skozi izdelavo konkretnega primera hibridne mobilne aplikacije. Hibridne mobilne aplikacije so aplikacije za mobilne naprave, narejene s pomočjo spletnih tehnologij, ki za razliko od spletnih aplikacij omogočajo poln dostop do funkcionalnosti naprave. V diplomi najprej predstavimo njihove prednosti in slabosti, nato pa opišemo postopek razvoja na primeru takšne aplikacije. Pri razvoju aplikacije se srečamo z ogrodji, kot sta Ionic in Angular, ki sta tudi predstavljena v diplomi, predstavljena pa so še druga orodja, ki nam poenostavijo razvoj hibridnih aplikacij. Pri razvoju aplikacije predstavimo vse korake, od načrtovanja, implementacije funkcionalnosti, povezave z zaledjem pa do uporabe vtičnikov za dostop do mobilne naprave. Na koncu predstavimo še namestitev same Ionic aplikacije na dve različni platformi, to sta iOS in Android. Pri uporabi vtičnikov in namestitvi na posamezne platforme smo uporabili Apache Cordovo, ki nam omogoča generiranje aplikacijskega paketa in dostop do funkcionalnosti naprave preko vtičnikov.

Ključne besede: hibridna mobilna aplikacija, Ionic, Angular, Apache Cordova, Ionic Native.

Abstract

Title: Creating hybrid mobile applications with Ionic framework

Author: Janez Čadež

The main objective of the thesis is to present hybrid mobile applications and show the process of development through building sample application. Hybrid mobile applications are application for mobile devices built with web technologies but unlike web applications, hybrid applications allow full access to device functionality. We first describe advantages and disadvantages of hybrid mobile applications and the process of development. When developing hybrid mobile applications, we encounter know frameworks like Ionic and Angular, that are also presented in the thesis, and which make the development easier. We present all the development steps, from planning, functionality implementation, API calls and plugins usage. At the end, we present how to use Ionic to deploy our application to iOS and Android platforms. For the plugins and deployment we used technology called Apache Cordova, which helps us to generate application packages and access native functionality of device.

Keywords: hybrid mobile application, Ionic, Angular, Apache Cordova, Ionic Native.

Poglavje 1

Uvod

S hitrim razvojem spleta in povezanih tehnologij, je tudi izdelava mobilnih aplikacij z uporabo spletnih tehnologij, tako imenovanih hibridnih aplikacij, vedno bolj mikavna izbira. V preteklosti so namreč imeli razvijalci precej težav pri razvoju hibridnih aplikacij, saj tehnologija še ni bila kos zahtevam, kar pomeni, da je bilo treba biti previden pri izbiri le-teh, saj so take aplikacije imele dostikrat precejšnje performančne težave. Z razvojem spletnih vsebovalnikov, kjer se te aplikacije izvajajo in samih mobilnih naprav, prihaja do tega, da večina uporabnikov ne opazi več razlike med hibridno in domorodno (*ang. native*) aplikacijo. Razvoj pa je posledično vplival tudi na zelo hitro delovanje hibridnih aplikacij.

Motivacija pri izdelavi hibridnih mobilnih aplikacij se pokaže npr. na realnem primeru, in sicer, če imamo v našem podjetju že zaposlene spletne razvijalce, nam za izdelavo mobilne aplikacije ne bo potrebno iskati novih kadrov, ali izobraževati starih, temveč lahko uporabimo obstoječa znanja. To predstavlja velik plus predvsem za manjša podjetja, ki nimajo sredstev za zaposlitev različnih razvijalcev za posamezne mobilne platforme.

1.1 Cilji diplomske naloge

Prvi cilj diplomske naloge je predstaviti hibridne aplikacije ter opisati njihove prednosti in slabosti. Drugi cilj pa je razviti dejansko hibridno mobilno aplikacijo in na podlagi te aplikacije prikazati in opisati izdelavo hibridne aplikacije.

V prvem poglavju sta opisana uvod in cilji diplomskega dela. Sledi predstavitev hibridnih aplikacij, njihove prednosti in slabosti, opis Ionic in Angular ogrodij, primerjava konkurenčnih ogrodij Ionic in React Native ter predstavitev ogrodij in orodij za razvoj teh aplikacij.

V tretjem, glavnem poglavju, predstavimo razvoj hibridne aplikacije v podjetju. V tem poglavju opišemo vse faze razvoja, vse od razdelitve dela, načrtovanja razvoja aplikacije, razvoja funkcionalnosti aplikacije, povezave z zaledjem, uporabo knjižnice za manipulacijo časov, do uporabe vtičnikov za dostop do naprave.

V četrtem poglavju opišemo namestitev izdelane aplikacije na dve različni platformi, iOS in Android. Na tem mestu opišemo tehnologijo Apache Cordova, ki jo uporabljamo za dostop do funkcionalnosti naprave in generiranje namestitvenega paketa za posamezne mobilne platforme.

V zadnjem, petem poglavju pa povzamemo diplomsko delo, predstavimo nekaj sklepnih ugotovitev in podamo predloge za nadaljnje delo.

Poglavje 2

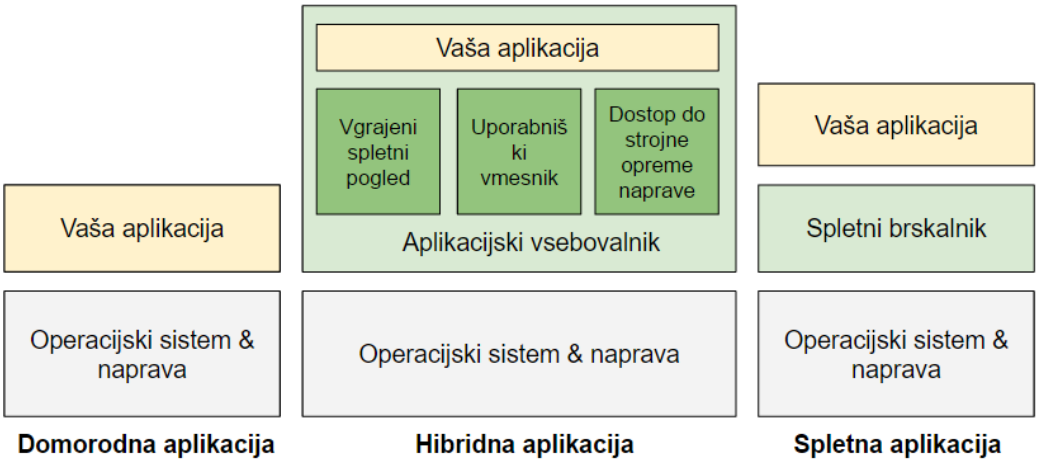
Hibridne mobilne aplikacije

V tem poglavju predstavljamo hibridne mobilne aplikacije in tehnologije za njihovo izdelavo.

2.1 Kaj so hibridne mobilne aplikacije

Hibridne mobilne aplikacije so zelo podobne ostalim mobilnim aplikacijam, ki jih lahko prenesemo iz trgovine aplikacij (Google Play Store, Apple App Store). Z njimi lahko dostopamo do podatkov spletnega strežnika, uporabljamo funkcionalnosti naprave ali pa npr. delimo vsebino aplikacije preko socialnih omrežij. Hibridne mobilne aplikacije so narejene s pomočjo spletnih tehnologij (HTML, CSS, JavaScript) in ogrodij, ki olajšajo delo z JavaScript jezikom.

Za razliko od spletnih aplikacij, ki se izvajajo na spletnem strežniku, se hibridne aplikacije izvajajo znotraj domorodne aplikacije, ki uporablja tehnologijo, ki se imenuje *WebView*. *WebView* si lahko predstavljamo kot poenostavljen brskalnik v aplikaciji, ki nam v povezavi s tehnologijo Apache Cordova omogoča tudi dostop do različnih (tehničnih) lastnosti naprave, kot so npr.: kamera, imenik ali pa merilnik pospeška. Te funkcionalnosti nam pogosto niso na voljo v spletnih aplikacijah oz. so precej omejene s strani proizvajalcev brskalnikov, poleg tega pa se lahko funkcionalnosti med posa-



Slika 2.1: Arhitektura mobilnih aplikacij [2].

meznimi brskalniki tudi razlikujejo.

Domorodne (*ang. native*) aplikacije se namreč razlikujejo od hibridnih po tem, da direktno dostopajo do funkcionalnosti naprave in uporabljajo API operacijskega sistema, zaradi česar so performančno najzmogljivejše. Slika 2.1 prikazuje razlike med arhitekturo domorodnih, hibridnih in spletnih mobilnih aplikacij.

2.2 Prednosti in slabosti hibridnih mobilnih aplikacij

Glavna motivacija pri razvoju hibridnih aplikacij je, da lahko uporabimo obstoječe znanje razvoja spletnih aplikacij za razvoj večplatformnih mobilnih aplikacij. Povedano drugače, hibridne mobilne aplikacije razvijamo v tehnologijah, kot so HTML, CSS in JavaScript, nato pa razvito aplikacijo prevedemo na različne platforme, kot so iOS, Android in Windows. Prav tako pa je velika verjetnost podpore s strani vodstva podjetja, saj se lahko obstoječi razvijalci spletnih aplikacij v podjetju usmerijo tudi v razvoj mo-

bilnih aplikacij, s čimer se zmanjšajo stroški poslovanja. Razvoj hibridnih aplikacij poteka zelo hitro, zato včasih tak način razvoja uporabljamo tudi za izdelovanje prototipov mobilnih aplikacij.

Ker pa z uporabo *WebView*-a uporabljamo še en nivo abstrakcije pri razvoju, nam to prinese nekaj performančnih pomanjkljivosti. To pomeni, da če razvijamo npr. igre oz. aplikacije, ki morajo biti zelo zmogljive, hibridne aplikacije niso optimalna izbira. Pri uporabi vtičnikov za dostop do funkcionalnosti naprave v naši kodi, je razvijanje aplikacije zamudnejše, saj ti vtičniki ne delujejo v spletni aplikaciji med razvijanjem aplikacije. To pomeni, da moramo našo hibridno aplikacijo ob vsaki spremembi pognati tudi na napravi, da stestiramo spremembo.

2.3 Ionic

V podjetju smo se odločili za razvoj nove različice mobilne aplikacije za sledenje časa z imenom **My Hours**. Aplikacija nam omogoča sledenje časa na projektih in opravilih ter možnost analize porabe časa za izboljšanje produktivnosti. Za razvoj aplikacije smo izbrali ogrodje Ionic, ki je odprtokodno ogrodje za razvoj hibridnih aplikacij. Logotip ogrodja je prikazan na sliki 2.2.

Ionic je brezplačno in odprtokodno ogrodje, ki ga vzdržuje podjetje Drifty Co [15]. Podjetje je bilo ustanovljeno leta 2012. V tistem času je bila uporaba spletnih tehnologij za izdelavo hibridnih mobilnih aplikacij še precej v povojih. Sam produkt Ionic je njihov najbolj znan izdelek in je tudi zelo popularen na platformi za odprtokodne projekte Github. Ionic je razdeljen v dva dela. Prvi del je samo ogrodje, medtem ko drugi del sestavljajo dodatne funkcionalnosti, ki so jih poimenovali Ionic Services. Oba dela Ionica imata zelo dobro dokumentacijo z mnogimi primeri aplikacij.

Ionic podpira tri platforme (iOS, Android in Windows), ima zelo dobro dokumentacijo in veliko število vtičnikov za dostop do funkcionalnosti naprave (Ionic Native). Ena izmed poglavitnih prednosti Ionica je, da z upo-

Slika 2.2: Logotip ogrodja Ionic.

rabo njihovih komponent (npr. komponente za izbiro datuma) ogrodje samodejno prevede stil v obliko prilagojeno glede na platformo. V primeru, da razvijamo za Android, nam Ionic samodejno generira komponente, ki sledijo oblikovalskim načelom Material Design, ki so priporočena za to platformo. Ko v ogrodju dodajamo platforme, pa uporabljamo Apache Cordovo, ki nam omogoča dostop do vmesnika naprave.

Proces izdelave Ionic mobilnih aplikacij je dokaj preprost. Vse, kar potrebujemo, je Ionic ukazna vrstica, s katero kreiramo nov projekt. S to ukazno vrstico tudi strežemo, testiramo in razvijemo aplikacijo na različne platforme. Hkrati nam omogoča funkcionalnosti živega osveževanja (*ang. live reload*), ki nam zelo pohitri razvoj, saj se brskalnik avtomatsko odzove na spremembe kode.

2.4 Angular

Za razvoj Ionic aplikacij ne uporabljamo neposredno JavaScript programskega jezika, ampak uporabljamo abstrakcijo (ogrodje) imenovano Angular. Angular je eden najpopularnejših JavaScript ogrodij za razvoj spletnih aplikacij [11].

Na vseh večjih projektih v jeziku JavaScript se podjetja odločijo za uporabo ogrodij oz. knjižnic in ne povsem čiste JavaScript kode. Knjižnice oz. ogrodja nam dodajo strukturo v ta programski jezik, omogočajo ustvarjanje predlog, modularni razvoj in predvsem čistejšo in berljivo kodo.

Slika 2.3: Logotip ogrodja Angular [12].

Angular, ki smo ga uporabili pri razvoju Ionic aplikacije je druga verzija tega JavaScript ogrodja in uvaja veliko izboljšav v primerjavi s prejšnjo različico, AngularJS. Ker Angular uporablja razširitev JavaScript jezika TypeScript, s tem doda dinamičnemu jeziku statične tipe, kar izjemno izboljša izkušnjo razvoja aplikacij. Za razliko od knjižnic oz. ogrodij, kot so npr. React, Vue oz. jQuery, je Angular popolno MVC (*ang. Model-View-Controller*) ogrodje in nam ponuja celoten nabor funkcionalnosti, kot so: dostop do zaledja, generiranje predlog, lokalizacija, dostopnost in še mnogo drugih. Za primerjavo pa knjižnica React ponuja samo pogledni del (*ang. view*).

Kot zanimivost lahko omenimo, da se uporablja semantično verzioniranje in zato napišemo samo Angular, ko govorimo o drugi različici tega ogrodja. Semantično verzioniranje razdeli verzijo ogrodja na večje spremembe, ki povzročijo nekompatibilnost in manjše spremembe, ki ne povzročijo nekompatibilnosti ter na popravke. Zaradi zgoraj omenjenih dejstev, se številka glavne verzije Angular ogrodja povečuje (pribl.) vsakih 6 mesecev, če navedemo

primer, v mesecu marcu (2017) je bila izdana različica 4, ki je nazaj kompatibilna z verzijo 2.

Angular nam prinaša veliko izboljšav v primerjavi s prejšnjo različico, kot je npr. lepša in čistejša koda, hitrejše delovanje v primeru velikih zapisov podatkov in možnost optimizacije kode za delovanje na več platformah. Ena od glavnih prednosti Angularja pred drugimi ogrodji oz. knjižnicami pa je, da je celovito ogrodje in nam ponuja tudi orodje za ukazno vrstico, ki nam poenostavi učni proces.

Ogrodje Angular je zelo optimizirano za hitrost in zmogljivost. Ogrodje nam pretvori naše predloge v kodo, ki je visoko optimizirana za današnje JavaScript navidezne stroje (*ang. code generation*). Ker želimo čim bolj skrajšati čas nalaganja aplikacij pri uporabnikih, pa se lahko poslužujemo tehnologije, ki jo v Angular svetu imenujejo *Angular Universal*. Ta nam omogoča, da našo aplikacijo strežemo iz spletnega strežnika le v HTML in CSS tehnologijah. Ker je naša aplikacija pred renderirana na strežniku in nato poslana na odjemalca, s tem skrajšamo čas nalaganja aplikacije in izboljšamo SEO specifikacije. Zadnja od lastnosti hitre Angular aplikacije pa je uporaba t. i. predčasnega procesa prevajanja (*ang. Ahead of Time Compilation*). To nam omogoča, da našo aplikacijo prevedemo v procesu kreacije in ne v procesu poganjanja (*ang. runtime*). S tem pridobimo na času izvajanja in velikosti aplikacije, saj ne vsebuje prevajalnika (*ang. compiler*), ki obsega približno polovico samega ogrodja Angular.

2.5 Primerjava ogrodij Ionic in React Native

Ogrodji Ionic in React Native sta dve najbolj popularni izbiri za razvoj mobilnih aplikacij z uporabo spletnih tehnologij, kot so HTML, JavaScript in CSS. Za razliko od ogrodja Ionic, React Native prevede spletno kodo v domorodne komponente in ne uporablja hibridne arhitekture.

To pomeni, da je uporabniška izkušnja boljša, ker domorodne komponente sledijo načelom izdanim za določen mobilni operacijski sistem (npr.

Android). Druga prednost te tehnologije je, da nam omogoča boljše performančne zmožnosti in boljše animacije. Slaba stran uporabe ogrodja React Native pa je, da niso vse domorodne komponente sistema na voljo, poleg tega pa je koda napisana v tem ogrodju je bolj specifična glede na platformo, kot v ogrodju Ionic.

React Native uporablja knjižnico React in JavaScript kodo za predloge, ki jo imenujemo JSX. To nam predstavlja strmejšo krivuljo učenja, sploh če uporabljamo druga ogrodja, kot npr. jQuery oz. Angular [16].

Obe ogrodji, Ionic in React Native, sta dobri izbiri, če hočemo kot spletni programer razviti mobilne aplikacije. Če že uporabljamo React knjižnico, bi izbrali React Native, v primeru uporabe Angular oz. drugega ogrodja, pa bi izbrali Ionic.

V ekipi za razvoj aplikacije **My Hours** smo se odločili za uporabo ogrodja Ionic zaradi dveh večjih razlogov. Prvi razlog je bil, da je bila obstoječa mobilna aplikacija že razvita v starejši različici Ionic ogrodja, tako da so bili programerji v ekipi vešči razvoja hibridnih mobilnih aplikacij. Poleg tega smo v podjetju razvijali tudi nekaj drugih produktov s pomočjo Angular ogrodja, zato je bila izbira Ionica najprimernejša.

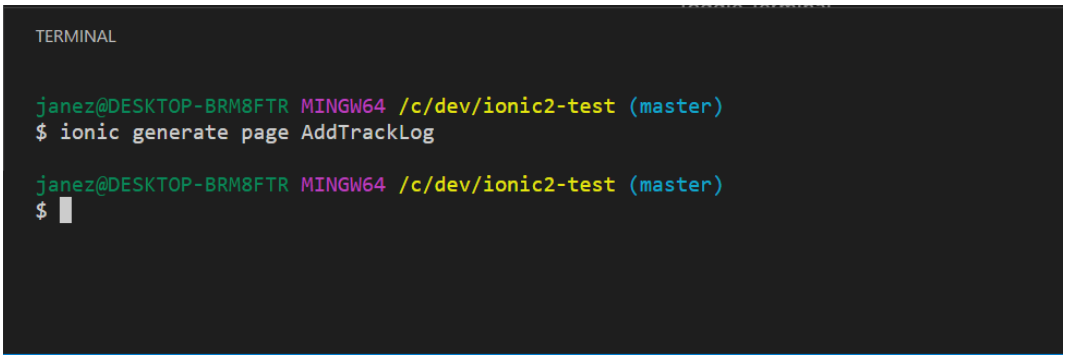
Med uporabo Ionic ogrodja nam je bila v veliko pomoč dokumentacija podjetja, primeri uporabe komponent in forum. Ker je ogrodje odprtokodno, lahko vsakdo sodeluje pri poročanju o napakah oz. pri samem razvoju ogrodja.

2.6 Orodja za razvoj Ionic aplikacij

Ionic je razvit v programskem jeziku JavaScript in ga namestimo prek NPM (NPM - Node Packet Manager), zato najprej potrebujemo nameščeno platformo NodeJS. Ko ga namestimo iz NPM repozitorija, dobimo orodje ionic-cli oz. orodje za ukazno vrstico, ki omogoča lažje generiranje in razvijanje Ionic projektov.

Ionic ukazna vrstica (CLI - Command-line Interface) nam omogoča pri-

pravo novega projekta, generiranje novih funkcionalnosti aplikacije (strani, servisi, moduli) in vse ukaze za delo s hibridnimi aplikacijami. Slika 2.4 prikazuje primer, kjer smo z uporabo Ionic ukazne vrstice generirali novo stran aplikacije imenovano **AddTrackLog**. Orodje nam je zgeneriralo tri datoteke, *AddTrackLog.html*, *AddTrackLog.ts* in *AddTrackLog.scss*.

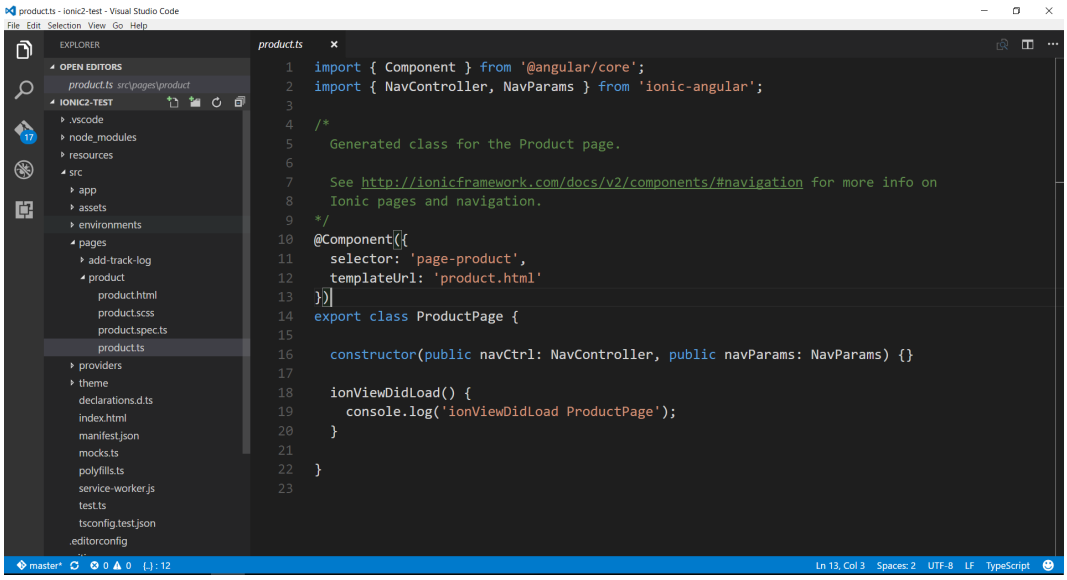


Slika 2.4: Generiranje nove strani z Ionic CLI.

Poleg samega ogrodja in orodja za ukazno vrstico, nam Ionic platforma ponuja še dodatne funkcionalnosti (nekaj teh funkcionalnosti je tudi plačljivih). To so npr. Ionic View (aplikacija za poganjanje naših hibridnih aplikacij na napravah), Ionic Services (potisna sporočila, avtentikacijska storitev ...) ali pa Ionic Creator - grafični vmesnik za hitro prototipiranje Ionic aplikacij.

Za razvoj Ionic hibridnih aplikacij potrebujemo še urejevalnik teksta. Takih urejevalnikov je mnogo, v podjetju smo se odločili za Visual Studio Code (slika 2.5). Kot zanimivost lahko omenimo, da je tudi ta urejevalnik hibridna aplikacija, saj je narejen s pomočjo spletnih tehnologij in z ogrodjem za izdelavo hibridnih aplikacij Electron.

V urejevalniku Visual Studio Code spreminjamo datoteke s končnico .ts, pri tem pa nam Ionic CLI avtomatsko generira JavaScript datoteke, jih streže na spletnem strežniku in nam osvežuje spletno stran. To nam omogoča zelo hitro razvijanje.



Slika 2.5: Urejevalnik Visual Studio Code.

Poglavje 3

Razvoj aplikacije

Aplikacija **My Hours** je orodje za sledenje časa. Za izboljšanje preglednosti lahko svoje sledenje časa v aplikaciji razdelimo na projekte in opravila. Projekti so lahko neodvisni oz. povezani s klientom, za katerega opravljamo določeno delo in želimo slediti svojemu opravljenemu času, npr. kdaj je bil opravljen čas oz. kje. Opravila pa predstavljajo dela, kot so npr. telefonski klici ali pa odgovarjanje na elektronsko pošto.

S sledenjem časa torej ugotovimo, na katerih opravilih ga porabimo največ. Na podlagi teh ugotovitev lahko optimiziramo svoj delovni čas in izboljšamo produktivnost sebe oz. svoje ekipe. Orodje **My Hours** nam služi tudi kot boljša alternativa za sledenje delovnega časa v preglednicah Excel, saj deluje kot spletna aplikacija (dostopna na vseh napravah preko spletnega brskalnika oz. mobilne aplikacije).

Glavne prednosti rešitve **My Hours** v primerjavi s tistimi, ki jih ponujajo konkurenti, so: preprosto delovanje (minimalistični uporabniški vmesnik), dobra orodja za generiranje poročil o porabi časa in dostopna cena. Sama rešitev deluje na način SaaS (SaaS - Software as a Service), kar omogoča hitre iteracije razvoja in zanesljivo delovanje.

3.1 Razdelitev dela

Aplikacijo **My Hours** smo razvijali v ekipi razvoja v podjetju. Ekipa razvoja se deli na ekipo za razvoj glavnega produkta **Time&Space**, ekipo za razvoj strojne opreme in ekipo za razvoj t. i. oblačnih aplikacij (*ang. Cloud Applications*). V ekipi za razvoj oblačnih aplikacij je manjša ekipa za razvoj storitve **My Hours**. Sam sem večino časa delal na obstoječem produktu **My Hours**, zato sem bil tudi izbran za delo na novi mobilni aplikaciji **My Hours**. V ekipi za oblačne aplikacije so že imeli izkušnje z razvojem hibridnih mobilnih aplikacij, s pomočjo uporabe Ionic ogrodja, kar je bil velik plus za lažji razvoj **My Hours** aplikacije.

Delo je potekalo v agilnem načinu razvoja programske opreme, imenovanem Scrum. Scrum je iterativni in inkrementalni način razvoja produktov. Ta način definira bolj fleksibilen in celovit pristop k razvoju produkta. Glavna značilnost tega način razvoje je, da ekipa razume, da se lahko želje stranke med samim razvojem spremenijo in je dovolj fleksibilna, da se prilagodi njihovim željam.

Scrum način razvoja je sestavljen iz seznama nalog (*ang. sprint backlog*), obdobja razvoja (*ang. sprint*) in uporabniških zgodb (*ang. user stories*) [14]. Vsako obdobje razvoja je v našem primeru trajalo 14 dni. Pred začetkom dela v tem obdobju razvoja smo pregledali seznam nalog in napolnili to obdobje razvoja. Poleg sledenja razvoja produkta glede na sprinte pa smo v ekipi razvoja prakticirali še t. i. stoječe sestanke (*ang. standup*), vsak dan zjutraj, kjer je, po vnaprej določenem intervalu, vsak zaposlen v ekipi povedal, s čim se bo ukvarjal ta dan, in če je zaključil naloge od prejšnjega dneva.

Kot član ekipe sem bil odgovoren za prototipiranje, načrtovanje in razvoj nove mobilne aplikacije. V primeru nejasnosti sem poprosil sodelavce za t. i. razvoj v paru (*ang. pair programming*), ki izboljša produktivnost razvoja. Po implementirani funkcionalnosti aplikacije sem sodelavce z izkušnjami razvoja teh aplikacij povprašal tudi za pregled kode (*ang. code review*).

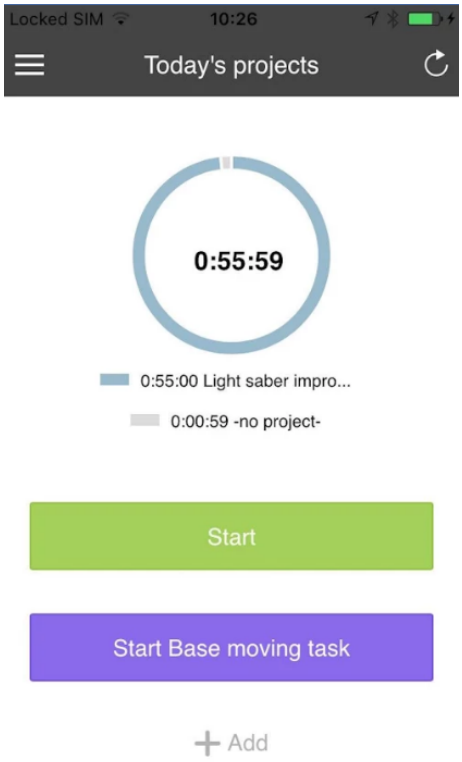
3.2 Načrtovanje razvoja aplikacije

Razvoj mobilne aplikacije v podjetju se je začel s fazo načrtovanja. Pri načrtovanju nove aplikacije smo se zgledovali po obstoječi aplikaciji **My Hours** za sledenje časa, pri tem pa smo razmišljali, kako bi lahko obstoječo aplikacijo izboljšali z vidika delovanja in funkcionalnosti. Tudi obstoječa aplikacija je bila narejena v ogrodju Ionic, le da v starejši verziji. Slika 3.1 prikazuje začetno stran obstoječe hibridne mobilne aplikacije **My Hours**. Skupaj smo se odločili, da bomo novo aplikacijo razvili čisto od začetka in ne bomo le posodobili obstoječe.

Velik del odločitve za razvoj nove verzije mobilne aplikacije **My Hours** je bil odziv (*ang. feedback*) uporabnikov prve različice aplikacije. Ti uporabniki so stopili v kontakt z nami preko trgovin aplikacij Google Play oz. App Store in elektronske pošte za podporo aplikacije **My Hours**. Aplikacija je bila narejena v prvi različici Ionic ogrodja in to je prineslo nekaj performančnih težav, ki so jih uporabniki tudi zaznali. Želeli so hitrejšo aplikacijo in več možnosti v sami aplikaciji.

Spletna različica aplikacije **My Hours** nam torej omogoča sledenje časa, urejanje projektov in vse druge napredne funkcije. Za razliko od spletne aplikacije pa je mobilna aplikacija podpirala le sledenje časa in pregled sledenja za določeno obdobje. Skupno smo se odločili, da v drugi verziji aplikacije najprej naredimo del za sledenje časa, ki je najpomembnejši del aplikacije, ter kasneje nadaljujemo in dodamo tudi funkcije za urejanje projektov. To bo omogočilo, da bosta spletna in mobilna aplikacija povsem neodvisni ena od druge.

V fazi načrtovanja mobilne aplikacije je zelo priporočena izdelava prototipa, pri katerem se vidi, kako naj bi ta aplikacija izgledala. Prototipiranje ima mnogo dobrih lastnosti, predvsem pa omogoča, da lahko predvidenim uporabnikom pokažemo prototip aplikacije, ki je v razvoju, uporabniki pa nam lahko na podlagi prototipa ponudijo povratne informacije še pred samim začetkom izdelave aplikacije. To nam prihrani čas in denar ter izboljša uporabniški vmesnik v primerjavi s produkti, ki gredo iz koncepta direktno



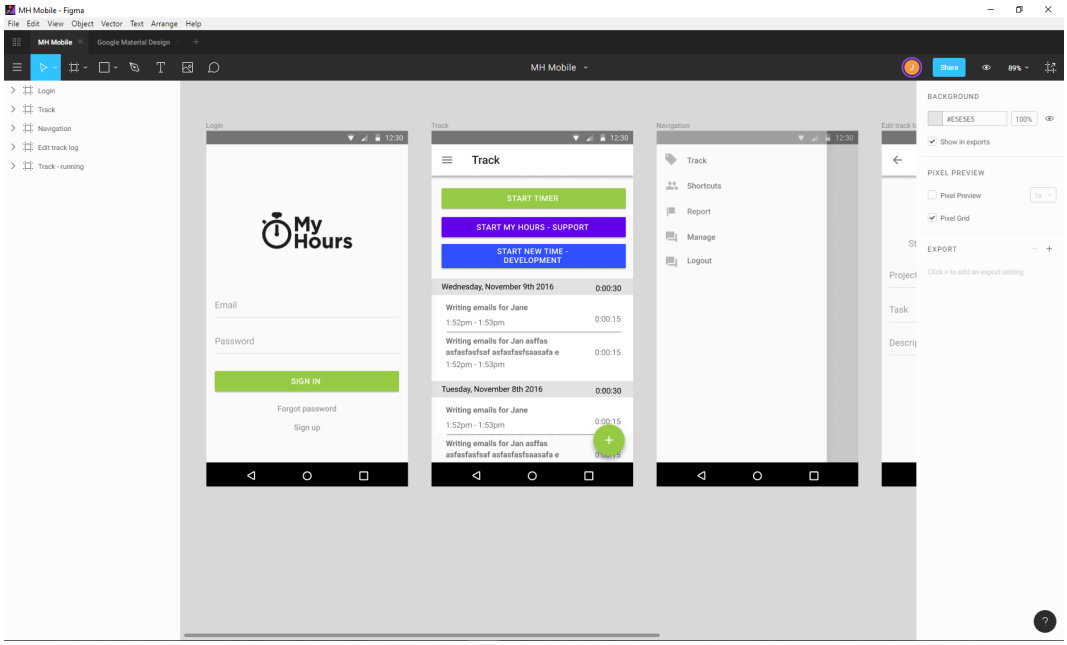
Slika 3.1: Obstoječa hibridna mobilna aplikacija **My Hours**.

v produkcijo.

Prvi korak prototipiranja je bilo načrtovanje pogledov (strani) aplikacije, z orodjem za izdelavo prototipov. Orodij za načrtovanje prototipov je mnogo, ponujajo pa nam vse od preprostega risanja do izbire različnih gradnikov, ki jih lahko med seboj tudi povežemo in s tem, dodamo občutek navigacije med stranmi v aplikaciji.

Izbrali smo orodje za prototipiranje Figma, ki že vsebuje gradnike, ki izgledajo identično kot elementi Material Design knjižnice za izgled na platformi Android. V tem orodju smo načrtovali najprej začetne zaslone (**Login**, **Signup**, **Forgot Password**), nato pa še druge bolj podrobne zaslone. Slika 3.2 prikazuje zaslonsko sliko orodja za prototipiranje Figma, na kateri lahko v sredini vidimo prototipe strani v aplikaciji. Na sliki so prikazane tri proto-

tipne strani v aplikaciji, in sicer: na levi strani je stran za vpis v aplikacijo, na sredini, glavna stran za sledenje časa, desno pa je prikazana navigaciji po aplikaciji s stranskim menijem.



Slika 3.2: Orodje za izdelavo prototipov Figma.

Na koncu prototipiranja smo dobili zelo realen videz prototipa aplikacije. V ekipi razvoja smo te prototipe pregledali, vsak od razvijalcev je podal svoje predloge in na podlagi teh predlogov smo prototipe tudi posodobili. Na koncu smo razviti prototip pokazali še projektnemu vodji na enem od sestankov in dodatno optimizirali prototip glede na povratne informacije projektnega vodje.

3.3 Funkcionalnost aplikacije

Ionic aplikacija je strukturirana po komponentah. Vsaka od funkcionalnosti aplikacije se nahaja v svoji komponenti, včasih pa je posamezna funkcionalnost razdeljena tudi na več komponent. Na primer: funkcionalnost sledenja

časa je razdeljena na glavno komponento (Track), ki je prikazana na sliki 3.3, ta pa vsebuje še poročilo časa (Track Report), in komponente za štartanje, dodajanje oz. urejanje časa (*StartTrackLog*, *AddTrackLog*, *EditTrackLog*).

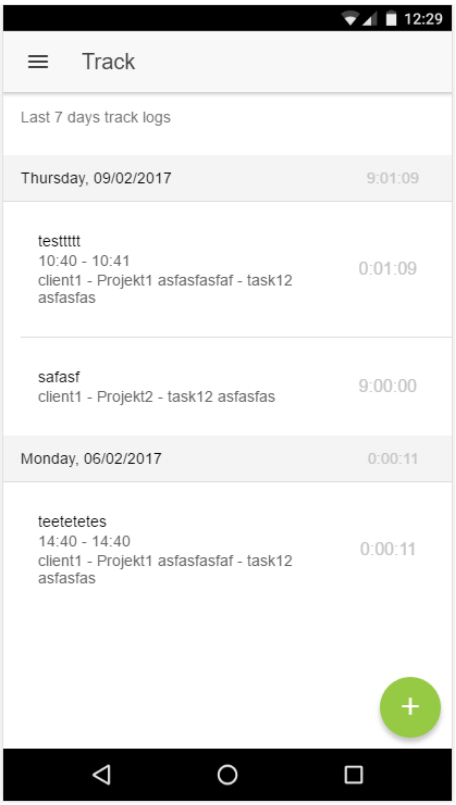
Aplikacija se inicializira v glavni komponenti, tj. AppComponent. Prav tu tudi pokličemo in nastavimo vse vtičnike Ionic Native, ki jih uporabljamo v aplikaciji. Po inicializaciji nastavimo glavno stran aplikacije, v našem primeru je to stran za vpis (**LoginPage**). Vse uporabljene komponente in zunanje dodatke (npr. knjižnica za delo s časi Moment) pa vključimo v skupno komponento AppModule. Te vključene komponente so nam na voljo v celotni aplikaciji. AppModule je TypeScript datoteka, ki vsebuje razred z meta opisom. V tem opisu navedemo vse deklaracije (vse strani aplikacije), uvoze (zunanje knjižnice) in npr. izvajalce (datoteke za povezavo z zaledjem).

V mapi app se nahaja tudi datoteka AppGlobals (komponenta za deklaracijo globalnih nastavitev aplikacije, kot so npr. clientId za API in naslov apiUrl). Poleg datoteke AppGlobals, je v tej mapi še App.scss, globalna datoteka za stile aplikacije.

V začetku razvoja aplikacije smo implementirali bolj osnovne strani kot so: **Login**, **Signup** in **Forgot Password**. Vse te strani vsebujejo preprosto formo za vnos podatkov, ki jo pošljemo v ozadje aplikacije in jo nato obdelamo.

Glavna funkcionalnost aplikacije je sledenje časa, ki se odvija na strani **Track** (slika 3.3). To je privzeta stran, na katero smo preusmerjeni takoj po vpisu v aplikacijo. Vsebuje poročilo o sledenju časa za zadnjih 7 dni in gumbe za kreiranje nove enote časa oz. štartanje sledilca časa.

Za navigacijo na druge strani aplikacije lahko aktiviramo stranski meni z navigacijo tako, da kliknemo na "hamburger" ikono v levem zgornjem kotu. Ta nam odpre stranski meni, ki ima povezave na **Track** stran, **Shortcuts** stran in povezavo na **Logout**. Navigacijo v orodju Ionic definiramo tako, da v glavni komponenti (AppComponent) definiramo tabelo z objekti. Vsak objekt vsebuje naslov povezave, ikono vnosa navigacije, in komponento, na katero hočemo uporabnika preusmeriti. Naj omenimo še, da navigacija v



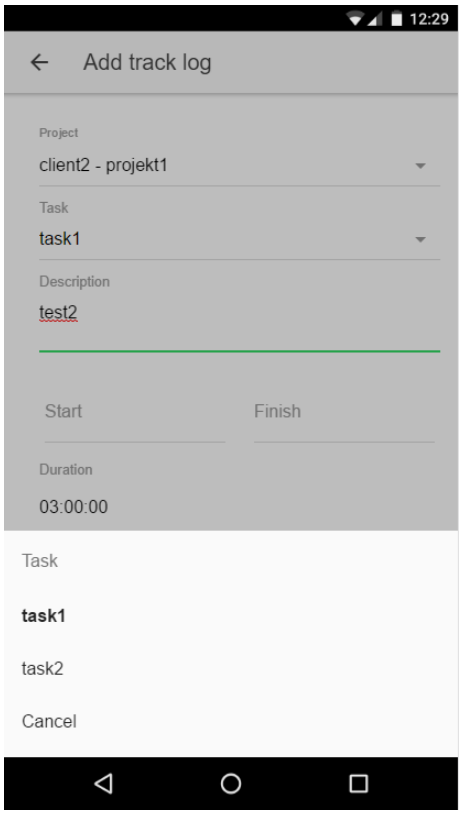
Slika 3.3: Privzeta stran aplikacije **My Hours** - **Track**.

ogrodju Ionic deluje kot sklad, torej strani dodajamo v sklad z *this.nav.push (imeKomponente)* in odvzemamo s sklada z ukazom *this.nav.pop()*.

Druga pomembna stran aplikacije je forma za ročni vnos časa oz. komponenta *AddTrackLog*. V tej komponenti najdemo formo za vnos podatkov, kot so: projekt, opravilo, opis, čas začetka sledenja, čas konca sledenja, trajanje, dodatni stroški in izbira dneva sledenja. Uporabniku se v vnosu za projekt naložijo projekti, na katere je uporabnik dodeljen v aplikaciji **My Hours**. Če izbere določen projekt, se mu naložijo še dodeljena opravila.

Na sliki 3.4 vidimo primer pojavnega okna (*ang. popup window*) za izbiro opravila, iz seznama uporabniških opravil na projektu. V primeru majhnega števila opravil na projektu (do 6 opravil), se nam odpre okno, kot je prikazano na sliki 3.4, v primeru več opravil pa se odpre pojavno okno z drsnikom, ki

je pozicionirano na sredini glavnega okna.

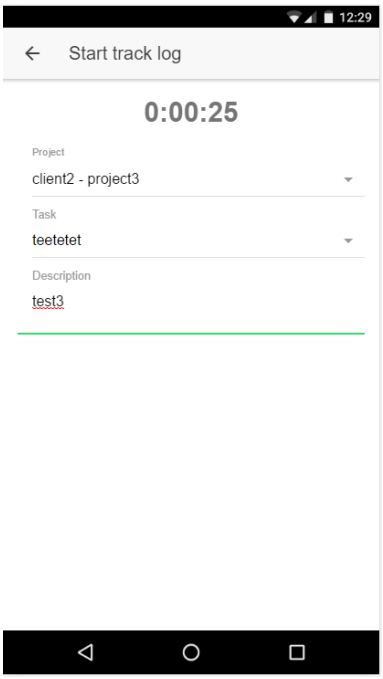


Slika 3.4: Prikaz možnosti izbire opravila iz seznama uporabnikovih opravil na projektu.

Stran podobna formi za ročni vnos časa je še komponenta za urejanje časa *EditTrackLog*. Pri tej komponenti že imamo vrednosti in jih samo prikažemo ter ponudimo možnost za spremembo le teh.

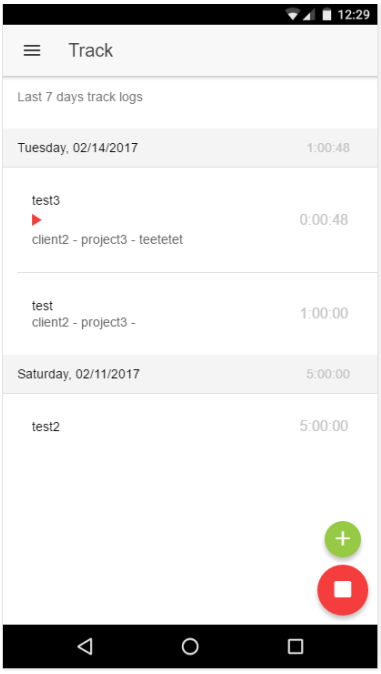
Zadnja od komponent za urejanje časa pa je komponenta za štartanje časovnika *StartTrackLog*, ki je prikazana na sliki 3.5. S to komponento hitro, brez ročne izbire, zaženemo časovnik, kateremu lahko opcijsko pripnemo še projekt oz. opravilo, in opis. V uporabniškem vmesniku, prikazanem na sliki 3.6, lepo vidimo prikaz, ko časovnik teče in rdeči gumb za prekinitev časovnika.

Na sliki 3.7 vidimo primer delčka kode, kjer prikazujemo plavajoče ak-



Slika 3.5: Prikaz **StartTrackLog** strani z možnost pripenjanja projekta, opravila in opisa.

cijske gumbe (*ang. floating action buttons*). V prvem odseku kode vidimo, da smo kreirali nov element za gumb (*ang. button*) in mu priredili *ion-fab* atribut ter dodali ikono za plus znak (ime *add* se v ogrodju Ionic avtomatsko preslika v plus ikono). V drugem odseku kode pa vidimo element, ki je zbirka plavajočih gumbov in ima dodan parameter *side*, ki pove, v katero smer se odpre zbirka elementov ob kliku. Znotraj zbirke *ion-fab-list* imamo gumb, ki pokliče metodo za začetek sledenja in zapre element. Poleg tega vsebuje, podobno kot prejšnji *button* element, še ikono in labelo z napisom *Start track log*. Oba elementa vsebujeta še pogojni stavek **ngIf*, ki v Angularju pove, da naj prikaže ta element samo v primeru, ko je pogoj resničen.



Slika 3.6: Prikaz **Track** strani s časovnikom in gumbom za prekinitev.

```
<button *ngIf="!runningTime" ion-fab>
  <ion-icon name="add"></ion-icon>
</button>
<ion-fab-list *ngIf="!runningTime" side="top">
  <button
    ion-fab
    (click)="onStartTrackLog(); trackingActionsFAB.close()"
    color="primary">
    <ion-icon name="play"></ion-icon>
    <div class="label">Start track log</div>
  </button>
</ion-fab-list>
```

Slika 3.7: Delček kode za prikaz plavajočih akcijskih gumbov.

3.4 Povezava z zaledjem

Povezava z zaledjem je ena od najpomembnejših opravil, ki jih naša mobilna aplikacija izvaja za svoje optimalno delovanje. V zaledju imamo spletni vmesnik - API (API - Application Programming Interface), z več končnimi točkami, ki jih kličemo iz naše aplikacije. Spletni API deluje v načinu REST (REST - Representational State Transfer), končne točke pa sprejemajo klice, kot so GET, POST, PUT in DELETE in vračajo podatke v JSON (JSON - JavaScript Object Notation) obliki.

Za povezavo z zaledjem nam ogrodje Ionic ponuja privzeto knjižnice ogrodja Angular, to je knjižnica HTTP. Priporočljiv način je, da te HTTP klice izvajamo znotraj servisov oz. izvajalcev (*ang. providers*), kot jih drugače imenujemo v Ionic žargonu.

Za avtentikacijo v mobilni aplikaciji **My Hours** smo uporabili tipično OAuth (OAuth - Open Standard for Authorization) strukturo. Ko se prijavimo v aplikacijo, pošljemo zahtevo za OAuth žeton, ki ga dobimo uspešno, če sta elektronski naslov in geslo pravilna. Ta žeton shranimo v lokalno shrambo in ga dodamo vsaki HTTP zahtevi, za pridobitev virov iz spletnega vmesnika. Da izboljšamo varnost takih aplikacij, nam naši žetoni potečejo po nekem določenem času (npr. po dveh urah). Nov žeton zahtevamo s pomočjo osvežitvenega žetona. To pa lahko naredimo tako, da smo pri vsaki HTTP zahtevi pozorni na odgovor spletnega vmesnika. Če pred našo zahtevo žeton preteče in ima posledično odgovor strežnika določeno statusno šifro, lahko zaprosimo za nov žeton.

Za vsako funkcionalnost aplikacije kreiramo svojega izvajalca oz. servis za klice določenih končnih točk na spletnem vmesniku. Npr. za klice naše osnovne funkcionalnosti upravljanja s časom imamo izvajalec LogProvider. V tem izvajalcu imamo metode, kot so *getRecentTrackLogs*, *addTrackLog*, *editTrackLog*. Te metode kličemo iz naših komponent in se nanje naročimo (*ang. subscribe*). V spodnjem primeru (slika 3.8), vidimo metodo za pridobivanje projektov, glede na uporabniški enotni identifikator. V metodi pokličemo projektnega izvajalca in metodo za pridobivanje projektov ter se

nanjo naročimo. Vsakič, ko dobimo podatke nazaj preko povratnih klicev, te podatke shranimo v projects spremenljivko in prekinemo izvajanje prikaza nalaganja. V primeru napake pa napako izpišemo s pomočjo metode *console.error(err)*.

```
getProjectsForUserId() {
  this.projectProvider.getProjectsForUserId().subscribe(
    data => {
      this.projects = data;
      this.loader.dismiss();
    },
    err => console.error(err)
  );
}
```

Slika 3.8: Primer uporabe naročanja (*ang. subscribing*) na podatke pri pridobivanju projektov za uporabnika.

Obljube (*ang. promises*) in opazovanci (*ang. observables*) sta dva načina dela z asinhroni klici spletnega strežnika. Obljube se od opazovancev razlikujejo po tem, da obravnavajo samo en dogodek, opazovanci pa več dogodkov hkrati. Obljuba nam omogoča, da pridobimo odgovor z uporabo metod, kot so *then*, in obravnavanje napak z uporabo metode *catch*. Poleg tega lahko obljube tudi verižimo, kar nam omogoči lepšo in razumljivejšo kodo, kot pa veriženje povratnih klicev v JavaScript jeziku (*ang. callbacks*). [13]

Opazovance (*ang. observables*) si lahko predstavljamo kot tok dogodkov, ki nam omogoča posredovanje več dogodkov v povratni klic aplikaciji, ali pa nobenega. V sodobnih ogrodjih je uporaba opazovancev priporočena pred uporabo obljub, saj nam opazovanci omogočajo več funkcionalnosti. Glavna prednost opazovancev pred obljubami je, da jih lahko prekinemo. Če nas rezultat nekega HTTP klica ne zanima več, opazovanci omogočajo preklic naročnine. Za razliko od opazovancev, bi obljuba v vsakem primeru vrnila odgovor HTTP klica, tudi če rezultata ne potrebujemo več. Opazovanci po-

nujajo še nekaj zelo uporabih operatorjev, s katerimi na način funkcionalnih programskih jezikov, preoblikujemo oz. uredimo odgovor našega spletnega vmesnika.

Eden od teh operatorjev je operator *map*, ki transformira vrednosti opazovane spremenljivke z uporabo funkcije na vsaki vrednosti. V spodnjem primeru (slika 3.9), nam operator *map* preslika JSON objekte, ki jih dobimo iz API-ja v JavaScript objekte, ki jih lahko manipuliramo v naši aplikaciji.

```
51
52      let response = this.http.post(`${this.apiUrl}/logs
53        logData, options)
54        .map(res => res.json());
55
56      return response;
57    }
```

Slika 3.9: Primer uporabe knjižnice HTTP za pošiljanje zahteve na spletni vmesnik.

3.5 Uporaba knjižnice MomentJS

Aplikacija **My Hours** dela v večini primerov s časi. Ker pa ponujamo veliko uporabniških nastavitev, lahko postane ročno pretvarjanje časov dokaj naporno. V tem primeru se lahko poslužujemo uporabe knjižnice za delo s časi. Knjižnica MomentJS je ena boljših izbir. V Ionic aplikacijo vključimo MomentJS tako, da dodamo vnos v datoteko package.json (paketi iz repozitorija NPM). Naslednji korak pa je samo še vključitev moment razreda v naši datoteki in že lahko uporabljamo metode v tej knjižnici.

Metode knjižnice MomentJS smo uporabili v straneh aplikacije za npr. pretvarjanje UTC formata začetnega in končnega časa enote sledenja časa. Poleg te uporabe pa se omenjena knjižnica velikokrat uporablja v t. i. pipah (*ang. pipes*). Pipe nam omogočajo transformacijo oz. formatiranje vhoda v poljuben format. Na sliki 3.10 je prikazan primer transformacije vhodnega datuma v zapis dneva in celotnega datuma, po formatu iz uporabniških nastavitev za zapis formata datuma. V metodo *transform* dobimo vrednost v spremenljivki *value* in to vrednost preoblikujemo v MomentJS objekt ter formatiramo v zapis dneva in celotnega datuma.

```
1  import { Pipe, PipeTransform } from '@angular/core';
2  import moment from 'moment';
3
4  @Pipe({ name: 'dateFormatter' })
5  export class DateFormatterPipe {
6      currentUser: any;
7
8      transform(value: any, formatTerm) {
9          return moment(value).format('dddd, '
10             + this.currentUser.account.dateFormat);
11     }
12
```

Slika 3.10: Primer uporabe pipe za formatiranje datuma.

3.6 Uporaba vtičnikov za dostop do naprave

Za dostop do naprave v naši aplikaciji smo uporabili zbirko ovojev za Apache Cordova, vtičnike imenovane Ionic Native. Ta zbirka nam omogoča dostop do domorodne funkcionalnosti naprave v naši aplikaciji.

Apache Cordova ovije HTML in JavaScript aplikacijo v domorodni zabojnik, ki lahko dostopa do funkcionalnosti naprave na različnih platformah. Te funkcije so dostopne preko JavaScript API-ja in nam omogočajo, da naslovimo skoraj vse naprave na trgu ter objavimo svojo aplikacijo v trgovini z aplikacijami.

Vsak Cordova projekt vsebuje datoteko *config.xml*, ki je globalna konfiguracijska datoteka in nam opiše, katere platforme, uporabljene vtičnike in druge nastavitve projekta ciljamo.

Ionic Native ovije vtičnikov povratni klic (*ang. callback*) v obljube oz. opazovance, ki nam predstavljajo dva tipična načina pridobivanja podatkov v ogrodju Angular. Prav tako kot samo ogrodje Ionic, se tudi Ionic Native namesti preko repozitorija NPM in je že privzeto vključeno v vsak naš zgeneriran projekt. Da pa lahko v aplikaciji uporabimo izbrani vtičnik, ga moramo najprej namestiti z uporabo Ionic oz. Cordova ukaznega orodja. Za tem ga še preprosto vključimo v našo datoteko, kjer ga bomo uporabili.

V naši aplikaciji **My Hours** smo uporabili nekaj Ionic Native vtičnikov, med katerimi je tudi vtičnik Splashscreen. Ta vtičnik nam med zagonom aplikacije prikaže indikator nalaganja in ga takoj pri prikazu začetne strani aplikacije tudi skrije. S takim indikatorjem omilimo percepcijo čakanja, da se aplikacija naloži.

Na slikah 3.11 in 3.12 vidimo implementacijo oziroma prikaz Splashscreen vtičnika. Na metodi *initializeApp()* pokličemo funkcijo *platform.ready()*, ko se aplikacija naloži, nam obljuba vrne rezultat in takrat skrijemo prikaz Splashscreen strani.

```
3
4    import { Splashscreen } from 'ionic-native';
5
6    initializeApp() {
7      this.platform.ready().then(() => {
8        Splashscreen.hide();
9      });
10   }
11
```

Slika 3.11: Uporaba vtičnika Splashscreen v aplikaciji.

Slika 3.12: Prikaz privzetega vtičnika Splashscreen na napravi.

Poglavje 4

Namestitev aplikacije

Cilj izdelave naše hibridne aplikacije je, da jo namestimo v trgovine za mobilne aplikacije, kot so Google Play oz. App Store. Da to dosežemo, je potrebno namestiti SDK (SDK - Software Development Kit) teh platform in jih dodati kot platforme v našo Ionic aplikacijo. Trenutno aplikacija še ni ponujena v trgovinah, tako da je prikazan samo postopek namestitve aplikacije prek Ionic ukazne vrstice in USB vhoda.

Gartner raziskava trga prodaje mobilnih naprav z operacijskimi sistemi [10] kaže močno prevlado Android operacijskega sistema (z 81,7 % deležem v četrtem četrtletju 2016). Na drugem mestu pa je platforma iOS (z 17,9 % deležem v četrtem četrtletju 2016). Druge platforme, kot so Windows, BlackBerry in druge pa imajo skupaj manj kot 1 % tržnega deleža, zato smo jih po analizi trga opustili. Pomembni rezultati te raziskave so prikazani tudi v tabeli 4.1.

Raziskava hkrati kaže, da je v porastu uporaba zmogljivejših naprav, torej to za nas pomeni, da bodo naše hibridne mobilne aplikacije delovale še hitreje zaradi zmogljivejših procesorjev ter ostalih komponent in večjega delovnega pomnilnika. Na podlagi vseh teh rezultatov raziskav smo se v podjetju odločili, da podpremo najbolj popularni platformi Android in iOS. V nadaljevanju sta opisana namestitvena postopka za posamezni operacijski sistem.

Operating System	4Q16 Units	4Q16 Market Share (%)	4Q15 Units	4Q15 Market Share (%)
Android	352,669.9	81.7	325,394.4	80.7
iOS	77,038.9	17.9	71,525.9	17.7
Windows	1,092.2	0.3	4,395.0	1.1
BlackBerry	207.9	0.0	906.9	0.2
Other OS	530.4	0.1	887.3	0.2
Total	**431,539.3**	**100.0**	**403,109.4**	**100.0**

Slika 4.1: Gartner raziskava trga prodaje mobilnih naprav glede na operacijske sisteme.

Zgenerirano aplikacijo lahko zaženemo na emulatorju oz. fizični napravi (z uporabi zastavice *–device* pri ukazu *ionic run*). Uporaba emulatorja nam omogoča, da lahko na enem računalniku zgeneriramo projekt za obe platformi in ga tudi stestiramo.

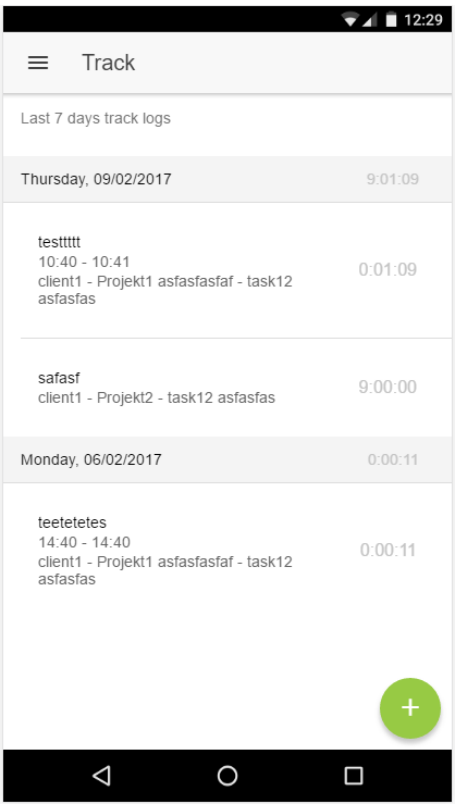
Ko se odločimo za generiranje produkcijske različice aplikacije z Ionicom, je zelo priporočljivo, da uporabimo produkcijske nastavitve v Ionic ukaznem orodju. Npr. *ionic build android –prod –release* nam optimizira našo aplikacijo in minificira kodo. Ta generirana aplikacija deluje hitreje na napravi in izboljša uporabniško izkušnjo.

4.1 Namestitev na Android platformo

Namestitev na platformo Android je dokaj lahek proces. Vse kar moramo namestiti, je Android SDK, ki ga lahko prenesemo iz uradne strani. Trenutna verzija Ionica navede minimalno verzijo, na kateri bo aplikacija še delovala in ciljno verzijo, ki je trenutno verzija 24. Dobra lastnost uporabe Ionica in hibridnih mobilnih aplikacij pa je, da za namestitev na Android platformi ne

potrebujemo celotnega IDE-ja kot je Android Studio, temveč le SDK.

Ko enkrat namestimo Android SDK, odpremo ukazno vrstico in dodamo Android platformo (*ionic platform add android*). S tem ukazom nam bo Ionic s pomočjo SDK-ja zgeneriral Android projekt. Drugi korak namestitve pa je samo še priklop Android naprave preko vhoda USB in omogočanje USB razhroščevanja na napravi, da lahko nameščamo svoje aplikacije na napravo. Po tem dejanju sledi samo še poganjanje aplikacije na Android napravi z *ionic run android –device*. Slika 4.2 prikazuje aplikacijo, zagnano na Android operacijskem sistemu.



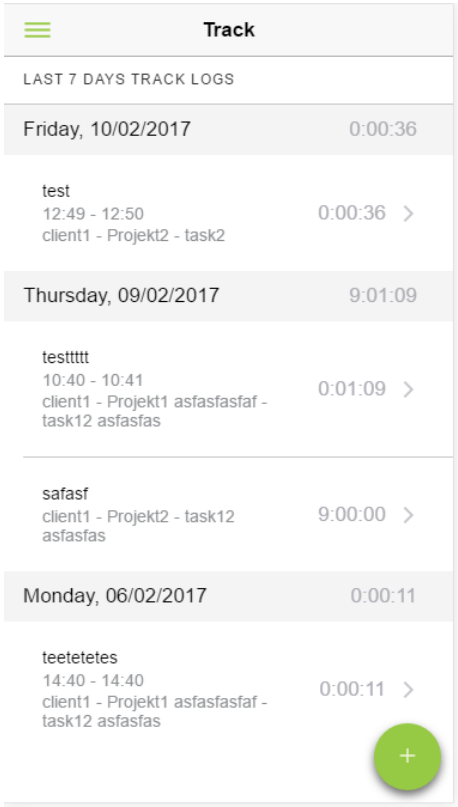
Slika 4.2: Izgled aplikacije na Android platformi.

4.2 Namestitev na iOS platformo

Za razliko od namestitve na Android, iOS platforma potrebuje generiranje oskrbovalnega profila za podpis aplikacij. Dobra stran omenjene platforme pa je, da od verzije iOS-a 9, ne potrebujemo plačljivega Apple Developer računa za testiranje svojih aplikacij.

Za iOS platformo potrebujemo računalnik z nameščenim operacijskim sistemom OSX, Xcode razvijalsko okolje, iOS SDK in brezplačen Apple Developer račun.

Po generiranju oskrbovalnega profila dodamo iOS platformo v Ionic ukaznem orodju (*ionic platform add ios*). Nato zaženemo gradnjo aplikacije z *ionic build ios*, odpremo zgeneiran iOS projekt v Xcode okolju in ga podpišemo. Zadnji korak je, da povežemo napravo prek vhoda USB in poženemo aplikacijo v Ionic orodju z *ionic run ios –device*. Slika 4.3 prikazuje aplikacijo zagnano na iOS operacijskem sistemu.



Slika 4.3: Izgled aplikacije na iOS platformi.

Poglavje 5

Sklepne ugotovitve

V tem poglavju predstavljamo zaključke, do katerih smo prišli pri razvoju mobilne aplikacije. Prav tako pa predstavimo ideje za nadaljnje delo.

5.1 Zaključki

V sklopu diplomske naloge je bila implementirana hibridna mobilna aplikacija **My Hours** za sledenje časa. Med načrtovanjem in implementacijo smo podrobno spoznali hibridne mobilne aplikacije. Spoznali smo, kaj so hibridne aplikacije, njihove prednosti in slabosti, izbrano ogrodje Ionic in orodja za izdelavo teh aplikacij.

V glavnem delu naloge smo se posvetili razvoju aplikacije, opisali smo vse, od načrtovanja aplikacije, razvoja funkcionalnosti, do povezave z zaledjem, opisali pa smo tudi uporabo vtičnikov za dostop do naprave Ionic Native. V glavnem delu diplomske naloge smo navedli nekaj posnetkov strani aplikacije in uporabljenih kod za prikaz korakov, potrebnih pri razvoju večplatformne hibridne mobilne aplikacije v ogrodju Ionic.

Med izdelavo mobilne aplikacije smo spoznali nekaj prednosti uporabe Ionic hibridnega ogrodja, kot je hiter razvoj, večplatformnost in možnost uporabe večine funkcionalnosti naprave z vtičniki Ionic Native. Ker naša aplikacija ni zelo zahtevna, kar se tiče delovnega spomina in grafike, je zelo

primerna za razvoj s hibridnimi tehnologijami. Kljub temu pa smo opazili nekaj nevšečnosti s kompatibilnostjo na različnih platformah iOS in Android operacijskega sistema.

Zadnje poglavje pa je namenjeno pripravi in namestitvi naše hibridne aplikacije na platformi, kot sta iOS in Android. Na tem mestu smo opisali posebnosti vsake od omenjenih dveh platform in postopek namestitve aplikacije na samo napravo.

5.2 Nadaljnje delo

V seznamu so navedene dograditve, ki bi jih lahko v prihodnosti implementirali v mobilni hibridni aplikaciji **My Hours**.

- **Implementacija bližnjic**: pri uporabi aplikacije **My Hours** ugotavljamo, da večinoma ponavljamo sledenja najpogostejših opravil dnevno. Npr., zjutraj pregledamo elektronsko pošto, potem nadaljujemo z raziskovalnim delom na projektu. Zato smo že v prejšnji mobilni aplikaciji implementirali funkcionalnost bližnjic, kjer predefiniramo projekt in opravilo. Tako lahko s klikom na bližnjico hitro začnemo slediti čas na projektu.

- **Lokalno shranjevanje šifrantov**: trenutno v mobilni aplikaciji lokalno shranjujemo le podatke o žetonih za avtorizacijo in podatke o uporabniku. V prihodnosti bi lahko ob zagonu aplikacije oz. preusmeritvi na glavno stran (**Track**) naložili asinhrono iz API-ja še šifrante projektov in opravil. Tako bi se izognili nalaganju projektov vsakič, ko vnašamo novo enoto časa oz. jo urejamo.

- **Delovanje mobilne aplikacije brez internetne povezave**: ogrodje Ionic nam omogoča prikaz obvestila, da je aplikacija brez internetne povezave z uporabo Ionic Native vtičnikov. Vseeno pa bi radi omogočili delovanje aplikacije tudi brez internetne povezave, vsaj del s sledenjem časa. Ko bi uporabnik ponovno vzpostavil povezavo, bi se te enote

časa sinhronizirale s strežnikom. Za takšno delovanje brez povezave bi lahko uporabili zunanjo knjižnico, ki bi uporabljala lokalno shrambo za delovanje.

Literatura

[1] My Hours. Dosegljivo: https://myhours.com/. [Dostopano 13. 2. 2017].

[2] Telerik - What is a hybrid mobile app. Dosegljivo: http://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/. [Dostopano 1. 2. 2017].

[3] Telerik - What is a webview. Dosegljivo: http://developer.telerik.com/featured/what-is-a-webview/. [Dostopano 1. 2. 2017].

[4] Ionic Framework. Dosegljivo: http://ionicframework.com/. [Dostopano 3. 2. 2017].

[5] Visual Studio Code. Dosegljivo: https://code.visualstudio.com/. [Dostopano 5. 2. 2017].

[6] Ionic Framework - Deploying. Dosegljivo: https://ionicframework.com/docs/v2/intro/deploying/. [Dostopano 5. 3. 2017].

[7] Don't Build It, Fake It First – Prototyping for Mobile Apps. Dosegljivo: https://www.interaction-design.org/literature/article/don-t-build-it-fake-it-first-prototyping-for-mobile-apps. [Dostopano 6. 3. 2017].

[8] The importance of prototyping your designs. Dosegljivo: http://www.creativebloq.com/netmag/importance-prototyping-your-designs-81412694. [Dostopano 6. 3. 2017].

[9] Semantic Versioning 2.0.0. Dosegljivo: http://semver.org/. [Dostopano 7. 3. 2017].

[10] Gartner Says Worldwide Sales of Smartphones Grew 7 Percent in the Fourth Quarter of 2016. Dosegljivo: http://www.gartner.com/newsroom/id/3609817.[Dostopano 7. 3. 2017].

[11] Top JavaScript Frameworks & Topics to Learn in 2017. Dosegljivo: https://medium.com/javascript-scene/top-javascript-frameworks-topics-to-learn-in-2017-700a397b711. [Dostopano 13. 3. 2017].

[12] Angular website. Dosegljivo: https://angular.io/. [Dostopano 13. 3. 2017].

[13] Angular 2 - promise vs observable. Dosegljivo: http://stackoverflow.com/questions/37364973/angular-2-promise-vs-observable. [Dostopano 20. 3. 2017].

[14] Scrum. Dosegljivo: https://en.wikipedia.org/wiki/Scrum_(software_development). [Dostopano 4. 4. 2017].

[15] Drifty Co. Dosegljivo: http://ionicframework.com/about. [Dostopano 8. 4. 2017].

[16] React Native vs Ionic: A Side-by-Side Comparison . Dosegljivo: https://www.codementor.io/fmcorz/react-native-vs-ionic-du1087rsw. [Dostopano 10. 4. 2017].