Primjena različitih tipova mobilnih aplikacija i alata za razvoj

Vuković, Filip

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Filip Vuković

PRIMJENA RAZLIČITIH TIPOVA MOBILNIH APLIKACIJA I ALATA ZA RAZVOJ

DIPLOMSKI RAD

Varaždin, 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Filip Vuković
Matični broj: 45252/16–R
Studij: Informacijsko i programsko inženjerstvo

PRIMJENA RAZLIČITIH TIPOVA MOBILNIH APLIKACIJA I ALATA ZA RAZVOJ

DIPLOMSKI RAD

Mentor:
Doc.dr.sc. Zlatko Stapić

Varaždin, rujan 2018.

Sadržaj:

1. Uvod

1.1. Uvod u temu

Pametni telefon je ručno osobno računalo koje posjeduje opsežne računalne mogućnosti. Moderni pametni telefoni imaju zaslon u boji osjetljiv na dodir s grafičkim korisničkim sučeljem koje pokriva prednju površinu i omogućuje korisniku korištenje virtualne tipkovnice za upisivanje i pritisak na ikonu na zaslonu. Interakcija se uglavnom vrši dodirom dok poneki uređaji još uvijek imaju nekoliko fizičkih tipki. Kao i računalo, pametni telefon koristi operacijski sustav kako bi mogao obrađivati razne aplikacije. Većina mobilnih operacijskih sustava posjeduje svoje interne pomoćne aplikacije koje se instaliraju na uređaju pri instalaciji mobilnog operacijskog sustava. Neke od tih aplikacija su: bilješke, web preglednik, fotoaparat, radio itd. Uz interne aplikacije korisnik može instalirati nove aplikacije koje su dostupne na trgovini aplikacija za određeni operacijski sustav. Drugim riječima, svaka aplikacija je posebna za svaku platformu, odnosno operacijski sustav.

Razvoj mobilnih aplikacija je proces razvijanja software-a za upotrebu na uređajima kao što su pametni telefoni ili tableti. Ovakve se aplikacije mogu instalirati na uređaj za određenu platformu ili mogu biti isporučene kao web aplikacije koje korisnik može koristiti na mobilnom web pregledniku. Takve aplikacije mogu koristiti značajke pametnog telefona. Programeri aplikacijskih softvera moraju uzeti u obzir dugi raspon veličina zaslona, hardverskih specifikacija i konfiguracija zbog intenzivne konkurencije u mobilnom razvoju i promjena unutar svake od platformi.

Mobilni razvoj odnosi se na razvoj aplikacija za tablete, pametne telefone i druge pametne uređaje koji pokreću operacijske sustave. Međutim, važno je napomenuti da se mobilni razvoj ne odnosi isključivo na razvoj mobilnih aplikacija nego obuhvaća pristup aplikaciji i njezinu upotrebu na mobilnom uređaju. Upravo zbog pristupa uređaju i načina uporabe uređaja mobilne aplikacije dijelimo na nativne, hibridne i na već spomenute web aplikacije. Navedene aplikacije razlikuju se po svojim performansama, grafičkom sučelju, utrošenom vremenu za implementaciju itd. Upravo te razlike nam govore da je vrlo važno pomno razmotriti koji tip mobilne aplikacije odgovara našim potrebama i našim zahtjevima. Odabir krive implementacije ili krivog tipa aplikacije može znatno utjecati na dobit i na produktivnost kako programera, tako i tvrtke.

Motivacija za odabir ove teme je naklonjenost autora hibridnim aplikacijama. Razlog naklonjenosti je među platformsko svojstvo takvih okvira gdje programer piše svoj kôd

jednom, a pokreće ih na svim uređajima. Važno je spomenuti da se hibridne aplikacije temelje na web tehnologijama koje su jednostavnije za implementaciju mobilnih aplikacija nego jezici nativnih aplikacija. Glavni nedostatak hibridnih aplikacija odnosi se na njihovu performansu. Hardverske specifikacije današnjih uređaja su slične računalima stoga je mišljenje autora da performansa aplikacija postaje nebitna u tom segmentu.

Temu ovoga diplomskog rada predložila je tvrtka FINA – Financijska agencija, Sektor informatike, koja je također sudjelovala i u sumentorstvu kroz razradu ove teme.

Svrha i cilj rada su istražiti sve tipove mobilnih aplikacija te uz vlastitu implementiranu aplikaciju za svaki tip usporediti i analizirati sve okvire koji će biti odabrani za implementaciju odabranih tipova. U ovom radu bit će dobiveni odgovori na pitanja:

1. Što su to hibridne, nativne i web aplikacije?
2. Prednosti i mane navedenih tipova?
3. Što je to Ionic?
4. Što je to Android studio?
5. Što je to Xamarin?
6. Što je to Materialize CSS?
7. Koji su bitni aspekti za odabir tipa aplikacije?
8. Koji tip aplikacije odabrati i zašto?

1.2. Struktura rada

Rad se sastoji od 8 sadržajno i logički međusobno povezanih cjelina:

1. Uvod - U prvom dijelu, definirana je tema samog rada te uz opis i motivaciju autora za ovu temu, definiran je cilj i svrha pisanja rada.

2. Tema aplikacije - Opisana je aplikacija koja će biti implementirana za sve okvire, uz razloge zašto je odabrana upravo ta aplikacija.

3. Hibridne aplikacije - Ovo poglavlje objašnjava što su hibridne aplikacije te kako one rade na mobilnom uređaju. Opisan je okvir Apache Cordova na kojem se temelje većina hibridnih okvira.

4. Analiza okvira za hibridne aplikacije - U ovom poglavlju provest će se analiza 9 odabranih okvira za hibridne aplikacije i zaključak na temelju analize. Svi okviri su analizirani prema sljedećim aspektima:

- Opis
- Prednosti
- Mane
- Ocjena

4.1. Ionic - Poglavlje Ionic predstavlja prvi hibridni okvir koji je odabrao autor za implementaciju prve aplikacije. U poglavlju je objašnjeno što je to Ionic te što je to Angular na kojem se Ionic temelji. Nakon toga slijedi opis implementacije aplikacije uz dodatna objašnjenja značajki Ionica koje su korištene.

4.2. Xamarin - Ovo poglavlje opisuje okvir Xamarin i biblioteku Xamarin.Forms. Ovaj okvir odabran je od strane mentora. Nadalje, opisan je Visual Studio u kojem je razvijena aplikacija na što se nastavlja opis implementacije aplikacije uz dodatna objašnjenja značajki Ionica koje su korištene.

5. Analiza okvira za web aplikacije - U ovom poglavlju provedena je analiza 3 odabrana okvira za web aplikacije i zaključak na temelju analize. Svi okviri su analizirani prema sljedećim aspektima:

- Opis
- Prednosti
- Mane
- Ocjena

5.1. Materialize CSS - Sljedeće poglavlje opisuje okvir za pristupni dio web aplikacije kao i Material Design koncept. U nastavku je dan opis implementacije aplikacije u navedenom okviru.

6. Analiza programskih jezika za izradu nativnih aplikacija - U ovom poglavlju se nalazi analiza 3 programska jezika za implementaciju nativne aplikacije uz zaključak.

Svi programski jezici su analizirani prema sljedećim aspektima:

- Opis
- Prednosti
- Mane
- Ocjena

6.1. Java i Android Studio - Predstavlja nativni tip aplikacije gdje je opisan IDE Android Studio i Java kao programski jezik. Također, opisana je i analizira sama implementacija aplikacije u Javi.

7. Usporedba svih tipova aplikacija - U ovom poglavlju uspoređene su sve aplikacije prema sljedećim kategorijama: performansa, zajednica, grafičko sučelje i potrošeno vrijeme. Svaka stavka opisana je za pojedinu aplikaciju nakon čega je izračunata prosječna ocjena svih ocjena prema aplikaciji.

8. Zaključak - U posljednjem poglavlju sažeti su bitni dojmovi za pojedinu aplikaciju te su opisane zaključne misli autora o odabiru okvira za implementaciju.

2. Tema aplikacije

Koristeći različita razvojna okruženja bit će implementirana identična aplikacija koja će se usporediti po faktorima čiji je značaj veoma bitan za razvoj mobilnih aplikacija. Tema aplikacije bit će aplikacija za vozače kamiona. Aplikacija će automatizirati poslovni proces uvođenjem sustava za praćenje distribucije tvrtke. Sustav se dijeli na dvije aplikacije:

Web aplikacija: unos, uređivanje i pregled svih dokumenta.
Mobilna aplikacija: Unos svih stanja radnih naloga prilikom cijelog procesa distribucije.

Web aplikacija već postoji te su implementirani svi API-ji za obradu podataka putem mobilne aplikacije. U ovom radu razvijat će se samo mobilna aplikacija.

Dionici mobilne aplikacije:

- Prijavljeni korisnik

Korisnički zahtjevi:

- Pregled svih radnih naloga dodijeljeni prijavljenom korisniku
- Aktivacija radno naloga
- Slanje otpremnice
- Slanje primke
- Prijava kvara

Funkcionalni zahtjevi:

- Upravljanje bazom podataka što uključuje stvaranje, pregledavanje uređivanje i brisanje:
  - Otpremnice
  - Primke
  - Slike
  - Radnih naloga
- Fotografiranje slike ili odabir slike iz galerije.

Nefunkcionalni zahtjevi:

- Sva unosna polja su obavezna pri slanju Primke.
- Sva unosna polja su obavezna pri slanju Otpremnice.
- Ograničenje učitavanje samo jedne slike na server.

- Aplikacija se ne smije ugasiti prilikom nestanka interneta.

Ova aplikacija je odabrana zato što se koristi u stvarnom svijetu. Uz ovu mobilnu aplikaciju već postoji web aplikacija preko koje se unose podaci u bazu te se prema tom unosu u bazu podaci prikazuju na mobilnoj aplikaciji. Drugim riječima, postoji server s raznim API-ijima gdje će se moći testirati sama brzina dohvaćanja podataka sa servera. Također, grafičko sučelje je vrlo bitan segment kad se uspoređuju ovakvi tipovi mobilnih aplikacija. Sve radne naloge treba prikazati na mobilnoj aplikaciji u obliku liste s kojom se može napraviti određena interakcija. Ovo je vrlo čest postupak kod mobilnih aplikacija te će biti vrlo bitno usporediti renderiranje tih naloga kao i sam izgled svih tih elemenata i njihovih animacija. Kod prijave kvara postoji mogućnost slikanja i slanja slike na server. U sklopu rada testirat će se kako se instalira i koristi određeni dodatak (za slikanje i slanje slike na server) za određeni okvir.

3. Hibridne aplikacije

Sličnost između hibridnih i nativnih aplikacija je velika. Mogu se pronaći u trgovinama aplikacija te se nakon toga instaliraju na uređaju. Pomoću njih se može slikati, podijeliti objave putem Facebooka ili ostalih društvenih mreža i igrati razne igre isto kao i na nativnim aplikacijama. Ako se želi razviti mobilna aplikacija za više platforma, npr. Android i iOS, potrebno je razvijati dvije aplikacije u različitih jezicima zbog čega nastaju novčani i vremenski problemi. Rješenje na spomenute probleme bio bi odabir i implementacija hibridnih aplikacija.

Hibridne aplikacije su zapravo web aplikacije koje su pisane klasičnim web tehnologijama, odnosno HTML, JavaScript i CSS. Hibridne aplikacije koriste preglednik uređaja za prikazivanje web sadržaja i za procesiranje JavaScript kôda. Hibridne aplikacije koriste apstraktni sloj web-prema-nativi (također poznat kao sloj mostova) koji omogućuje JavaScript-u pristup mnogim uređajima specifičnim mogućnostima i nativnim API-jima koji nisu općenito dostupni s mobilnog web preglednika (Nizamettin i Khanna, 2013). Drugim riječima, hibridne aplikacije su smještene unutar izvorne aplikacije koja koristi WebView mobilne platforme. Upravo zbog toga takve aplikacije imaju pristup korištenja mobilnim funkcijama poput fotoaparata, geolokatora itd. koje su inače često onemogućene iz običnog mobilnog preglednika. Kod hibridnih mobilnih aplikacija kôd se piše samo jednom, a verzije aplikacije za pojedinu platformu koja se želi i koja je dostupna u okviru koji se koristi za razvijanje takvih aplikacija dobijemo upravo kompajliranjem tog kôda. Iz tog razloga se takve aplikacije zovu hibridne jer nisu nativnog oblika odnosno postoji dodatni sloj između aplikacije i uređaja (Bristowe, 2015).

Neki od popularnijih okvira za razvijanje hibridnih aplikacija su Ionic, Framework 7, Xamarin, PhoneGap, a neki od njih će biti detaljno istraženi te će se razviti manje aplikacije koje će se usporediti po sljedećim svojstvima:

- Dokumentacija
- Početak
- Performansa
- Zajednica
- Grafičko sučelje

U sljedećem poglavlju bit će opisan hibridan okvir Apache Cordova zato što je taj okvir temelj većini okvira hibridnih aplikacija. Apache Cordova je kompatibilna s većinom popularnih hibridnih okvira. Upravo zbog te kompatibilnosti programeri mogu jednostavno proširiti okvir za razvoj aplikacija otvorenog kôda i stvoriti nevjerojatne mobilne aplikacije koristeći popularne alate poput Ionic okvira i IDE-ova poput Visual Studio-a.

3.1. Programski okvir

U ovom radu se spominje i koristi mnogo okvira gdje je samo za hibridne aplikacije napravljena analiza s 9 okvira takvog tipa aplikacije. Stoga je vrlo važno razumjeti što je okvir i kako nastaje.

Softverski okvir je platforma gdje se zajednički kôd s generičkom funkcionalnošću može modificirati ili nadjačati od strane programera . Okviri imaju oblik biblioteke gdje se dobro definirano sučelje aplikacijskog programa (API) može ponovno upotrijebiti bilo gdje unutar softvera koji se razvija (Schmidt, 1997).

Okvir je ponovno upotrebljiva aplikacija koja se može specijalizirati za izradu prilagođenih aplikacija. Mnogi okviri koriste ili se temelje na nekom uzorku dizajna. Uzorak predstavlja ponavljajuće rješenje za razvoj softvera ili problem unutar određenog konteksta. Uzorci i okviri mogu se zajedno primjenjivati za poboljšanje kvalitete komunikacijskog softvera hvatanjem uspješnih strategija razvoja softvera. Uzorci hvataju sažetak dizajna i softverskih arhitektura u sustavnom obliku koji razvojni programeri mogu lako shvatiti. Jedan od takvih primjera okvira je AngularJS okvir koji se temelji MVC (model-view-controller) uzorku dizajna. Stoga se zaključuje da se okvir sastoji od konkretnih dizajna, algoritama i implementacija u odabranim programskim jezicima (Schmidt, 1997).

Postoje razni okviri za sve tipove aplikacija i tehnologija, ali se oni uvijek nadograđuju i svakodnevno se stvara potreba za novim okvirima. Postavlja se pitanje kako je moguće da se uz te silne okvire stvara potreba za novima? Na okvir se može gledati kao na aplikacije koje će se implementirati u njemu, a moraju zadovoljavati sve potrebne specifikacije, ciljeve i kvalitetu koji mu korisnički zahtjevi nadmeću.

Sljedeće karakteristike su neke od onih koje bi se trebale ispitati kako bi se vidjelo da li ih okvir posjeduje te hoće li ih moći zadovoljiti:

Ispravnost - Pokazuje da li je okvir sposoban da izvrši svoje zadatke kako je definirano zahtjevima i specifikacijama.

Robusnost - Sposobnost okvira da funkcionira čak i u abnormalnim uvjetima.

Proširivost - Pojašnjava da li okvir ima mogućnost proširivosti.

Upotrebljivost - Postoji li sposobnost okvira da se ponovno koristi u cijelosti ili djelomično za nove aplikacije.

Kompatibilnost - Da li je moguće okvir koristiti, odnosno da li je kompatibilan s drugim okvirima ili sustavima (Kazman, 2012).

U ovom radu će se detaljno analizirati jedan okvir za nativne aplikacije, jedan za web aplikacije te 2 za hibridne aplikacije. Kroz analizu će se promatrati njihove razlike te će se također, vidjeti na primjeru razlika između gore navedenih karakteristika.

3.2. Apache Cordova

Kao što je već navedeno Apache Cordova (https://cordova.apache.org/) je kompatibilna s većinom hibridnih okvira. Apache Cordova danas ima veliku ulogu u hibridnim aplikacijama. Ona je platforma za mobilni razvoj mobilnih aplikacija. Pomoću Cordove mogu se izgraditi aplikacije za mobilne uređaje pomoću osnovnih web tehnologija (HTML, CSS, JavaScript) što bi značilo da se ne mora nužno koristiti nativne API-ije za odabranu platformu bilo to iOS, Android, Windows itd. Također, postoje mogućnosti korištenja raznih proširenja HTML elementa i JavaScript značajki. Kao što je već spomenuto, postoji mnogo okvira za hibridne aplikacije (Apache Cordova, 2018).

Cordova okvir uvelike olakšava razvoj programera za pakiranje hibridnih mobilnih aplikacija za svaku ciljanu mobilnu platformu. Stoga aplikacija napravljena u Cordovi može se instalirati baš kao i izvorne mobilne aplikacije usprkos razvoju web tehnologija. Istovremeno, okvir omogućuje razvojnim programerima da aplikaciji omoguće pristup nativnim uređajima putem različitih dodataka. Razvojni programer mora razumjeti različite aspekte Cordova okvira kako bi odabrao pravi okvir za razvoj hibridnih mobilnih aplikacija (Apache Cordova, 2018).

Sa sigurnošću se može reći da su Cordova-ini dodaci sastavni dijelovi ekosustava Cordova-e. Oni pružaju sučelje za Cordova-u i njezine nativne komponente kako bi međusobno komunicirale i povezivale standardne API-je s uređajima. Projekt Apache Cordova održava skup dodataka nazvanih *Core Plugins*. Ovi temeljni dodaci omogućuju aplikaciji pristup značajkama uređaja kao što su baterija, kamera, kontakti itd. (Apache Cordova, 2018).

Pored jezgrenih dodataka postoji i nekoliko dodataka treće strane koji pružaju dodatna povezivanja značajki koje nisu nužno dostupne na svim platformama. Mogu se potražiti dodaci Cordova-e pomoću pretraga za dodatke ili NPM. Također, mogu se razviti vlastiti dodaci kao što je opisano u vodiču za razvoj dodataka. Dodaci mogu biti potrebni za komunikaciju između Cordova-e i prilagođenih izvornih komponenata. Razni pluginovi su korišteni u projektima, a ovo su neki od primjera koji se instaliraju preko NPM -a:

- `npm install cordova-plugin-geolocation` –instalacija dodatka za korištenje geolokacije u našoj aplikaciji
- *`npm install cordova-plugin-file-transfer`* – instalacija plugina za transfer dokumenta na server
- *`npm install cordova-plugin-camera`*– instalacija plugina za korištenje kamere

4. Analiza programskih okvira za hibridne aplikacije

4.1. Usporedba programskih okvira za razvoj hibridnih aplikacija

Kada se govori o razvoju hibridnih aplikacija važno je spomenuti kako se tijekom razdoblja javljaju novi okviri s ciljem olakšavanja života programera. Međutim, odabir jednog od okvira bogatog značajkama i korisnički prilagođenog nije tako jednostavno kao što se čini. Iz tog razloga napravljena je analiza popularnih okvira.

U nastavku analizirat će se devet različitih okvira za izradu hibridnih aplikacija. Ionic, Xamarin, Kendo UI, Onsen UI, Framework 7, PhoneGap, Mobile Angular UI, Intel XDX i Appcelerator's Titanium bit će ukratko opisani nakon čega će se navesti njihove glavne prednosti i nedostaci. U samoj konačnici svaki od navedenih okvira bit će ocijenjen na temelju 5 bitnih kriterija za razvoj mobilnih aplikacija, a to su:

- Dokumentacija
- Početak
- Performansa
- Zajednica
- Grafičko sučelje

Kriteriji za ocjenjivanje okvira odabrani su proučavajući stručnu literaturu koja uspoređuje navede okvire. **Dokumentiranje** okvira je važan proces u razvoju aplikacijskog okvira i predstavlja preduvjet za njegovo korištenje. Jednostavna, točna, lako uporabljiva, razumljiva i učinkovita dokumentacija je konačni cilj dokumentacije pojedinog okvira. Dokumentiranje okvira treba početi od samog početka razvoja okvira i razvijati se zajedno sa njim. **Početak** govori o zahtjevnosti rada određenog okvira u kojem programer nema mnogo iskustva. **Performanse** mobilne aplikacije određene su korisnikovom percepcijom o tome koliko se aplikacija dobro izvodi. Drugim riječima, mjeri se brzina aplikacije, koliko brzo se pokreće, koliko dobro koristi memoriju uređaja i kako se glatko animacija ili elementi ponašaju. **Zajednica** okvira je proces gdje se članovi zajednice okupljaju kako bi kolektivno djelovali i generirali rješenja za zajedničke probleme. Kod zajednice najvažnija je redovita interakcija i stvaranje prilika za razmjenu znanja, ideja i iskustva. **Grafičko sučelje** je veoma važan kriterij pri odabiru okvira za mobilni razvoj te se razlikuje kod svih okvira. Svaki okvir

ima svoje posebno napravljene komponente i svoje načine kako učiniti izgled što prirodnijim (AppDynamics, 2018).

Analiza navedenih okvira provedena je kako bi se bolje upoznale njihove prednosti i mane i omogućilo lakše uspoređivanje istih. Ova analiza će poslužiti kao pomoć pri odabiru hibridnih okvira mentoru i sumentoru rada koji će nadalje biti detaljnije analizirani u radu. Ocjenjivanje kriterija pojedinog okvira je zaključeno ispitivanjem literature dostupne za okvire i ekstrakcijom pojmova, prednosti i nedostataka vezanih uz okvire i filtriranjem tih podataka. Na temelju provedene analize bit će odabrana 2 okvira. Prvi okvir će biti odabran od strane autora rada dok će drugi okvir biti odabran od strane mentora.

Tablica 1. Analiza okvira za hibridne aplikacije

	Ionic	Xamarin	Kendo UI
OPIS	Ionic je besplatan i otvorenog je kôda, Ionic nudi biblioteke HTML, CSS i JS optimizaciju za mobilne uređaje za izgradnju visoko interaktivnih aplikacija. Izgrađen sa Sassom i optimiziran za AngularJS (Ionic, 2018).	Xamarin je alat za razvoj međuplatformskih aplikacija. Sa Xamarin-om možete koristiti C # za aplikacije za iOS, Android i Universal Windows. Uz Xamarin Forms, dizajn sučelja za sve tri platforme može se ostvariti unutar XAML-baziranog okvira (Microsoft, 2018).	Kendo UI je sveobuhvatni okvir koji se sastoji od 70 + UI widgeta, obiljem gadgeta za vizualizaciju podataka, izvora podataka na strani klijenta i ugrađenom MVVM bibliotekom. Omogućuje integraciju AngularJS i Bootstrap te se također distribuira kao dio više proizvodnih jedinica (Gejotres, 2015).

PREDNOSTI	• Lako naučiti, puno definiranih komponenata • Kvalitetna, velika i konzistentna dokumentacija • Među platformski • Angular- Google • Cordova (Dumić, 2017), (Gejotres, 2015)	• Performanse skoro kao kod Nativnih aplikacija • Nativan UX • Među platformski (Dumić, 2017)	• jQuery i AngularJS podrška • Dokumentacija • Stvara osjećaj nativne aplikacije • Predlošci • Podrška (Dumić, 2017), (Gejotres, 2015)
NEDOSTACI	• Lošija performansa (Dumić, 2017), (Gejotres, 2015)	• Zajednica, podrška • Veća veličina aplikacije • Code Sharing 90% • Nije pogodna za aplikacije s jačom grafikom (Dumić, 2017)	• Miješanje AngularJS i jQuery(većina ljudi ne voli) • Komercijalan je (Dumić, 2017), (Gejotres, 2015)
OCJENE	Dokumentacija : 9 Početak : 10 Performansa : 5 Community : 8 GUI: 10	Dokumentacija : 7 Početak : 6 Performansa : 10 Community : 7 GUI: 8	Dokumentacija : 8 Početak : 10 Performansa : 5 Community : 8 GUI : 8

	ONSEN UI	Framework 7	PhoneGap
OPIS	Onsen korisničko sučelje pomaže u razvoju hibridnih i web aplikacija. Ako se razvijaju hibridne aplikacije može se koristiti pomoću naredbenog retka Cordova PhoneGap ili pomoću alata Monaca (Gejotres, 2015).	Framework7 je besplatan i open source mobilni HTML okvir za razvoj hibridnih mobilnih aplikacija ili web aplikacija s iOS izvornim izgledom i dojmom. Sve što je potrebno za njegovo korištenje je jednostavan HTML izgled i prilozi okvira CSS i JS datoteke (Jscrambler, 2017).	S obzirom na to da je fokusiran na mobilni razvoj i da je open-source, PhoneGap je jedan od najpopularnijih okvira za razvoj aplikacija kojim se koriste mnogi iskusni programeri. Pomoću okvira PhoneGap mogu se razvijati robusne i bogate aplikacije koje su u potpunosti prilagođene hibridne aplikacije koje učinkovito funkcioniraju na više platformi (Jscrambler, 2017).
PREDNOSTI	• Open source i besplatan • Širok spektar 3rd pluginova • Monaca IDE • Cordova • Radi s bilo kojim JavaScript okvirom (Dumić, 2017), (Gejotres, 2015)	• Open source i besplatan • Dobra performansa • Puno U-I gotovih elemenata • Dokumentacija • Mnogo gotovih ui elemenata, lako ih prilagoditi (Dumić, 2017)	• Open source i besplatan • Veliki izbor biblioteka • Brza izrada prototipa (Dumić, 2017)

NEDOSTACI	• Puno manja Zajednica od konkurencije • Podrška dostupna jedino preko StackOverflowa (Dumić, 2017), (Gejotres, 2015)	• Nema početnog paketa • Nije međuplatformski okvir (Dumić, 2017)	• Loša performansa • Manjak UI komponenta • Dokumentacija (Dumić, 2017)
OCJENE	Dokumentacija: 4 Početak: 10 Performansa: 5 Zajednica: 6 GUI:8	Dokumentacija: 9 Početak: 5 Performansa: 8 Zajednica: 8 GUI:9 samo IOS!	Dokumentacija: 5 Početak: 8 Performansa: 4 Zajednica: 9 GUI: 8

	Mobile Angular Ui	Intel XDX	Appcelerator's Titanium
OPIS	Mobile Angular UI je hibridni mobilni aplikacijski okvir izgrađen na Bootstrap 3 (Gejotres, 2015).	Intel XDK je okruženje za razvoj mobilnih aplikacija s više platformi koja omogućuje izradu mobilnih i tabletnih aplikacija pomoću HTML5 i JavaScript (Jscrambler, 2017).	Titanium nudi proširivo razvojno okruženje za stvaranje hibridnih aplikacija. Sadrži SDK otvorenog kôda s više od 5000 API-ja za mobilne uređaje i mobilne uređaje, Studio, snažan IDE, Alloy, MVC okvir i Cloud Usluge (Angelini, 2012).

PREDNOSTI	• Dodatne komponente koje nedostaju u Bootstrap 3 • Lagan i "ogoljen" Angular • Super za stvaranje mobilne aplikacije iz desktop aplikacije (Dumić, 2017), (Gejotres, 2015)	• HTML, CSS, JS mobile i Cordova • Responzivan dizajn • Lagana provjera na različitim uređajima kroz XDK emulate tab (Dumić,2017), (Jscrambler, 2017)	• Open source i besplatan • Nativne UI komponente • Appcelerator pruža dodatne vrijednosti kao što je BaaS, app analitike i tržište s 3rd party komponentama (Angelini, 2012)
NEDOSTACI	• Loša dokumentacija • Nedovoljno zreo uspoređujući s ostalim okvirima • Zajednica ne postoji (Dumić, 2017), (Gejotres, 2015)	• Performansa • Specifične značajke za platforme • Limitirane funkcionalnosti zbog *security* modela • Zajednica (Dumić, 2017), (Jscrambler, 2017)	• Lokalno upravljanje SDK • Normalizacija UI kroz platforme (Angelini, 2012)
OCJENE	Dokumentacija: 5 Početak: 5 Performansa: 5 Zajednica: 3 GUI:8	Dokumentacija: 5 Početak: 6 Performansa: 5 Zajednica: 5 GUI:9	Dokumentacija: 6 Početak: 6 Performansa: 5 Zajednica: 6 GUI:9

Kao glavni problem hibridnih aplikacija mnogi članci navode kako aplikacija takvog tipa nema osjećaj nativnog izgleda te kako je performansa hibridnih aplikacija dosta sporija od nativnih. Većina hibridnih okvira ima mnogo definiranih komponenta koje izgledaju poprilično dobro te ih je jednostavno implementirati, odnosno ubaciti u projekt. Takvim aplikacijama nedostaje nativan osjećaj gdje nedostaju razne animacije pri korištenju. Upravo takva svojstva dijele nativan UI od hibridnog.

Nakon usporedbe 9 najpopularnijih okvira hibridnih aplikacija donesen je zaključak kako je Ionic okvir trenutno najbolji okvir za razvoj hibridnih aplikacija. Spoj AngularJS-a, HTML-a, Cordove, raznih gotovih komponenata te bogate i detaljne dokumentacije trenutno nema konkurencije među okvirima hibridnih aplikacija. Ionic će biti detaljno analiziran te je odabran od strane autora.

Sljedeći odabrani okvir bit će Xamarin koji je u posljednjim godinama u velikom usponu. Xamarin je u vlasništvu Microsoft-a stoga se može zaključiti da programeri imaju dobru pozadinu okvira i neku vrstu sigurnosti. U ovom radu detaljno će se proći biblioteka Xamarin.Forms koja omogućuje nativan UI što je velika prednost kod hibridnih aplikacija. Xamarin.Forms je odabran od strane mentora nakon provedene analize.

4.2. Ionic

Posljednjih se godina razvoj mobilnih aplikacija s Ionic-om smatra jednim od najpouzdanijih i najboljih rješenja za razvoj hibridnih mobilnih aplikacija.

Ionic je besplatan SDK za hibridni mobilni razvoj. Prva je verzija izašla 2013. te se temeljila na AngularJS-u i Cordovi, odnosno namjera je bila omogućiti razvoj mobilnih aplikacija koristeći web tehnologije (HTML, CSS, JavaScript), ali da pritom ta aplikacija bude međuplatformsko to jest da se može koristiti na više platformi (Android, iOS, Windows itd.). Također, mora se istaknuti da je glavni cilj Ionica da uz pisanje jednog kôda za više platformi ne izgubi osjećaj nativne mobilne aplikacije (Ionic.com, 2018).

Ionic je usmjeren na moderne mobilne uređaje, točnije na Android. Podržava Android 4.1. verziju i iznad, iOS verziju 7 i iznad te se mogu razvijati i mobilne aplikacije za operacijski sustav Windows. Ionic koristi Cordovu kako bi mogao koristiti nativne značajke pametnih telefona kao što su: Kamera, GPS, Tipkovnica, Bluetooth, WiFi itd.

Slika 1: Ionic logo

(Ionic, 2018)

Uz pomoć već definiranih komponenata Ionic stvara sučelje aplikacije, ali i samu interakciju s korisnikom te je poprilično blizu nativnoj mobilnoj aplikaciji. Primjeri tih komponenata uključuju tipke, kartice, izbornike, popise, prozore itd. (Bristowe, 2015).

Osim ovih komponenata koje pružaju oblikovanje, izgled i dojam za mobilne aplikacije, Ionic pruža i prikladna ponašanja poput navigacije i pokreta. Neka od tih ponašanja su pomicanje, rotacija prstiju itd. Ionic uključuje i zbirku ikona zvanih Ionicons. Sve ove komponente su elementi koji se upotrebljavaju za izradu aplikacija u Ionicu zajedno s dodacima Cordove.

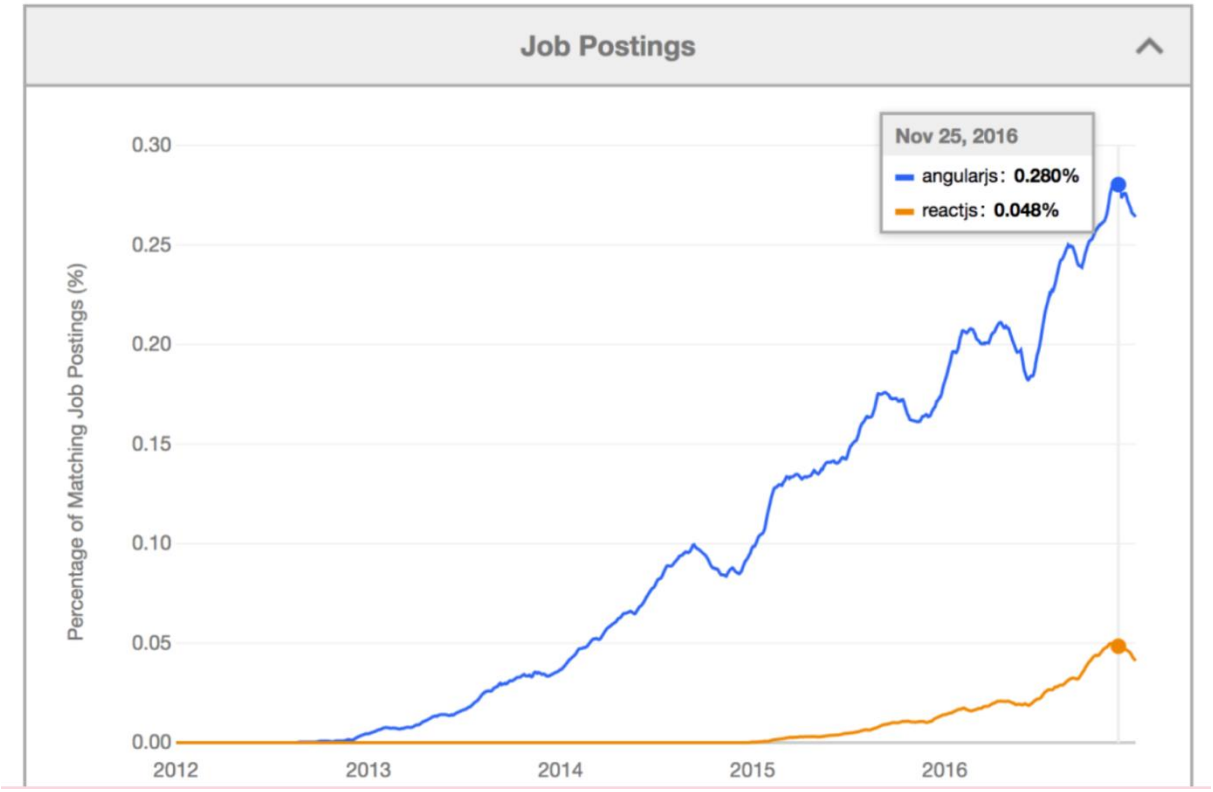
4.2.1. AngularJS

Od Ionic verzije 2.0, Ionic koristi Angular 2 i Typescript, ali pošto je aplikacija iz ovog rada rađena u Ionic 1 koja se ipak bazira na AngularJS-u, koristit će se prva verzija AngularJS-a.

AngularJS je besplatan web JavaScript okvir koji se koristi isključivo na prednjem sustavu. Angular održava Google što je njegova velika prednost te su njegova zajednica i podrška ogromne i uvelike olakšavaju rad programerima. AngularJS ima mnoge prednosti u odnosu na druge JavaScript okvire na strani klijenta. AngularJS koristi MVC dizajn uzorak i obuhvaća taj uzorak u potpunosti. Model, pogled i kontroler su jasno definirani u AngularJS i služe za pojednostavljivanje procesa razvoja. Uz AngularJS programeri mogu izraditi aplikacije koje imaju jasno razdvajanje između funkcionalnih slojeva (Williamson, 2015). AngularJS omogućuje proširivanje logike HTML-a te stvaranje zasebne logičke cjeline. Glavna značajka AngularJS-a je vezanje JavaScript-a varijabla na HTML. Vrijednosti tih

JavaScript varijabli mogu se ručno postaviti unutar kôda ili se mogu preuzeti iz statičkih ili dinamičkih JSON resursa. S tom značajkom, odnosno prilagođenim Angular direktivama (ng-if, ng-repeat, ng-html) može se kreirati zavidna logika, smanjiti kôd JavaScript-a i dobiti puno čišće cjeline (Angular.org,2016).

Mali nedostatak takvog okvira to jest takvih dinamičkih stranica je performansa u nekim slučajevima gdje se nagomilava puno takve logike. Potrebno je biti oprezan i u svim slučajevima tražiti najbolju praksu. AngularJS je izrazito popularan, ali u zadnje vrijeme ima ozbiljnu konkurenciju u ReactJS-u koji većina programera počinje favorizirati. Na sljedećoj slici vidi se potražnja za AngularJS i ReactJS do 2017.



Slika 2: Objave ponuda za posao u tehnologijama AngularJS i ReactJS (Alexseyenko, 2017)

4.2.2. Početak rada u Ionic-u

Za Ionic se može reći da je početak vrlo jednostavan uz puno gotovih i definiranih komponenata. Ionic dolazi s alatima koji će pomoći programeru da postavi, izgradi i pokrene svoj Ionic projekt te se ti alati nalaze u Ionic komandno-linijskom sučelju (CLI). biblioteke pristupnog dijela i CLI su otvorenog kôda tako da se mogu besplatno koristiti prilikom razvoja aplikacija.

Prva stvar koja se treba instalirati jest Node.js koji je okvir JavaScripta na strani poslužitelja. Koristit će se njegov paket menadžer (NPM) koji će omogućiti da se instaliraju svi paketi i alati za projekte koji će biti obrađeni u radu. Također, koristit će se Node.js za pokretanje usluge razvojnog poslužitelja koji će omogućiti pregled aplikacija u pregledniku. Node.js se jednostavno instalira preuzimanjem instalacije sa službene stranice (https://nodejs.org/en/).

Sljedeći korak je instalacija samog Ionica i Cordove preko odabranog paket menadžera NPM. U nastavku su navedene sve naredbe koje je potrebno izvršiti.

- Naredba za instaliranje posljednje verzije Ionic-a
  `npm install -g ionic@latest`
- Naredba za instaliranje posljednje verzije Cordove
  `npm install -g cordova@latest`

„Ionic start“ naredba će stvoriti novu aplikaciju iz predloška. Mogu se navesti svi predlošci sa sljedećim navedenim opcijama.

`--type` - Koji tip projekta se želi (npr. Ionic-angular, ionic1)

`--display-name` - Ime aplikacije

`--cordova` - Uključuje integraciju Cordove

`--no-deps` - Bez instaliranja npm ovisnosti

`--no-git` - Bez inicijaliziranja novog git repozitorija

`--no-link` - Bez povezivanja aplikacije s web aplikacijom Ionic Dashboard

`--pro-id` - Definirati app ID za povezivanje s Ionic Dashboard-om

`--bundle-id` - Definiranje bundle ID za aplikaciju

Projekt u ovom radu kreiran je sljedećom naredbom:

```
ionic start diplomski-ionic-app sidemenu --type ionic1
```

Mora se spomenuti još jednom kako je početak u Ionic-u vrlo lagan i programeri se vrlo lako snađu s kad krenu s razvijanjem aplikacija. Kada se kreira projekt dobije se već vrlo dobro definirana struktura gdje su sve veće cjeline odvojene, ali opet dostupne. Također, bitno je napomenuti da Ionic 2 ima puno inovativnih značajki te je struktura malo drugačija od

verzije 1. Sljedeća struktura je struktura aplikacije koja se obrađuje u ovom radu i koja je kreirana u verziji 1.

- bower.json - Bower zavisnosti
- config.xml - Cordova konfiguracijski dokument
- gulpfile.js - Gulp zadaci
- ionic.project - Ionic konfiguracija
- package.json - Node zavisnosti
- platforms - iOS/Android folder, sa specifičnim stvarima za svaku platformu
- plugins - Datoteka gdje se nalaze svi instalirani Cordova i Ionic pluginovi
- scss - Scss code
- www - Aplikacija - JS, Css, slike, biblioteke i slično

Druga vrlo bitna značajka kod kreiranja nove aplikacije su starteri (Starters). Starteri su izgrađeni unutar spremišta Ionic Starters preklapanjem aplikacije startera na skup osnovnih datoteka, stvaranja komprimirane arhive datoteka i njihova prijenosa diljem svijeta. Ionic CLI zatim preuzima i ekstrahira arhivu predložaka i prilagođava datoteke za svaku novu aplikaciju (Ionic.com, 2018). Ovo su svi predlošci koji se mogu koristiti:

- Tabs - Početni projekt s jednostavnim sučeljem s tabovima.
- Blank - Prazan projekt.
- Sidemenu - Novi projekt sa standardnim menijem i navigacijom u zaglavlju.
- Super - Novi projekt s unaprijed definiranim stranicama, providerima i najboljim praksama za razvijanje u Ionic-u.
- Conference - Projekt koji demonstrira stvarnu aplikaciju.
- Tutorial - Tutorial projekt koji prolazi kroz bitne dijelove službene Ionic dokumentacije.

4.2.3. Aplikacija i analiza Ionic-a

Za početak je kreirana nova HTML stranica i novi kontroler za stranicu prijave. Novi kontroler je potrebno definirati u index.html kao glavni dokument gdje se uključuju svi ostali dokumenti, kontroleri, razne biblioteke i razne gotove JavaScript skripte.

Nakon toga definira se novo stanje (state) gdje se definira koji će kontroler i HTML stranica biti pozvani kada pristupimo određenom url-u ili stanju.

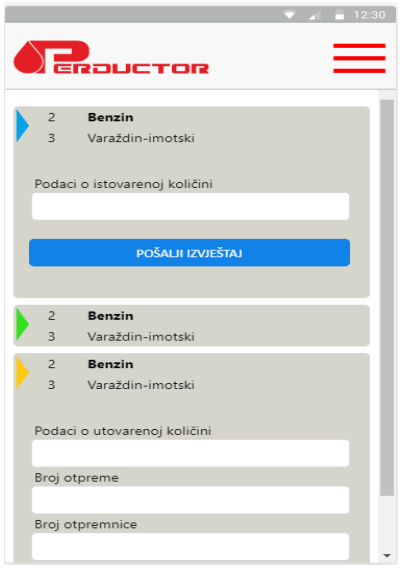
Programski kod 1: Primjer implementacije stanja u Ionic-u

```
.state('bapp.login', {
      url: '/login',
      views: {
      'menuContent': {
            templateUrl: 'templates/login.html',
            controller: 'LoginCtrl'
                  }
            }
      })
```

Struktura je dosta jednostavna, vidi se pozadinska slika, dva unosna polja i tipka za pokušaj prijave.

Slika 3: Stranica prijave

Sljedeća stranica je ispis svih naloga za prijavljenog vozača. Na Slici 4. vidi se da postoji više statusa gdje svaki od tih statusa ima drugačija polja za slanje izvještaja.



Slika 4: Ispis radnih naloga

Prvi korak je bio poziv servisa za dohvat svih naloga prema ID-u ulogiranog korisnika.

Programski kod 2: AJAX poziv u Ionic-u

```
var request = $http({
        method: "POST",
                url:'http://perductor.hr/aplikacija/rest/radninalozi.php',
          data: { driver: window.localStorage.getItem("id") },
          headers: { 'Content-Type': 'application/x-www-form-urlencoded' }

        });
request.success(function (data) {$scope.warrents = data})
```

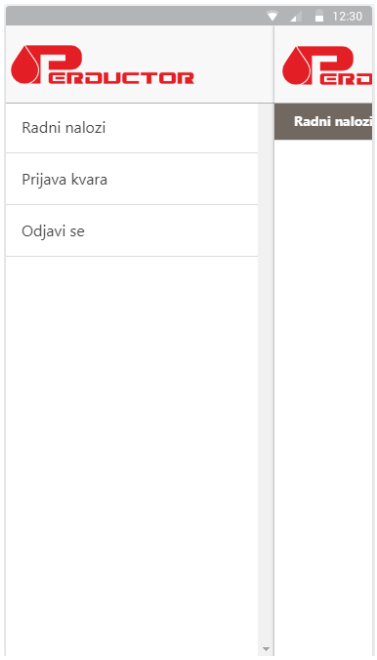
Programski kôd 2. prikazuje najjednostavniji AJAX (Asynchronous JavaScript and XML) poziv gdje se upisuje kojom metodom se šalje, mjesto gdje se nalazi servis, ulazni parametri (ID korisnika) i zaglavlja. Ako je poziv uspješan podaci se spremaju u Angular varijablu.

Drugi korak je prikazivanje liste na HTML stranici. AngularJS-ove direktive “ng” uvelike pomažu oko manipuliranja podataka na HTML dokumentu. Kod prikaza takve liste korišten je *ng-repeat* koji će za određenu kolekciju ponavljati određeni dio HTML kôda.

Programski kod 3: Primjer korištenja Angular direktiva ng-if i ng-repeat

```
<div ng-repeat="warrent in warrents ">
     <div ng-if="warrent.status==3" id="warrent{{warrent.id}}"
ng-click="openWarrentGreen(warrent.id)">
          <img id="image3" src="img/green_status.png" />
               <table style="margin-left:8px">
               <tr>
               <td>{{warrent.cargotype}}</td>
               <td                          style="font-weight:
               bold">{{warrent.orderedquantity}}</td>
               <tr/>
               <tr style="padding-left:-210px">
               <td>{{warrent.orderdate}}</td>
               <td>{{warrent.route}}</td>
               <tr/>
               </table>
          <br/>
          <p>{{warrent.note}}</p>
     </div>
</div>
```

Svaki radni nalog ima svojstvo statusa prema kojem se prikazuje određeni HTML s direktivom *ng-if=”uvjet”.* U ovom isječku kôda vidljiv je primjer za zeleni status gdje su prikazani svi potrebni podaci. “Hamburger” izbornik se dobiva automatski jer je kreiran projekt s opcijom *sidemenu*. Postoje početni elementi koji su zamijenjeni s Radni nalozi, Prijava kvara, Odjavi se.



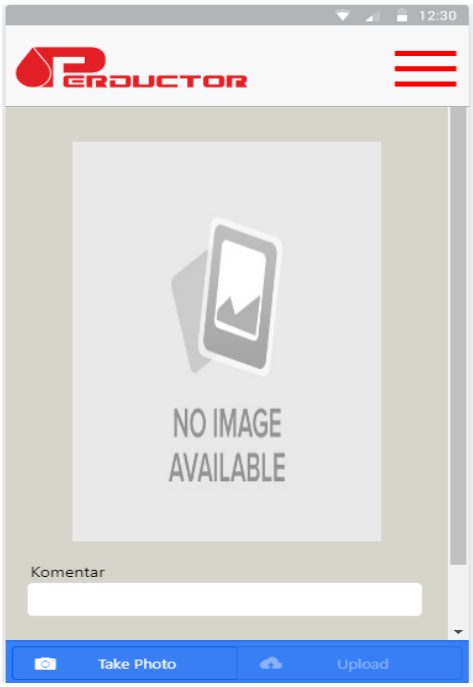
Slika 5: Izbornik u Ionic-u

Zadnja stranica je prijava kvara, odnosno učitavanje slike na server s dodatnim komentarom. Za implementaciju ove značajke bili su potrebni dodaci Cordove kako bi bilo moguće koristiti neke nativne API-je od uređaja kao što su kamera i pristup galeriji slika.

Ovo su dodaci koji su instalirani :

- `ionic plugin add cordova-plugin-file`
- `ionic plugin add cordova-plugin-file-transfer`
- `ionic plugin add cordova-plugin-filepath`
- `ionic plugin add cordova-plugin-camera`

Uz praćenje dokumentacije navedenih dodataka, implementacija učitavanja i slanja slike na server je vrlo jednostavna.



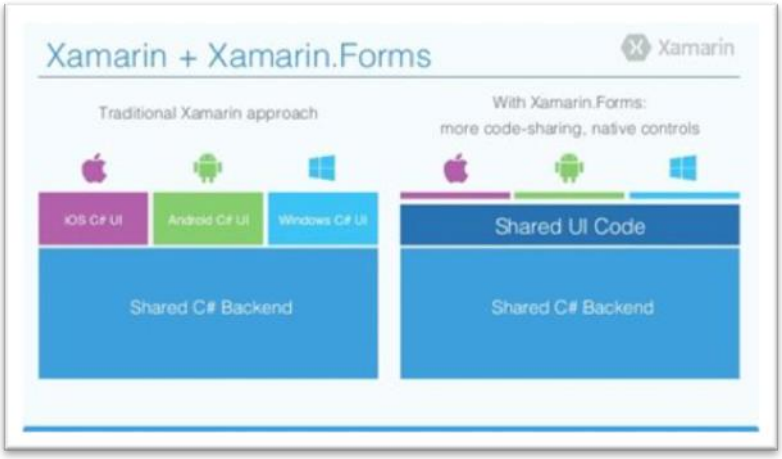
Slika 6: Stranica za učitavanje slike

Ionic je kvalitetno dokumentiran te takva dokumentacija uvelike olakšava rad u samom okviru. Važno je spomenuti da postoje mnoge opcije pri kreiranju samog projekta koje instaliraju već gotove komponente koje će smanjiti potrošeno vrijeme. Uz AngularJS i HTML vrlo je jednostavno kreirati mobilne aplikacije te početnik u Ionic-u za vrijeme kratkog perioda može postati produktivan. Nedostatak performanse u radu ove aplikacije nije primijećen te postoje mnoge komponente koje vrlo dobro izgledaju na mobilnom uređaju. S druge strane, neke od već definiranih komponenta je teže stilizirati te se programeru uvijek otvara opcija za korištenje čistih HTML elemenata. Takav odabir se ne preporučuje jer većina HTML kontrola će izgledati dosta „mrtvo“ i statično na mobilnom uređaju.

Ionic ima daleko više prednosti nego nedostataka te se kontinuirano nadograđuje svakom novom verzijom i to je vrlo pohvalno. Ionic svakako spada u top hibridne okvire što je i ova implementacija pokazala.

4.3. Xamarin

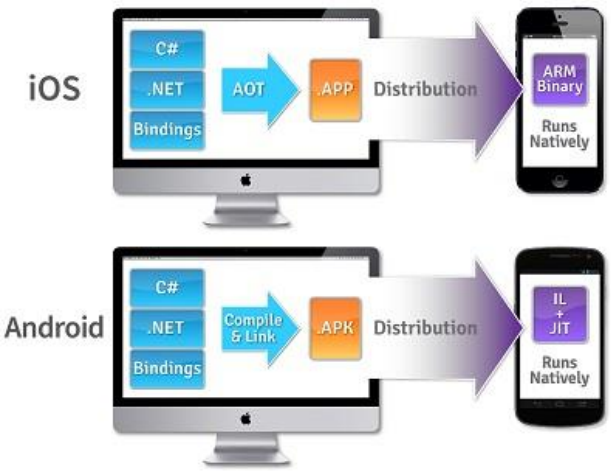
Xamarin je platforma za razvoj hibridnih aplikacija razvijena 2011. godine, a 2016. je napravljena akvizicija od strane Microsoft-a. Bazu čine .NET okvir i Xamarin-ove biblioteke koje služe za kompajliranje u nativni kôd, a programski jezik je C#. Ova platforma podržava Windows, Windows Phone, iOS, Android, te OS X. U posljednje dvije godine zanimanje za Xamarin-om sve više raste. Jezgru Xamarin aplikacije čine bazni kôd koji je zadužen za komunikaciju s operacijskim sustavom, a na njega se naslanja sloj u kojem se vrši poslovna logika. Iznad toga ide specifičan programski kôd za svaku od platformi koja se planira podržati kao i prezentacijski sloj (Microsoft.com, 2016).



Slika 7: Arhitektura Xamarin-a

(Chauhan, 2017)

Platforma Xamarin ima dva dijela. Sastoji se od Xamarin.Android i Xamarin.iOS (ili MonoTouch). Razvoj sliči nativnom razvoju koristeći nativne alate no s obzirom na to da se koristi C#, razvojni programer može ponovno koristiti poslovnu logiku između platformi. Xamarin u osnovi obuhvaća platforme za izradu iOS, OSX, Android i Windows Phone (Microsoft.com, 2016).



Slika 8: Kompajliranje u Xamarin-u

(qode.pro, 2015)

Xamarin se može nazvati kompletnim alatom koji omogućuje razvijanje mobilne aplikacije za više platformi. Ono što Xamarin čini posebnim jest njegova sposobnost da svoje aplikacije prevede u izvršni kôd za iOS, Android i Windows Phone. Za iOS platformu se koristi objective-C ili swift, dok se za Android platformu koristi Java. Ono što čini Xamarin je kombiniranje tih programskih jezika putem IDE Xamarin Studio zbog čega je potrebno kontrolirati C# kako bi se razvila aplikacija na iOS, Android-u i Windows Phone-u te imala neke pojmove svake platforme.

Kada se govori o iOS-u postoji jednostavnija situacija gdje programski kôd jednostavno treba rekompajlirati u machine kôd prije pokretanja jer nema virtualnog stroja, dok je problem kod Androida baš u tome što je nativni jezik Java, što dodatno komplicira stvar. Kad se aplikacija kompajlira to je ustvari pretvoreni C# kôd u posredni bytecode kôd koji je naravno čitljiv za Mono, a zatim se ovaj virtualni stroj također dodaje aplikacijskom APK. Ovo je razlog zašto APK kreiran od strane Xamarin-a zauzima više memorije. Dalvik i Mono je napisan u C-u i vrti se na Linuxu. Drugim riječima, kad aplikacija na Android-u radi, obje virtualke rade i dijele podatke kroz posebne omotače (Chauhan, 2017).

4.3.1. Xamarin.Forms

Xamarin.Forms je biblioteka koja omogućuje stvaranje lokalnog korisničkog sučelja za iOS, Android i Windows Phone iz jedne jedinstvene, dijeljene C# baze podataka. Ona pruža više od 40 međuplatformskih grafičkih kontrola i izgleda koji su preslikani na izvorne kontrole što znači da su korisnička sučelja potpuno nativna. Xamarin.Forms je okvir koji dopušta programerima da brzo stvaraju korisnička sučelja preko platforme. Ona pruža vlastitu apstrakciju za korisničko sučelje koje će biti izvedeno pomoću nativnih kontrola nad iOS-om, Androidom ili Universal Windows Platformom (UWP). To znači da aplikacije mogu dijeliti velik dio njihova korisničkog sučelja i zadržati nativan izgled i dojam ciljane platforme (Hanselman, 2014).

Xamarin.Forms omogućuje brzo implementiranje prototipnih aplikacija koje se s vremenom mogu razvijati u složene aplikacije. Budući da su aplikacije Xamarin.Forms nativne aplikacije, nemaju ograničenja drugih alata kao što su sandbox za preglednik, ograničene API-je ili slabe performanse. Aplikacije napisane pomoću Xamarin.Forms mogu koristiti bilo koji od API-ja ili značajki temeljne platforme kao što su CoreMotion, PassKit i StoreKit na iOS-u; NFC i Google Play usluge na Android-u i pločice na sustavu Windows. Osim toga, moguće je izraditi aplikacije koje će imati dijelove njihovog korisničkog sučelja stvorene pomoću Xamarin.Forms dok su drugi dijelovi izrađeni pomoću nativnog alata za korisničko sučelje (Chauhan, 2017).

Primjene Xamarin.Forms su iste arhitekture kao i tradicionalne među-platformske aplikacije. Najčešći je pristup korištenjem prijenosom biblioteke ili zajedničkih projekata za čuvanje zajedničkog kôda i stvaranje aplikacija specifičnih za platformu koja će iskoristiti zajednički programski kôd (Hanselman, 2014).

Postoje dvije tehnike za stvaranje korisničkih sučelja u Xamarin.Forms. Prva tehnika je stvaranje korisničkog sučelja u cijelosti s izvornim kôdom C#. Druga tehnika je uporaba Extensible Application Markup Language (XAML), deklarativnog označnog jezika koji se koristi za opisivanje korisničkih sučelja. Za više informacija o XAML-u potrebno je proučiti XAML osnove (Hanselman, 2014).

4.3.2. Visual Studio

Microsoft Visual Studio predstavlja revoluciju u razvoju aplikacija i aplikativnih rješenja. Ne postoji niti jedno integrirano razvojno okruženje za developera toliko korisnički prijateljsko (User friendly), unaprijeđeno, kvalitetno i što je najbitnije stabilno kao što je VisualStudio.

Visual Studio je integrirano razvojno okruženje (IDE) skup alata u jednom programu koji pomaže u pisanju programa. Bez Visual Studio-a bilo bi potrebno otvoriti uređivač teksta, napisati sav kôd i pokrenuti kompilator naredbenog retka kako bi se stvorila izvršna aplikacija (May, 2010).

Doslovno se mogu razvijati sve vrste aplikacija, od mobilnih aplikacija, aplikacija za tablete ili čak igrice za Xbox konzole. Visual Studio posjeduje uređivač kôda koji podržava IntelliSense i refaktoriranje kôda. Od ostalih alata obavezno se mora spomenuti GUI, odnosno Form dizajner, dizajner klasa, dizajner sheme baze podataka. Postoji još mnogo značajki koje Visual Studio ima, a mogu se instalirati i razni dodaci (May, 2010).

Visual studio podržava mnoge programske jezike. On omogućuje da uređivač kôda (code editor) i *debugger* podržavaju skoro sve programske jezike. Ugrađeni jezici su C, C#, C++, VB.Net (preko VisualStudio-a). Može podržavati i ostale programske jezike poput Python, Ruby, Node.js itd. Podrška za programske jezike se dodaje pomoću posebnog VS paketa koji se zove *Language Service*. On definira razna sučelja koja VSPackage implementacija može implementirati kako bi podržala razne funkcionalnosti. Funkcionalnosti onda uključuju označavanje sintakse bojom, dovršavanje imenovanja kao i podudaranje zagrada (May, 2010).

4.3.3. Početak rada u Xamarin

Pošto je Xamarin Microsoft-ov okvir može se pretpostaviti da je cijeli taj proces, odnosno krivulja učenja vrlo rastuća. *Community* verzija Visual Studio-a se može preuzeti besplatno sa službene stranice (https://visualstudio.microsoft.com/downloads/).

Instalacija je vrlo jednostavna te kada se uspoređuje s Ionicom vidljivo je kako je to nedostatak kod okvira Ionic. Nakon instalacije kreiranje i početak rada u Xamarin-u je podosta jednostavan. Sljedeći navedeni koraci potrebni za kreiranje novog projekta.

1. Create new Project
2. Cross platform

3. Blank App
4. Create

Xamarin osim *blank template* ima i *master detail template* predloške što je na prvi pogled razočaravajuće, ali ako se posjeti službena stranica Microsoft-a i pronađe Xamarin postoje razni uzorci gotovih aplikacija i kôdova koji mogu uvesti neke značajke Xamarin-a te uvelike olakšati neke standardne implementacije na mobilnoj aplikaciji. Poneke od tih aplikacija su zavidno napravljene i od velike su pomoći. U nastavku su pojašnjeni neki od uzoraka (Microsoft, 2018):

- Todo - todo lista aplikacija gdje su podaci spremljeni na SQLite bazu podataka, izrađena u Xamarin.Forms.
- My Shoppe - demo aplikacija koja omogućuje prodavačima praćenje njihove prodajne izvedbe, pregledavanje kontakata, upravljanje narudžbama i pregledavanje kataloga proizvoda.
- Chat example - ovaj uzorak je demonstracija pokretanje servera i spajanje na isti te jednostavna implementacija dopisivanja.
- Adventure - ovaj uzorak pokazuje kako napraviti jednostavnu 2d igru za iOS.
- Calendar Demo - ovaj primjer pokazuje kako koristiti kalendar API i događaje.
- Circle Map - demo kako napraviti krug distance na mapi oko jedne točke.

Ovo su samo neki uzorci koji su izdvojeni, postoje još mnogi bogati uzorci koji mogu uvelike olakšati izradu aplikacije, pogotovo ako je osoba početnik u Xamarin-u.

Xamarin koristi metodu dijeljenja programskog kôda, cjelokupna struktura rješenja trebala bi implementirati arhitekturu s više slojeva koja potiče dijeljenje kôda. Pristup Xamarin-u je grupiranje kôda u dvije vrste projekta:

Zajednički projekt - Ovo je sloj aplikacije gdje se piše sva dijeljena implementacija koja će se koristiti na svim platformama. Projekti zajedničkog kôda trebaju referencirati samo skupove koji su dostupni na svim platformama. Zajednički projekti trebali bi implementirati što više ne-UI funkcija što bi moglo uključivati sljedeće slojeve (Microsoft, 2018):

- Podatkovni sloj -Sloj koji vodi brigu o fizičkom pohrani podataka, npr. SQLite-NET, alternativnu bazu podataka poput Realm.io ili čak XML datoteka.

- Pristupni sloj podacima - Određuje API koji podržava potrebne operacije podataka za funkcionalnost aplikacije kao što su metode za pristup popisu podataka, pojedinačne stavke podataka te njihovo stvaranje, uređivanje i brisanje.
- Pristupni sloj servisa - Sadrži kôd koji pristupa udaljenim mrežnim resursima (web usluge, preuzimanja slika itd.) i eventualno pohranjivanje rezultata.
- Poslovni sloj - Definicija klase modela i klase fasada ili upravitelja koji izlažu funkcionalnost aplikacijama specifičnim za platformu.

Projekti za aplikaciju specifični za platformu - uključuje potrebne sklopove za vezanje na SDK svake platforme (Xamarin.iOS, Xamarin.Android, Xamarin.Mac ili Windows) kao i zajednički projekt. Na ovoj razini dodane su specifične značajke platforme, izgrađene na komponentama izloženim u zajedničkom projektu (Reynolds, 2014).

Projekti specifični za platformu trebali bi provesti:

- Aplikacijski sloj -Specifične funkcionalnosti za pojedinu platformu i obavezna pretvorba između objekata poslovnog sloja i korisničkog sučelja.
- Sloj korisničkog sučelja - Zasloni, prilagođene kontrole korisničkog sučelja, prikaz logike potvrde.

Potrebno je poznavati osnovne nativne funkcionalnosti svake platforme kako bi se moglo snaći u Xamarin projektu Reynolds, 2014).

4.3.4. Aplikacija i analiza Xamarin-a

Implementacija aplikacije započeta je kreiranjem nove *ContentPage* forme u dijeljenom projektu za prvu stranicu aplikacije, odnosno prijavu korisnika. Nakon toga postavljeni su UI elementi potrebni za stranicu prijave koji su pronađeni na službenoj dokumentaciji.

Prvi problem je nastao već kod kreiranja unosa podataka koji bi trebao izgledati kao standardni HTML-ov element odnosno unosno polje s obrubom. Problem je bio što takav element ne postoji. Rješenje je implementiranje vlastitog elementa koji se neće nalaziti u podijeljenoj verziji, nego će biti implementiran za svaku platformu posebno. Bilo je potrebno napraviti prilagođeno iscrtavanje (*Custom Renderer*) za svaku platformu. Više o tome u sljedećem odlomku.

Takva grafička kontrola u Xamarin-u ima svoje posebno iscrtavanje za svaku platformu gdje se kreira nativna instanca te kontrole. Jedna od značajki alata Xamarin.Forms je sposobnost manipuliranja korisničkim sučeljem pomoću različitih dostupnih API-ja specifičnih za platformu, bilo da mijenja izgled i styling prirodnih kontroliranih elemenata pomoću prilagođenih iscrtavanja (Daniel, 2017).

Cilj prilagođenog iscrtavanja je upravo nadjačavanje tih nativnih kontrola te modificiranje istih za svaku platformu posebno. Pošto je za ovu aplikaciju potrebno standardno HTML unosno polje, element koji se treba nadjačati je *Entry*. Inače je rijetkost u općenitom razvijanju grafičkog sučelja da se ovakvi UI elementi nadjačavaju i da se modificiraju, ali u Xamarin-u je to čest slučaj upravo zbog činjenice da postoji nativan UI za Android i za iOS.

Sljedeći kodovi opisuju kreiranje nove Xamarin.Forms prilagođene kontrole koja će naslijediti *Entry*, ali će imati dodatni obrub oko unosnog polja.

Programski kod 4: Primjer prilagođene kontrole koja nasljeđuje nativnu kontrolu

```
using System;
using Xamarin.Forms;
namespace DiplomskiApp
{
    public class RoundedEntry : Entry
    {
    }
}
```

Nakon što je kreirana može se koristiti na vlastitom XAML dokumentu s XML oznakom:

Programski kod 5: Primjer poziv prilagođenog elementa

```
<local:RoundedEntry />
```

Sljedeći korak je kreiranje prilagođenog iscrtavanja za obje platforme. Primjer će biti pokazan za Android verziju. Iscrtavanja su ustvari klase koje nasljeđuju *EntryRenderer*.

Nadjačana inačica *OnElementChanged* metode u *MyEntryRenderer* klasi je mjesto za obavljanje izvorne prilagodbe kontrole. Tipkovnicom koja se referira na nativnu kontrolu koja se koristi na platformi može se pristupiti preko entiteta kontrole. Osim toga, referenca na kontrolu Xamarin.Forms može se dobiti putem elementa Element iako se ne koristi u primjeni.

Svaka prilagođena vrsta renderera ima atribut *ExportRenderer* koji registrira renderer u Xamarin.Forms.Ovo je primjer iscrtavanja za Android platformu:

Programski kod 6: Primjer renderera za Android platformu

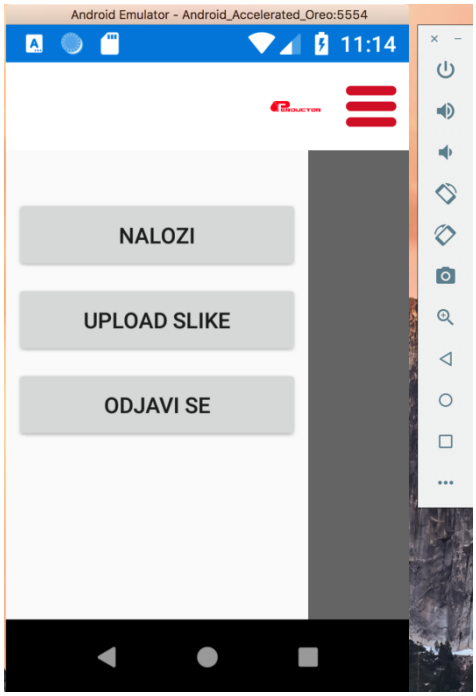
```
[assembly: ExportRenderer(typeof(RoundedEntry), typeof(Rounden
dEntryRendererAndroid))]

     namespace DiplomskiApp
{

     public class RoundendEntryRendererAndroid : EntryRenderer

     {

     public RoundendEntryRendererAndroid(Context context) : ba
se(context)

     {}

     protected override void OnElementChanged(ElementChangedEv
entArgs<Entry> e)

     {
     base.OnElementChanged(e);

     if (e.OldElement == null)

          {
          var gradientDrawable = new GradientDrawable();
          gradientDrawable.SetCornerRadius(10f);
          gradientDrawable.SetStroke(1, Android.Graphics.Color
          .Black);gradientDrawable.SetColor(Android.Graphics.C
          olor.White);

          Control.SetBackground(gradientDrawable);
```

```
            Control.SetPadding(50, Control.PaddingTop, Control.P
            addingBottom, Control.PaddingLeft);
            }
        }
    }
```

Nakon implementiranja vlastitog unosnog polja ovo je izgled stranice za prijavu (Slika 9.)

Slika 9: Stranica prijave u Xamarin-u

Za kreiranje specifičnog Android izbornika potrebno je implementirati *Master-Detail Page*, točnije novu stranicu koja upravlja s dvije povezane stranice gdje prva stranica prezentira sve podstranice koje postoje, a druga stranica prezentira detaljnu stranicu određene podstranice. Ovako izgleda implementirani primjer za ovaj rad.



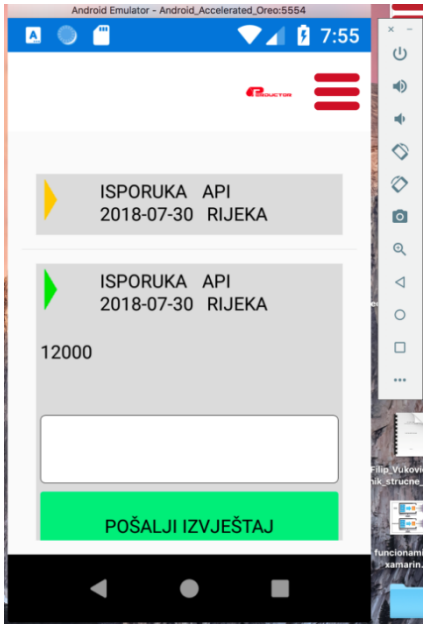
Slika 10: Izbornik u Xamarin-u

Sljedeća stranica je prikaz svih radnih naloga za ulogiranog korisnika. Nakon što je kreirana nova XAML stranica, implementiran je i AJAX poziv za dohvaćanje svih radnih naloga.

Programski kod 7: Primjer AJAX poziva u Xamarinu

```
HttpContent content = null;
var dict = new Dictionary<string, string>();
dict.Add("driver", "1");
var json = SerializeObject(dict);
content = new StringContent(json, Encoding.UTF8, "application
/json");

using (var response = await client.PostAsync("http://perductor
.hr/aplikacija/rest/radninalozi.php", content))
```

Nakon toga napravljena je *Warrant* klasa koja će služiti za spremanje podataka sa servisa. Za prikaz svih radnih naloga na stranici korištena je ListView komponenta gdje je definiran svaki od iteriranih elementa liste radnih naloga, koje će atribute posjedovati te kako će izgledati.



Slika 11: Stranica svih naloga u Xamarin-u

Nakon implementacije prikaza svih radnih naloga, zadnji dio aplikacije je implementiranje slikanja i učitavanja slike. Prvi korak je bio instalacija potrebnog dodatka za samu implementaciju, odnosno Xam.Plugin.Media.Ovaj dodatak omogućuje sve što je potrebno, a to je slikanje i odabir slike te nakon toga učitavanje slike na server što je uvelike olakšalo rad.

Za razliku od Ionic-a, Xamarin se može koristiti IDE-u odnosno u Visual Studio-u. Visual Studio uvelike olakšava osnovne korake od kreiranja projekta do implementacije. Može se reći da nakon instalacije Visual Studio-a programer može krenuti razvijati aplikaciju u Xamarin-u. To je velika prednost Xamarin-a gdje programer može više kontrolirati svoj rad. Također C# je vrlo jak i robustan programski jezik te je također prednost Xamarin-a. Pri početnoj analizi pronađen je opis Xamarin-a na službenoj stranici gdje piše da je ovaj okvir 90% među platformski. Razvijanje aplikacije je pokazalo da taj podatak i nije točan jer pri kreiranju prvog elementa to jest unosnog polja s obrubom, potrebno je bilo napraviti prilagođen element za sve platforme. Takvo unosno polje je osnovni element koji bi ipak trebao biti u zajedničkom projektu. Nadalje u implementaciji je pronađeno još nekoliko takvih elemenata koje je bilo teže modificirati ili ih je potrebno napraviti za svaku platformu posebno. Xamarin je uz taj nedostatak ostavio sličan pozitivan dojam kao i Ionic iako koriste posve različite tehnologije.

Xamarin je bio posljednji hibridan okvir koji će biti obrađen u ovom radu. U sljedećem poglavlju analizirat ćemo Web okvire gdje će biti odabran jedan od okvira pristupnog dijela za detaljno analizu i obradu.

5. Analiza programskih okvira za web aplikacije

5.1. Usporedba programskih okvira za razvoj web aplikacija

Web aplikacije obično se programiraju u jeziku koji podržava preglednik kao što su JavaScript i HTML, budući da se ti jezici oslanjaju na preglednik kako bi se program izvršio. Neke od aplikacija su dinamične što zahtijeva obradu na strani poslužitelja. Druge su potpuno statičke bez potrebe za obradom na poslužitelju. Web aplikacija zahtijeva da web poslužitelj upravlja zahtjevima klijenta, aplikacijskim poslužiteljem za obavljanje traženih zadataka, a ponekad i baze podataka za pohranu podataka.

Za web aplikacije koje će biti namijenjene i za mobilne uređaje daleko su bitniji okviri i tehnologije pristupnog dijela nego pozadinskog. Razlog tomu je jednostavan, mobilni uređaji će samo koristiti tehnologije pristupnog dijela te takve aplikacije moraju biti dinamične dok dohvaćanje podataka odnosno pozadinski dio odlazi u drugi plan. Postoje razni okviri pristupnog dijela koji u posljednjim godinama uvelike povećavaju focus na elemente i komponente namijenjenima mobilnim uređajima. Za analizu odabrani su 3 popularna okvira koji će biti analizira i uspoređenima, a to su: *Bootstrap, Foundation i Skeleton*. U Sljedećem poglavlju će biti detaljno analiziran i implementiran jedan okvir koji će biti odabran od strane sumentora.

Opis kriterija analize i ocjenjivanja je opisan u 4. poglavlju ovog rada.

Tablica 2: Analiza okvira za web aplikacije

	Bootstrap	**Foundation**	**Skeleton**
OPIS	Bootstrap je najpopularniji okvir za HTML, CSS i JavaScript za razvoj responzivnih, mobile-first web stranica (Singla, 2018).	Foundation je skup responzivnih front-end okvira koji olakšavaju dizajniranje lijepih responzivnih. Zaklada je semantička, čitljiva, fleksibilna i potpuno prilagodljiva (Singla, 2018).	Semantic UI je moderan front-end razvojni okvir, baziran na Lessu i jQuery. Ima nježan, suptilan i plosnati izgled koji omogućuje lagano korisničko sučelje (Arsenault, 2018).

PREDNOSTI	- Lagan razvoj - Odličan 3rd party podršku i zajednica - Mala ovisnost o jQuery - StackOverflow - Odličan grid sustav (Singla, 2018).	- Mogućnost prilagodbe stranice kako ne bi izgledala kao ostale stranice napravljene s Foundationom - Fleksibilni gridovi - Widgeti - Dodatni servisi (Singla, 2018).	- Jednostavan - Responzivni gridovi - Učitava samo one komponente koje su potrebne - Lijepo dizajniran - Službena podrška za aplikacije treće strane (Arsenault, 2018).
NEDOSTACI	- Većina stranica napravljena u Bootstrapu izgleda slično - Nije napravljen za - Profesionalno korištenje (Singla, 2018).	- Lošija zajednica - Teže za početnike - Slaba podrška (Singla, 2018).	- Vrlo velika veličina datoteke - Nije za programere početnike / neupoznate s JavaScriptom - Dosta bugova (Arsenault, 2018).
OCJENE	Dokumentacija : 10 Početak : 8 Performansa : 5 Zajednica : 10 GUI :7	Dokumentacija : 7 Početak : 5 Performansa : 7 Zajednica: 6 GUI :10	Dokumentacija : 8 Početak : 5 Performansa : 4 Zajednica: 8 GUI :7

S obzirom na funkcije ne smije se zanemariti da se *Foundation* brzo razvija, ali *Bootstrap* CSS održava svoju masovnu podršku zajednice. Smatra se da svaki pojedini okvir pojednostavljuje cijeli proces razvoja web aplikacija. Smatra se da *Bootstrap* nitko neće skinuti s „trona“ te da će ostali okviri ovog tipa uvijek biti samo alternative *Bootstrap-a.*

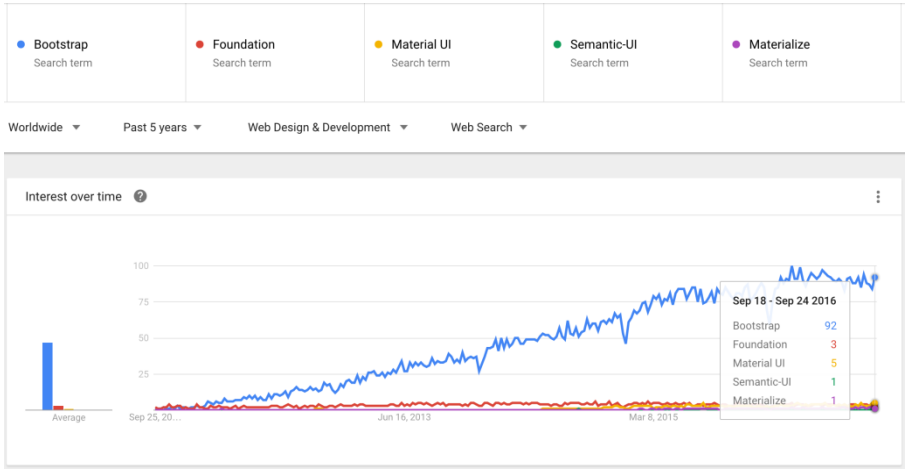
Nakon konzultacija sa sumentorom FINA-om, odlučeno je da će se u ovom radu detaljno analizirati Materialize CSS. Ovaj okvir spada među novije okvire pristupnog dijela kojeg opisuje responzivan dizajn koji se temeljeni na Material Design-u.

5.2. Materialize CSS

Materialize CSS je responzivan okvir pristupnog dijela koji pruža unaprijed definiranu listu stiliziranja koja podosta olakšava posao što se tiče responzivnosti web stranica gdje su potrebna elementarna znanja o HTML, CSS i JavaScriptu. Ovaj okvir se temelji na Google-vom dizajn jeziku Material koji će biti opisan u sljedećem nastavku rada. Ukratko rečeno, Materialize CSS prati koncepte i principe Material Design-a uvođenjem fizike, prostora, svjetlosti i pokreta u dizajn. Drugim riječima, osim širine i visine svaki element ima i dubinu (Materializecss.com, 2018).

Materialize dolazi s CSS-om i okvirom jQuery zajedno sa skupovima komponenata i ikona. Slično kao i Bootstrap, Materialize koristi sustav s 12 mrežnih stupova kao temelj za oblikovanje responzivnih elemenata. Raznolikost komponenata korisničkog sučelja nije bogata kao u Bootstrap-u, ali postoji nekoliko komponenata koje Bootstrap ne pruža, poput paralakse i tosta. Kada se govori o kompatibilnosti preglednika, Materialize radi sa svim modernim preglednicima, uključujući Internet Explorer 10+ (Materializecss.com, 2018).

Materialize je temeljni sustav koji funkcionira na svim platformama i svim uređajima, brine se za sve: osnovni stil, prilagođene komponente, animacije i prijelaze. Ovaj je okvir još uvijek prilično nov, ali vrlo obećavajući alat koji će ubrzati razvoj i poboljšati korisničko iskustvo. Dodatne informacije mogu se pronaći na Web stranici tvrtke Materialize.



Slika 12: **Objave natječaja za posao prema okvirima**
(Arsenault, 2018)

5.2.1. Material Design

Material Design je dizajn jezik napravljen 2014. godine od strane Google-a. Iako je njegova prvobitna namjera bio dizajn za mobilne uređaje, može se koristiti i kao dizajn za Web. Material Design se može upotrebljavati na svim novijim verzijama Androida, odnosno od verzije API 21 do najnovijih. Važno je spomenuti da kako se nove verzije stvaraju tako i Google nadograđuje Material Design za sve svoje platforme i aplikacije.

Google je postavio vrlo rigorozne specifikacije za Material Design te postoji podosta težih riječi za razumjeti, ali je napravljen za Android kako bi se što više poboljšalo i unaprijedilo korisničko sučelje. Material Design se fokusira na fizičke materijale, a ti su materijali digitalni, odnosno oni su elementi kreirani pikselima. Korisnici ih mogu pritisnuti, skrolati, povući itd. Svaki takav materijal je zaseban, može se nazvati objektom unutar digitalnog svijeta. Dakle, aplikacija sa samo jednom tipkom je aplikacija koja sadrži jedan materijal ili se može odabrati kompleksniji primjer gdje je više materijala u aplikaciji povezano, te čine jednu komponentu. Za Material Design se može reći da su njegovi glavni ciljevi upravo upravljanje i unaprjeđenje tih korisničkih interakcija. Neke od bitnih karakteristika koje se dotiču upravo tih interakcija i kojim Material Design upravlja su kretanje, dubina, svjetlost, sama hijerarhija sadržaja (Mew, 2015).

5.2.2. Početak rada u Materialize-u

Materialize dolazi u dva različita oblika uz mogućnost odabira verzije koja se želi instalirati ovisno o želji, stručnosti i preferenciji prema tehnologiji od strane programera. Kako bi se počela upotrebljavati Materialize potrebno je samo preuzeti jednu od sljedećih verzija.

- Materialize - Standardna verzija s CSS-om i JavaScriptom.
- Sass - Ova verzija sadrži sve potrebe SCSS dokumente te je moguće samostalno odlučivati koje komponente okvira se žele uz što je potreban i Sass kompajler.

Također, osim manualnog preuzimanja moguće je koristiti i paket menadžere NPM ili Bower.

U ovom radu korištena je prva verzija, odnosno CSS verzija. Projekt je kreiran s NPM-om. Nakon preuzimanja je raspakiran u direktorij gdje se nalaziti i Web stranica. Ovo je jednostavan okvir pristupnog dijela, stoga je i struktura vrlo jednostavna.

```
webStranica/
        css
        • materialize.css
        js
        • materialize.js
        login.html
        warrent.html
        upload.html
```

Ovo je vlastiti primjer povezivanja svih potrebnih stvari kako bi se mogao koristiti okvir.

Programski kod 8: Inicijalizacija projekta u Materialize-u

```
<html>
      <head>
        <link
        href="https://fonts.googleapis.com/icon?family=Material+
        Icons" rel="stylesheet">
        <link type="text/css" rel="stylesheet"
        href="css/materialize.min.css"
        media="screen,projection"/>
        <meta name="viewport" content="width=device-width,
        initial-scale=1.0"/>
      </head>
        <body>
        <!--JavaScript at end of body for optimized loading-->
        <script type="text/javascript"
        src="js/materialize.min.js"></script>
      </body>
</html>
```

Na službenoj stranici postoje samo dva predloška koja se mogu koristiti. Ovaj podatak na prvu zvuči premalo no potrebno je spomenuti kako je ovo jednostavan okvir gdje će se uz osnovno znanje HTML-a, JavaScript-a i CSS-a lako naučiti koristiti i ovaj okvir. Sljedeća dva predloška sasvim su zadovoljavajuća za korištenje i postavljanje Materialize-a.

- Starter Template –Ovo je najjednostavnija početna stranica sa značajkom zaglavlja, poziva na radnju i ikonu.

- Parallax Template – Ovo je također najjednostavnija početna stranica sa značajkom zaglavlja, poziva na radnju i ikonu, ali uz korištenje Parallaxa.

5.2.3. Aplikacija i analiza Materialize CSS-a

Prvi korak kod web aplikacije je bio postavljanje servera. Za potrebe izrade aplikacije istraženo je kako napraviti, odnosno dobiti besplatne usluge poslužitelja. Registrirana je besplatna domena na adresi https://www.000webhost.com. Ovdje se potrebno samo registrirati te je server spreman za korištenje i idealan je za ovakve testne aplikacije. Aplikacija koja se obrađuje u ovom radu se nalazi na adresi https://filipvukoviccc999.000webhostapp.com/.

Kao što je vidljivo prva stranica je stranica prijave gdje su namjerno korištena stilizirana unosna polja od Materialize-a, a ne standardna koja su na svim ostalim aplikacijama. Razlog tomu je upravo taj da se vidi što više Material Design elemenata prikazanih na mobilnom uređaju.

Ako programer ima iskustva s Bootstrap-om lako će se prilagoditi Materialize-u. Grid sustav ima 12 stupaca bez obzira na rezoluciju ekrana. CSS klase koje su korištene stranici prijave su sljedeće:

`Col` – označava da je ovaj div stupac.

`s10`– ovaj div će biti širok 10 stupaca na malim ekranima.

Push i pull klase služe kako bi se mogao promijeniti redoslijed stupaca. Jednostavno se dodaje push-s2 ili pull-s2 u klasu gdje "s" označava prefiks klase zaslona (s = mali, m = srednja, l = velika), a broj iza toga je broj stupaca koji se žele gurati ili povući.

`pull-s1 m6` – za male ekrane gurnut će za jedan stupac i za srednje ekrane šest stupaca.

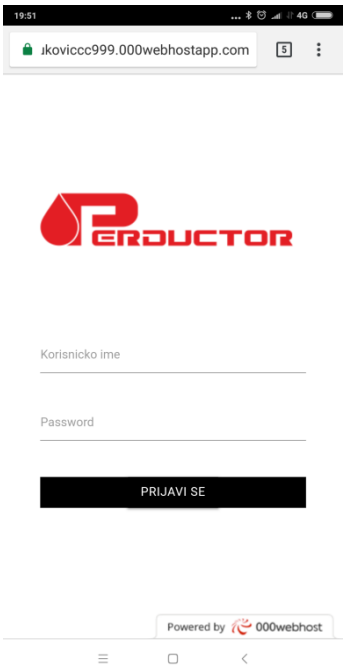
`pull-m3 l4`– za srednje ekrane gurnut će za jedan stupac i za velike ekrane šest stupaca.

Za unosna polja korištena su standardna Design Material i Materialize unosna polja gdje se klikom na unosno polje te dobivanjem fokusa na unosno polje labela pomjera iznad polja. Sljedeći kôd je primjer kôda za dobivanje takvog unosnog polja.

Programski kod 9: HTML elementi za implementaciju stranice prijave

```
<div class="input-field col s12">
<label for="email">Korisničko ime</label>
<input type="email" class="validate" name="email" id="email"
/>
</div>
```

Ovako izgleda stranica prijave aplikacije koja se obrađuje u radu (Slika 13).



Slika 13: Stranica prijave u Materialize-u

Pošto se Materialize temelji na jQuery -u, kao JavaScript okvir nije se želio koristiti neki drugi okvir koji je više namijenjen za manipulaciju podataka nego jQuery koji je namijenjen više za DOM manipulaciju. Ovdje se posebno misli na iteraciju i prikaz svih radnih naloga s API-ja na našoj stranici koji su uz to svi interaktivni.

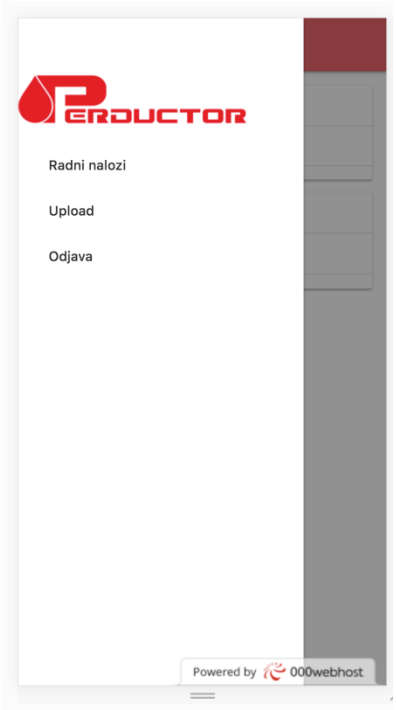
Sljedeći korak je bio kreiranje izbornika gdje u dokumentaciji postoji primjer kako napraviti klasičan Android izbornik sa strane. Takav izbornik se lako definira s već kreiranim komponentama, kada dodamo CSS klasu *sidemenu* na neki div dovoljno je samo da se nad tom klasom instancira izbornik sa sljedećim kôdom.

Programski kod 10: Inicijaliziranje izbornika u Materialize-u

```
$(document).ready(function(){
```

```
    $('.sidenav').sidenav();
});
```

S modificiranjem izbornika nije bilo problema s obzirom na okvir te se kao i na prijašnjim aplikacijama samo dodalo potrebne linkove u izbornik.



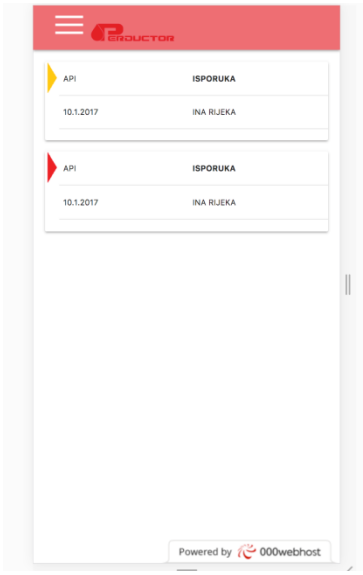
Slika 14: Izbornik u Materialize-u

Sljedeći korak je implementacija radnih naloga gdje je već spomenuto da će biti dosta teža implementacija s jQuery-jem koji nema direktive kao npr. AngularJS, koje olakšavaju manipulaciju podacima i HTML-om. Sljedeći kôd je prikaz API poziva te pri dobivanju odgovora se iterira kroz svaku stavku liste koje smo dobili.

```
$.ajax({
      type: "POST",
      url: "http://perductor.hr/aplikacija/rest/radninalozi.php
      ",
      data: {driver:1}
      success: function(data){
              $.each(data, function( index, value )
              {
       });
}});
```

Programski kod 11:Primjer AJAX poziva u Ionic-u

Sljedeći zadatak je kreirati HTML za prikaz radnog naloga za svaki nalog u listi. Cijeli HTML je kopiran u jednu varijablu s podacima s API-ija i na kraju dodan u dokument. Time su dobiveni svi radni nalozi kao što je bilo zadano.



Slika 15: Stranica radnih naloga u Materialize-u

Zadnji dio aplikacije je omogućiti slanje slike koja se dobije korištenjem mobilnog fotoaparata ili otvaranjem galerije slika te odabir jedne od slika. Postojala je doza skeptičnosti prema ovom zadatku zbog mišljenja da se takve stvari trebaju programirati dodatno u nekom prikladnom programskom jeziku. Nakon istraživanja, pronađeno je da su takve stvari već implementirane u HTML-u. Ovo je službena HTML5 stranica gdje se mogu pronaći sve HTML mogućnosti na mobilnom uređaju:http://mobilehtml5.org/.

Implementacija za vlastiti zadatak je doslovno jedna linija kôda.

Programski kod 12: Implementacija otvaranje fotoaparata

```
<input type="file" accept="image/*">
```

Ovom linijom kôda će se automatski otvoriti odabir za galeriju ili otvaranje fotoaparata te se nakon odabira dobiva dokument u inputu s kojim se dalje lako manipulira.

Izgled implementirane aplikacije u Materialize CSS-u je vrlo pohvalan. Razlog tomu je upravo što se ovaj okvir temelji na Material Designu koji stvara vrlo lijep i nativan izgled i dizajn. Postoje mnogi elementi koji su vrlo slični nativnoj verziji tih elemenata. Prvi primjer unosno polje koje je identično Android-ovom nativnom elementu gdje labela mijenja poziciju s obzirom na fokus unosnog polja. Međutim, postoji i jedna vrlo loša strana ovog okvira. Loša strana je zajednica i održavanje samog okvira. Postoje mnoge loše prakse i neshvaćene greške koje su se javljale u ovom okviru. Instalacija ovog okvira je teoretski jednostavna gdje je potrebno uključivanje 3 skripte u glavnu HTML stranicu, ali postoje mnoge greške i nekompatibilnosti između verzije jQuery-a i Materialize-a. Nadalje ista greška se javila prilikom instalacije izbornika, gdje odabrana verzija Materialize-a nije bila kompatibilna s verzijom izbornika koji je preuzet sa službene stranice. Ideja Materialize je vrlo dobra, ali treba još mnogo poraditi na osnovnim stvarima okvira i podrške.

Nakon obrade okvira za Web aplikacije slijedi analiza 3 programskih jezika za implementaciju nativnih aplikacija. Za razliku od Web aplikacije u nativnoj aplikaciji moramo razvijati i programirati za sve platforme. Nativnost aplikacije s druge strane omogućava bolju performansu i nativan UI, više o nativnim aplikacija u sljedećem poglavlju.

6. Analiza programskih jezika za razvoj nativnih aplikacija

6.1. Usporedba programskih okvira za razvoj web aplikacija

Kada se kaže da aplikacija radi nativno na nekom uređaju, bilo to mobitel, računalo ili neki drugi uređaj to bi značilo da aplikacija radi na uređaju bez ikakvih posrednih slojeva između aplikacije i uređaja. Iako se nativne aplikacije ne odnose samo na mobilne uređaje ipak u ovom kontekstu se najviše taj pojam pojavljuje. Za računala, aplikacije možemo pisati u Javi koje će se izvoditi na svim operacijskim sustavima u kojima postoji JVM (Java Virtual Machine). Problem dolazi kod Apple-ove platforme iOS koji ga ne podržava. Dvije glavne mobilne platforme su Apple-ov iOS i Google-ov Android. Native aplikacije napisane su u kôdu koji se koristi uređaj i njegov operativni sustav. Na primjer, programeri razvijaju aplikacije za iOS u Objective C-u ili Swift-u, dok se aplikacije za Android razvijaju većinom u Javi.

U ovom radu bit će implementirana nativna aplikacija za platformu Android. Android platforma je odabrana iz razloga jer je daleko popularnija od iOS platforme te aplikacija za Android ima mnogo više. Odabrani jezici za analizu su Java, Kotlin i C++. Java i Kotlin su službeni jezici Android platforme te su ta dva programska jezika bila logičan odabir. Treći programski jezik je C++ u kojem se može raditi nativne aplikacije zahvaljujući NDK-u

Android Native Development Kit (NDK) je alat koji omogućuje razvojnim programerima ponovnu upotrebu kôda pisanih u C / C ++ programskim jezicima i ugradnju u njihovu aplikaciju putem *Java Native Interface* (JNI). U sljedećem poglavlju slijedi analiza navedenih programskih jezika za implementaciju nativne aplikacije za Android (Google, 2018).

Opis kriterija analize i ocjenjivanja je opisan u 4. poglavlju.

Tablica 3: Analiza okvira za nativne aplikacije

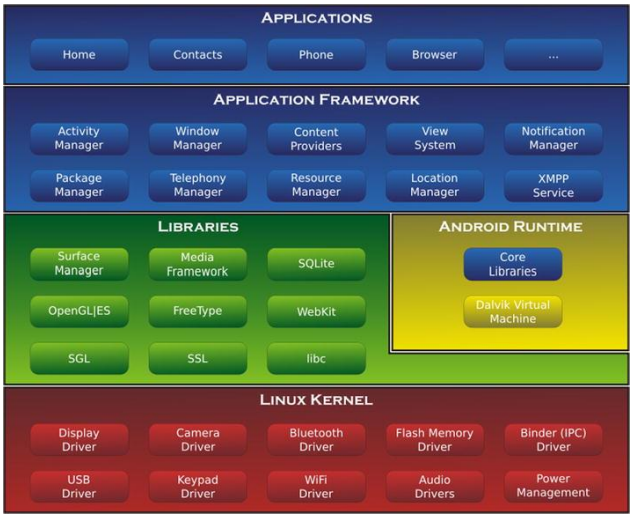
	Kotlin	Java	C++
OPIS	Kotlin je izvrsno prilagođen razvoju Android aplikacija, donoseći sve prednosti suvremenog jezika Android platformi bez uvođenja novih ograničenja. (Google.com, 2018)	Java je prvi službeni jezik androida. Android se snažno oslanja na temelje Jave. Android SDK uključuje mnoge standardne Java biblioteke. (Google.com, 2018)	C + + se može kompajlirati za Android i mogu se proizvoditi Android aplikacije za nativne aktivnosti. Platforma koristi CLANG alatnu traku prilikom sastavljanja za Android (Zuniga, 2016).
PREDNOSTI	• Povećava produktivnost tima • Kompatibilan s postojećim kôdom • Pouzdan • Službeni jezik android • Sigurnosne značajke (Google.com, 2018)	• Službeni jezik android • Nastao prije 19 godina • Zajednica • Podrška (Schildt, 2014)	• Brže kôdiranje • Visual Studio • Postojeći kôd za igre • Performansa (Zuniga, 2016)
NEDOSTACI	• Sporije kompajliranje • Mala Zajednica • Nov jezik (Google.com, 2018)	• Brzina • Low-level programiranje nije podržano (Schildt, 2014)	• Nije službeni jezik androida • Slaba podrška za Android • Težak (Zuniga, 2016)
OCJENE	Dokumentacija : 7 Početak : 8 Performansa : 8 Zajednica: 8	Dokumentacija : 10 Početak : 10 Performansa : 8 Zajednica: 10	Dokumentacija : 5 Početak : 5 Performansa : 10 Zajednica: 5

Iako Kotlin nudi mnoge prednosti koje Java nema on još uvijek ima neke nedostatke. Kad se dopusti vlastitom timu da eksperimentira s Kotlin-om potrebno je istaknuti da ne zaborave da prijelaz na novi jezik nije uvijek uzbudljiv programerima koji su već pronašli alate i strategije za njih. Također, smatra se da je potreban jedan period upoznavanja i prilagodbe s Kotlinom. S druge strane, mora se imati na umu da će Java ostati neophodna za razvoj Android aplikacija.

6.2. Java i Android Studio

Postoji više načina za kreiranje nativnih mobilnih aplikacija, kako za Android, tako i za iOS. Za nativnu aplikaciju odabrana je Android platforma jer je činjenica da Android dominira mobilnim operacijskim sustavima. Drugo pitanje bilo je na koji će se način implementirati, točnije u kojem jeziku. Odabrana je preporučena opcija, odnosno Java i Android Studio.

Vrlo bitna činjenica je da je Android u vlasništvu Google-a koji je kupio Android 2005. godine. Android operacijski sustav se temelji na Linux jezgri. Android OS je tehnologija otvorenog kôda koja se proteže na preko 400 milijuna uređaja diljem svijeta. Ova tehnologija sastoji se od različitih komponenata koje proizvođačima uređaja omogućuju samostalan rad. Arhitektura se razvrstava na pet primarnih dijelova: aplikacija, okvir aplikacije, izvornih biblioteka, Android runtime i Linux kernela kao što je prikazano na sljedećoj slici (Burnetter, 2011).



Slika 16: Arhitektura Android platforme

(Burnetter, 2011)

6.2.1. Android Studio

Android Studio je službeni IDE za Android. Svrha je za Android da ubrza razvoj i da pomogne izraditi najkvalitetnije aplikacije za svaki Android uređaj koji se temelji na IntelliJ IDEA. Android studio može se besplatno preuzeti i koristiti za razvijanje vlastitih aplikacija.

Android studio zapravo pruža programerima razne alate kako bi se mogle izgraditi aplikacije i rješenja za sve Android uređaje gdje spadaju Android Wear, Android Auto, Android TV, naočale i naravno pametne telefone i tablete (Google, 2018).

Neophodno je spomenuti da Android Studio svojim značajkama pomaže pri cijelom procesu razvoja aplikacije. Postoje mnogi alati koji programeru mogu olakšati programiranje, a ovo su neki od najbitnijih Google (2018):

- Instantno pokretanje - omogućuje automatsko prihvaćanje napravljenih promjena i u kôdu i u resursima u aplikaciju koja je već pokrenuta.
- Inteligentan kôd editor - Editor predlaže programeru rješenja problema, bolju praksu, podcrtava napisane pogreške, upozorenja itd.
- Kvalitetan emulator - Uz sve značajke stvarnog uređaja, postoje mnogi alati za debugiranje i izazivanje nekih akcija koje želimo testirati.
- Predlošci aplikacija - Postoje mnoge mogućnosti za korištenje gotovih implementacija koje se često pojavljuju, u našem radu vidjeli smo korištenje izbornika.
- Alati i okviri za testiranje - Postoje mnogi okviri koji olakšavaju testiranje kao što su Mockito, Espresso i UIAutomator.
- Layout Editor - Ovaj editor nam omogućuje pregled grafičkih pregleda i mijenjanje elemenata bez pokretanja aplikacije.
- Vector Asset Studio - Pomaže dodati ikone i uvesti datoteke skalabilnih vektorskih grafičkih (SVG) i Adobe Photoshop dokumenata (PSD) u projekt kao vektorske izvore
- APK Analizer - pruža uvid u strukturu APK-a nakon dovršetka postupka izgradnje

6.2.2. Java

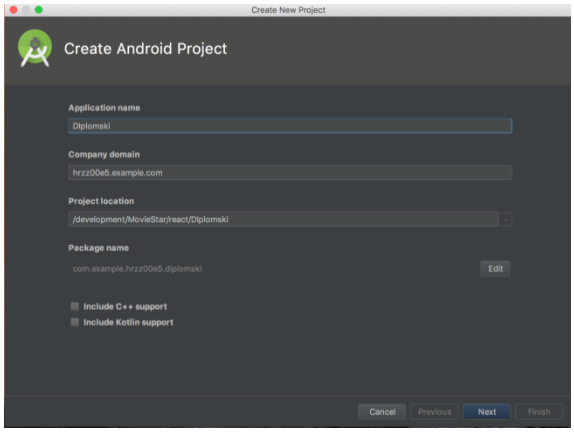
Iako je Java jezik koji može napraviti software za više platformi, velika prednost u odnosu na većinu jezika je to što se programi napisani u Javi bez problema izvode na svim operacijskim sustavima za koje postoji JVM (*Java Virtual Machine*). S druge strane, nedostatak je što to nije slučaj za iOS. Kada se napiše Java aplikacija, kompajlirani kôd, odnosno bytecode se izvršava na operacijskim sustavima. Za razvijanje aplikacija u Javi potreban je SDK (*Java software development kit*) koji uključuje kompajler, interpreter i ostale alate za razvijanje aplikacije. Uvelike ubrzava rad ako se radi s nekim IDE-om (*Integreted development environments*). Pošto se u ovom radu izrađuje Android aplikacija, odabran je Android Studio o kojem će se više pričati u sljedećem podnaslovu (Schildt, 2014).

6.2.3. Početak rada u Android Studio-u

S obzirom na to da je Android Studio službeni IDE za Android to može reći koliko je zapravo moćan i koliko pomaže pri razvijanju aplikacija. Kada se instalira Android Studio, programer je spreman za rad, naravno uz pretpostavku da ima instaliranu Javu. Kao što je već spomenuto, programski jezik je Java koji je i službeni jezik Android-a. Za izradu sučelja koristi se XML koji ponekad može stvarati problem ako programer nema iskustva. Također, ovaj projekt se temelji na Gradle-u. Gradle je alat, odnosno automatizirani sustav za izgradnju aplikacija. Temelji se na Apache Ant-u i Apache Maven-u te je vrlo bitno reći da koristi *Groovy* jezik za konfiguraciju umjesto XML-a kao Maven (Google, 2018).

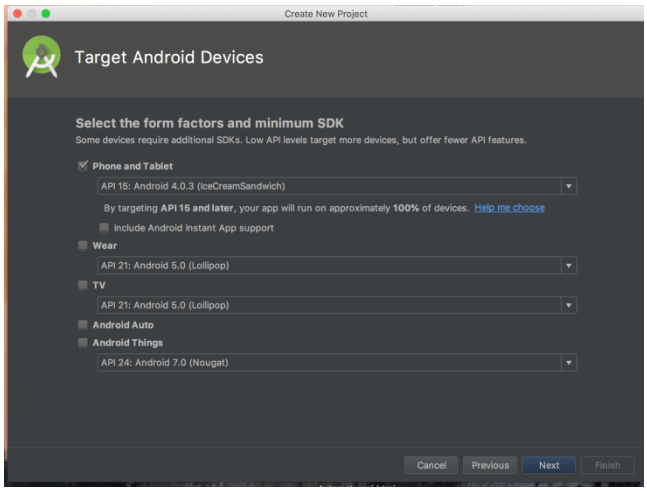
Pri kretanju učenja izrade Android aplikacije s Android Studio-m i kod samog razvoja uvelike će pomoći njihova ogromna zajednica i vrlo bogate dokumentacije. StackOverflow je pun raznih pitanja o svim područjima koja su pokriveni razvojem Android aplikacija s Android Studio-om, što je vrlo bitno.

Sad će se krenuti s kreiranjem novog projekta za aplikaciju. U glavnom izborniku može se pokrenuti čarobnjak za kreiranje novog projekta.



Slika 17: Kreiranje projekta - 1. Korak

Nakon postavljanja projekta, postoji mogućnost odabira verzije Android-a za svaki uređaj koji se želi koristiti. Pošto će ova aplikacija biti samo za pametne telefone, odabran je telefon i tablet. Odabrana je verzija 15 kao najstarija verzija koju će ova aplikacija podržavati.



Slika 18: Kreiranje projekta - 2. Korak

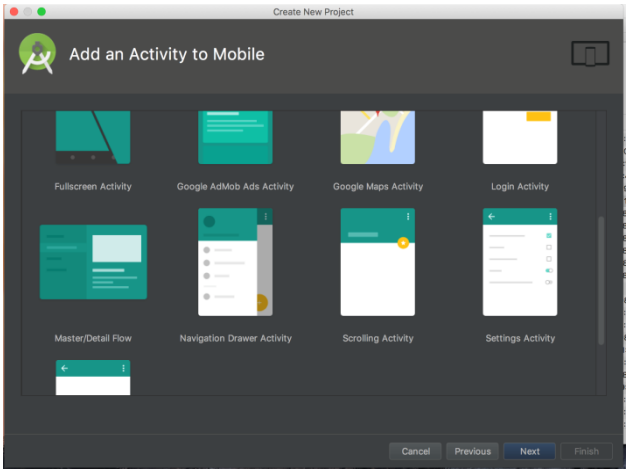
Već je ranije spomenuta bogata zajednica i bogata dokumentacija gdje također postoje mnogi već gotovi primjeri, aktivnosti i predlošci koji se mogu iskoristiti pri izradi aplikacije te će uvelike pomoći u izradi i uštedjeti mnogo vremena. Poneki od njih se nalaze i u Android Studio-u te se mogu odabrati pri kreiranju novog projekta (Google, 2018):

Login Activity - Ovaj predložak kreira standardnu stranicu prijave korisnika s unosom e-maila i lozinke.

Master/Detail Flow - ovaj primjer se mogao vidjeti kod izrade aplikacije u Xamarin-u gdje postoji lista podataka te se klikom na pojedini element iz te liste odlazi na detaljni prikaz tog elementa.

Navigation Drawer Activity - Ovo je predložak koji je korišten u vlastitoj aplikaciji gdje se dobiva standardni izbornik s navigacijskom trakom.

Settings Activity - Ovaj Activity prikazuje korisničke postavke za aplikaciju te proširuje PreferenceActivity.

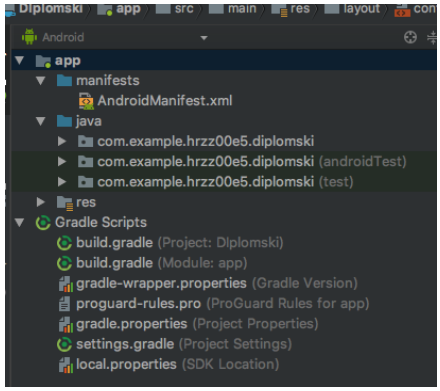


Slika 19: Kreiranje projekta - 1. Korak

Android aplikacija je napravljena od Android modula i Gradle skripta. Modul je skup izvornih datoteka i postavki projekta gdje se odvajaju razne funkcionalnosti u module. Dobra je praksa odvajati razne dijelove aplikacije u module poput povezivanje s bazom podataka itd. Također, mogu se konfigurirati ovisnosti jednog modula prema drugom gdje bi se ako postoji modul baze podataka taj modul koristio u drugim modulima gdje bi se spremali podaci u bazu podataka. Svaki Android modul ima sljedeće elemente:

- Manifest - Sadrži AndroidManifest.xml.
- Java - Sadrži sav Java izvorni kôd s testnim klasama.
- Res - Sadrži sve ostale resurse kao što su to slike, UI stringovi, XML izglede itd.

Uz module postoje i Gradle Scripts gdje se nalaze sve postavke i sve uključene ovisnosti koje su potrebne u projektu.



Slika 20: Struktura projekta u Android Studio-u

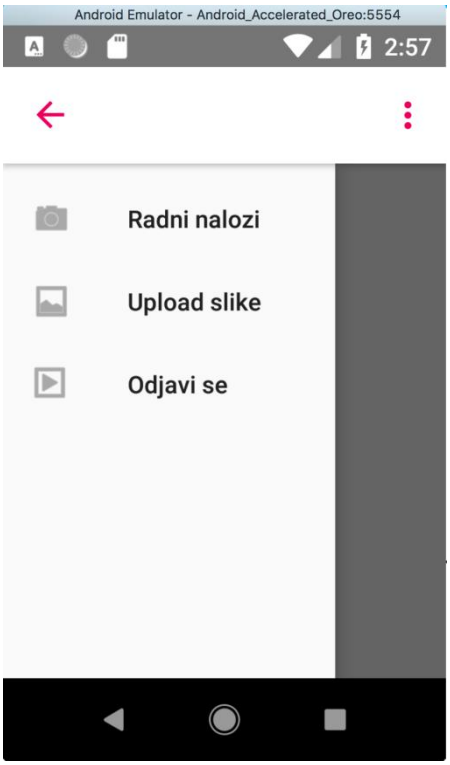
6.2.4. Aplikacija i analiza u Android Studio-a

Klasa Activity ključna je komponenta Androidove aplikacije, a način pokretanja i sastavljanja aktivnosti temeljni je dio modela aplikacije platforme. Za razliku od programskih paradigmi u kojima se aplikacije pokreću s glavnom metodom, Android sustav inicira kôd u primjeru Aktivnosti pozivajući se na određene metode povratnog poziva koji odgovaraju određenim fazama svog životnog ciklusa.

Kod kreiranja projekta postoji jedna, odnosno glavna aktivnost gdje se nalazi izbornik. Budući da prijava korisnika ne smije imati izbornik kreirana je nova aktivnost za prijavu.

Slika 21: Stranica prijave u Android Studio-u

Iako je odabran Navigation Drawer Activity kako bi se dobio gotov izbornik i navigacijska traka, bilo je potrebno dosta toga ukloniti kako bi se dobio sljedeći čisti izbornik s 3 vlastita elementa. U MainActivity klasi nalazi se slušač za izbornik gdje se klikom otvara novi fragment za pojedini link u izborniku. Fragment predstavlja ponašanje ili dio korisničkog sučelja u FragmentActivity. Može se kombinirati više fragmenata u jednoj aktivnosti kako bi korisničko sučelje bilo izgrađeno s više okna i ponovno se upotrijebio fragment u više aktivnosti.



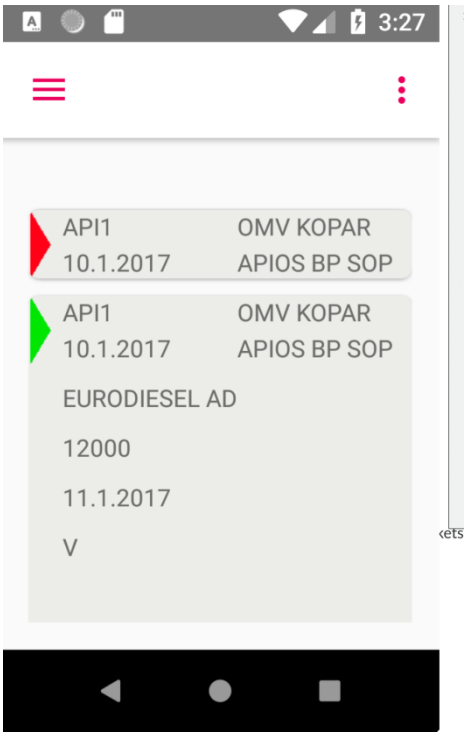
Slika 22: Izbornik u Android Studio-u

Sve naloge kao i do sad treba prikazati u listi. Prema Android dokumentaciji potrebno je koristiti RecyclerView sa CardView-om pošto će elementi ove aplikacije biti dodatno modificirani.

U modelu RecyclerView nekoliko različitih komponenata radi zajedno kako bi se prikazali naši podaci. Opći kontejner za naše korisničko sučelje je objekt RecyclerView koji se dodaje u izgled. RecyclerView se ispunjava pregledima koje pruža upravitelj izgleda. U listu dodajemo podatke dobivene preko poziva API-ija, a sljedeći kôd pokazuje kako to izgleda u Javi:

Programski kod 13: Primjer Ajax poziva

```
String Request stringRequest = new
StringRequest(Request.Method.POST,
"http://perductor.hr/aplikacija/rest/radninalozi.php", new
Response.Listener<String>() {
@Override
public void onResponse(String response) {
     System.out.println("TUUUU"+response);
     try {
          JSONArray jsonArr = new JSONArray(response);
          List<RadniNalog> novaLista= new ArrayList<>();
          //puni listu i adapter
          RVRadniNalogAdapter adapter = new
          RVRadniNalogAdapter(novaLista,getContext());
          rv = (RecyclerView) rootView.findViewById(R.id.rv);
          LinearLayoutManager llm = new
          LinearLayoutManager(getActivity());
          rv.setLayoutManager(llm);
          rv.setAdapter(adapter);

     } catch (JSONException e) {
     e.printStackTrace();
          }
     }
}
```



Slika 23: Stranica radnih naloga u Android Studio-u

Na stranici https://github.com/kosalgeek/PhotoUtil pronađena je biblioteka koja će olakšati implementaciju fotografiranja ili odabira slike iz galerije te prijenos te iste slike na server. Nakon što je skinut PhotoUtil.jar dodan je u biblioteke u projekt. Također, dane su dozvole za korištenje kamere dodavanja podataka u samu memoriju mobitela. To je napravljeno dodavanjem sljedećih dozvola u AndroidManifest.xml:

Programski kod 14: Sigurnosne dozvole za upotrebu hardware uređaja

```
<uses-feature android:name="android.hardware.camera2"
android:required="true"/>
<uses-permission android:name="android.permission.CAMERA"/>
<usespermissionandroid:name="android.permission.WRITE_EXTERNAL
_STORAGE"/>
```

Nakon toga jednostavnim naredbama pokreće se ili fotoaparat ili otvaranje galerije:

Programski kod 15: Kod za pozivanje fotoaparata ili galerija

```
startActivityForResult(cameraPhoto.takePhotoIntent(),CAMERA_RE
QUEST);
startActivityForResult(cameraPhoto.takePhotoIntent(),GALLERY_R
EQUEST);
```

Zatim se prema putanji spremljene slike, slika pretvara u oblik pogodan za prijenos s naredbama:

Programski kod 16: Enkodiranje

```
Bitmap bitmap =
ImageLoader.init().from(selectedPhoto).requestSize(512,512).ge
tBitmap();

String encodedImage = ImageBase64.encode(bitmap);
```

Android Studio i Java su sigurno najbolji odabir za implementaciju Android aplikacija. Kvalitetu proizvoda koja se dobiva ovim tipom implementacije može se reći da je još uvijek nenadmašiva. Iako je u početnom istraživanju glavna prednost nativnih aplikacija bila performansa, tijekom implementacije nije toliko došla do izražaja. S druge strane, grafičko sučelje odnosno nativan UI je pokazao svoju kvalitetu pri izradi aplikacije. Elementi se mogu vrlo lako modificirati i prilagođavati te su vrlo dobrog izgleda. Postoje razne sitne animacije koje daju nativan osjećaj korisniku. Svojstvo nativnosti grafičkog sučelja se najviše primijetilo kod izbornika gdje klikanje i povlačenje izbornika je vrlo dobro animirano. Nakon

implementacije izbornika u Ionic okviru dobije se dojam dobrog i kvalitetnog izbornika koje Ionic pruža. Nakon nativne verzije ipak se osjeti znatna razlika. Nadalje, vrlo je važno spomenuti i Android Studio čiji alati uvelike pomažu programeru u svim segmentima razvijanje aplikacije, a ne samo u programiranju.

Ovo je bila zadnja obrada i implementacija mobilne aplikacije za vozače. U sljedećem poglavlju napravljena je analiza svih obrađenih okvira gdje će se svi okviri promatrati kao isti tip aplikacije odnosno kao mobilne aplikacije što one ustvari i jesu.

7. Usporedba rezultata provedenih analiza

Nakon isprobavanja i implementiranje aplikacije u svim navedenim okvirima potrebno je usporediti sve okvire, ali ne prema kategorijama, nego prema tipu aplikacije to jest prema hibridnoj, nativnoj ili Web aplikaciji. Za provedbu usporedbe na sve okvire će se gledati kao da su istog tipa, a to je aplikacija za pametni telefon. Usporedit će ih se prema sličnim faktorima koji su postavljeni i za početnu analizu svih kategorija. Odabrani kriteriji su sljedeći: performansa, zajednica, grafičko sučelje i potrošeno vrijeme. Ovog puta će se sve analizirati iz vlastitih iskustva i primjera, odnosno ne toliko teorijski koliko praktično. Na kraju će se izračunati i prosječna ocjena svih okvira. Mnogo je bolja ideja da se analizira sve prema faktorima, a ne prema okvirima kako bi se dobila što bolja usporedba i što bolji utisak na analizu. Također, cilj je da se uopće ne gleda na teorijsko analiziranje, nego da se sve ocjene donesu na temelju vlastitih zapažanja.

7.1. Početak rada s novim okvirom

U ovo poglavlju bit će analizirani i uspoređeni svi okviri prema iskustvu autora od instalacije okvira do kraja implementacije. U analizi će se uspoređivati jednostavnost kreiranja projekta, značajke samog okvira, prednosti i nedostaci okvira, sama implementacija itd.

Ionic - Ionic je opravdao svoju reputaciju, krenuvši od same instalacije okvira, do kreiranja projekta koje verzije se želi s kojim početnim Starterom. Struktura projekta je također za pohvalu, gdje se AngularJS sam inicijalizira sa index.html-om te odvojenom i definiranom strukturom kontrolera i app.js-a. To daje uvid u dobru praksu korištenja samog AngularaJS-a.

Jedini problem na koji se naišlo su verzije okvira gdje su i struktura i tehnologija drugačiji. U verziji 2.0 se koristi Angular 2 koji je znatno različit od AngularJS -a. Također, koristi se i TypeScript te je potrebna adaptacija. Većina tutoriala je namijenjena za verziju 1 (verzija 2 je puno napredovala u zadnjih 2-3 mjeseca) stoga instalacijom najnovije verzije može doći do zbunjenosti.

Android Studio i Java - Dovoljno je reći da nijedna tehnologija nema svoj IDE osim nativnih aplikacija u Javi. Instalacija je vrlo jednostavna kao i kreiranje novog projekta uz razne aktivnosti koje se mogu uključiti i uvelike pomoći pri implementaciji. Struktura projekta je malo kompliciranija, potrebno je više vremena kako bi se shvatilo gdje se što

nalazi i kako se koristi. Potrebno je dosta toga proučiti prije nego što se krene u samu implementaciju bilo čega.

Xamarin - Instalacijom Visual Studio-a instalirano je sve što je potrebno za rad i početak učenja u Xamarin-u. Visual Studio znatno olakšava razvoj od kreiranja projekta pa do svih značajki koje Visual Studio posjeduje. Veoma je praktična instalacija dodataka koji se žele ondje gdje postoji sučelje za pretragu i opis dodatka, značajke koje nudi itd. U Ionic-u postoji obaveza sve to ručno upisivati u terminal te pozvati NPM ili neki drugi paket menadžer za instalaciju nekog dodatka. Aplikacija je implementirana na Mac-u te se moraju spomenuti problemi s Visual Studio-m pogotovo jer vrijeme implementacije nije bilo predugo. Također, nije bio samo jedan problem nego više njih, od neprepoznavanja nekih napisanih linija do pamćenja starih errora itd. U projektu se lako snaći te je struktura iznenađujuće dobra iako postoji odvojena implementacija za svaku platformu.

Materialize CSS - Mora se uzeti u obzir da je ovo samo pristupni-dio(front-end) okvir te da je instalacija podosta jednostavna, odnosno uključivanje 3 skripte u HTML dokument. Također, pohvalno je da se lagano nauči korištenje samog okvira, korištenje grid sustava te ostalih značajki koje Materialize posjeduje. Materialize ima podosta nedostataka kod svih tih korištenja. Kod kreiranja izbornika pojavio se problem gdje kôd iz dokumentacije ne radi. Postoji podosta problema u Materialize-u, odnosno vidi se da dosta toga još nedostaje. Veliki problem predstavljale su verzije gdje neke metode nisu postojale iz izbornika u izabranoj verziji okvira gdje ta metoda ima istu sintaksu korištenja i u jednoj verziji postoji, a u drugoj verziji postoji isto, ali prvo slovo metode je veliko. Nadalje, uključivanje jQuery-jem je izazvalo dodatne probleme tamo gdje nije svaka verzija jQuery-a kompatibilna sa svakom verzijom Materialize-a. Ovakve postavke bi trebale biti osnovne te tu ne bi trebalo biti problema.

Tablica 4: Usporedba početka rada u okviru prema svim okvirima

Ionic	Nativna u Android s.	Xamarin	Materialize
9	10	8	5

7.2. Performansa

Performansa je naravno bitna kod svake aplikacije, ali pogotovo kod mobilne, gdje su specifikacije uređaja mnogo slabije nego kod računala. Ovaj faktor je analiziran osobnim zaključcima koji su stečeni tijekom implementacije te mjerenjem brzine izvršavanje zahtjeva i renderiranje istih. U zahtjevu je bilo potrebno poslati jedan parametar i kao odgovor dobiti 1171 radnih naloga koje treba prikazati na ekranu.

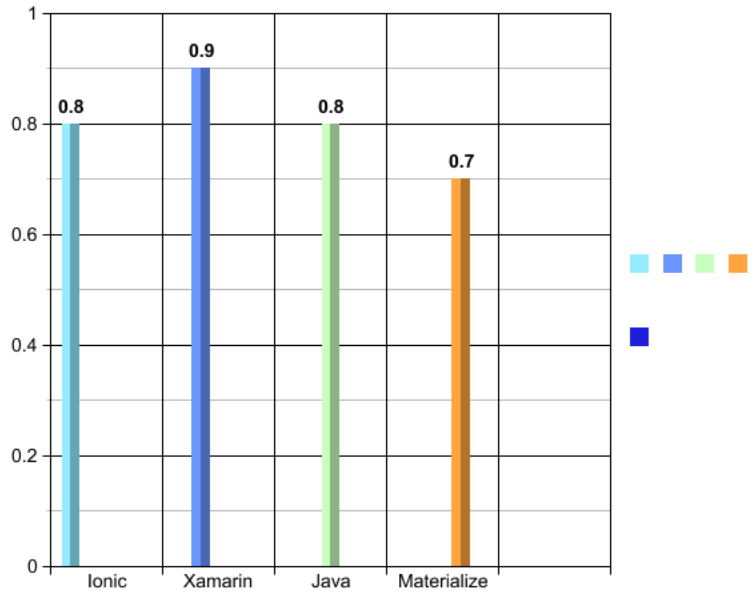
Ionic - Bitno je spomenuti da u aplikaciji nije bilo toliko puno zahtjevnijih komponenata, ali svejedno je bilo podosta UI-elementa. Ne može se reći da se osjeti bitna razlika u usporedbi s nativnom aplikacijom ili aplikacijom u Xamarin-u kada je količina podataka normalna, ali se podosta osjeti kada je podataka više kao što se vidi u primjeru.

Android Studio i Java - Nativna aplikacija je sigurno najbolje rješenje ako je aplikacija koja se treba implementirati velika ili ako obrađuje velike količine podatke. Dovoljno pokazuje primjer iz ovog rada gdje se iscrtavanje svih elemenata izvršilo za 12 sekundi.

Xamarin - Xamarin ima mogućnost da poziva direktno neke nativne biblioteke, stoga ga poneki članci svrstavaju u nativnu verziju, ali on također ima poseban sloj koji se zapravo vrti na mobilnom uređaju, stoga smatram da je hibridna aplikacija. Kao što se može vidjeti performanse su podosta slične nativnoj verziji.

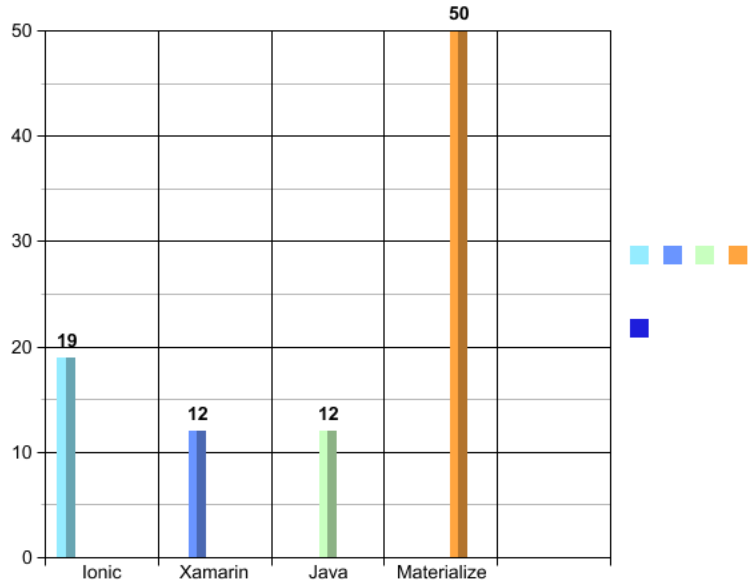
Materialize CSS - Kod web aplikacije mora se primijetiti kako dosta duže traje odlazak s jedne stranice na drugu što za pametne telefone nije baš ugodno. Obrada zahtjeva je bila puno brža nego kod ostalih aplikacija, ali renderiranje je 4 od 5 puta izbacilo pogrešku stranice.

Sljedeći graf (Slika 24.) nam pokazuje koliko je vremenski bilo potrebno svakom okvira da obradi zahtjev s identičnog API-ija. U zahtjevu koji ima preko tisuću elementa ovakva razlika je zanemariva.



Slika 24: Usporedba svih okvira prema vremenu dobivanja odgovora sa servisa

Razlika za obradu zahtjeva je bila mala, ali u vremenu potrebnom za renderiranje više od 1000 elemenata razlika je već povelika. Slika 25. prikazuje usporedbu svih okvira prema vremenu potrebnom za renderiranje svih dohvaćenih podataka. Xamarin je začuđujuće izjednačen s nativnom aplikacijom dok je Ionic dosta sporiji. Dok je Materialize CSS gubi svaki korak u renderiranju težih zahtjeva.



Slika 25: Usporedba svih okvira prema vremenu prikaza elemenata

Tablica 5: Ocjene performanse svih okvira

Ionic	Nativna u Android s.	Xamarin	Materialize
7	10	10	5

7.3. Zajednica i raširenost

Ovaj rad pokazao je da je zajednica pojedinog okvira uz ideju samog okvira među najvažnijim faktorima za odabir okvira, ali da ipak ideja nije dovoljna za uspjeh i njegov razvoj.

Ionic zajednica je iz moje perspektive za svaku pohvalu. Ionic je na verziji 4, a izlaskom svake verzije drastično raste prema gore. Promjena s 1.0 na 2.0 gdje Ionic prelazi s AngularJS-a na Angular 2 i TypeScript samo govori koliko je Ionic ambiciozan i koliko rade na sebi. Stvarno nije teško biti programer uz takvu raširenost i takvu zajednicu okvira.

Android Studio i Java - Pitanja o Androidu u Javi ima najviše na stranici StackOverflow i vlasnik je Google, što je dovoljan pokazatelj o njihovoj zajednici i raširenosti tehnologije. Pri istraživanju za neko pojedino rješenje problema prvi link je najčešće bio rješenje točnog onog problema koji se pojavio pri razvoju.

Xamarin- Nažalost, Xamarin nema tako dobru zajednicu i često su programeri osuđeni na svoje snalaženje. Verzije Xamarin-a ne mogu se mjeriti s primjerice Ionic-om kod kojeg su vidljivi ogromni pomaci s raznim novim značajkama. U dosta slučajeva možda je bolje pretražiti Google-ovu dokumentaciju jer je omotač dosta sličan nativnom SDK-u. Također, bilo je potrebno dobro se potruditi kako bi se pronašao valjan primjer implementacije dodatka te okidanje slike ili odabir galerije.

Materialize CSS - Materialize je okvir koji je pokazao kako ideja nije jedini bitan faktor. Zajednice kod Materialize-a praktički i nema, dok je ideja vrlo dobra uz temeljenje na Material Designu. Kao što je već rečeno, postoji mnogo nelogičnosti i prepreka koje vjerojatno i nisu toliko velike, ali jednostavno nema ispraviti takve pogreške.

Tablica 6: Ocjena zajednice i raširenosti svih analiziranih okvira

Ionic	Nativna u Android s.	Xamarin	Materialize
9	10	6	4

7.4. Grafičko sučelje

U ovom poglavlju je napravljena analiza grafičkog sučelja svih tipa aplikacija. Analiza je napravljena prema iskustvu autora stečenim tijekom implementacije. Neki vrlo važni segmenti promatranja su: jednostavno implementacije UI-a, Izgled, nativnost, proširivost i fleksibilnost.

Ionic - Kao što je već rečeno Ionic ima bogatu zajednicu te se implementira sve više i više novih komponenata koje vrlo dobro izgledaju na mobilnim aplikacijama. Mora se spomenuti da je bilo podosta problema s nekim Ionic elementima koji su teški za modificiranje. Također, potrebno je za svaki novi element pogledati dokumentaciju kako se to radi u Ionicu, odnosno koja je najbolja praksa jer se uvijek može pobjeći od problema s čistim HTML-om, ali ovo je svakako jedna stvar na kojoj bi Ionic još mogao poraditi.

Android Studio i Java - Google Material Design daje poseban osjećaj klikanja po samoj aplikaciji. Također, ono što najviše igra ulogu kod nativnog grafičkog sučelja je upravo osjećaj kao da se klika po nekom fizičkom objektu. Sitne animacije daju jako dobar izgled i odlično korisničko iskustvo.

Xamarin- Xamarin ima nativno grafičko sučelje za obje platforme. Ipak, poneki elementi se moraju sami implementirati te tako način gubi nativnost. Isto tako, teško je modificirati neke elemente, gdje je primjerice za dodavanje slike na određenu poziciju u zaglavlju bilo potrebno raditi implementaciju na obje platforme.

Materialize CSS - Materialize je bio dobar odabir za okvir prednjeg dijela jer se podosta može naučiti o razvijanju okvira. Ideja samog okvira je vrlo dobra te njegov dizajn i animacije imaju veliki potencijal. Svi ti elementi izgledaju odlično na računalu, ali i na mobilnim uređajima. Ima poprilično nativni izgled i interakciju.

Tablica 7: Ocjene grafičkog sučelja svih analiziranih okvira

Ionic	Nativna u Android s.	Xamarin	Materialize
5	10	6	9

7.5. Utrošeno vrijeme

U ovom se dijelu razlikuje nativna aplikacija od svih ostalih jer nije među-platformska aplikacija te je potrebno dvostruko više vremena za implementaciju. Može se vidjeti da i za ostale okvire ne vrijedi uvijek pravilo: „piši jednom, pokreni svugdje“.

Ionic - Aplikacija je pokrenuta i na iOS-u te osim status zaglavlja koje je trebalo instalirati, sve je bilo identično kao na Android verziji. Vrlo je bitno koristiti pravilno CSS jer će poneki elementi biti različiti na iOS-u. Ionic je 100% cross-platform okvir, ali svejedno postoje sitnice koje se trebaju dorađivati.

Android Studio i Java - Nativna aplikacija vremenski je rađena kao i hibridna aplikacija, ali aplikacija u Ionic-u je napravljena cijela, do sitnih detalja dok su ostale aplikacije rađene bez pretjeranog stiliziranja. Naravno, ovo je aplikacija samo za Android platformu što bi značilo da sve isto treba proći i za iOS platformu te je vrlo vjerojatno da će biti dodatnog mijenjanja dizajna za pojedinu verziju. Naravno, ovo je najveća mana nativnih aplikacija.

Xamarin - Pri prvom doticaju sa Xamarin-om na svim službenim dokumentacijama i na neslužbenim člancima piše kako je Xamarin 90% među-platformski okvir, ali to nije tako. Potrebno je uložiti dodatno vrijeme za adaptaciju svake platforme. Pri implementaciji stranice prijave trebalo je postaviti standardno unosno polje s obrubom, a tog elementa nema. Stoga je bilo potrebno raditi CustomerRenderer za svaku platformu te implementirati takvo unosno polje za svaku platformu u njegovom nativnom obliku. Mora se reći kako takvih modifikacija ima mnogo, stoga će se dosta vremena izgubiti na tome.

Materialize CSS - Materialize je vrlo jednostavan za implementaciju, jednostavno korištenje grid sustava te jednostavno uključivanje raznih značajki omogućuje brzo razvijanje aplikacija. Kao što je već napisano, zajednica je podosta loša te postoje neshvatljive pogreške koje je nekad teško riješiti. Važno je spomenuti kako se u Materialize-u ne mogu koristiti sve značajke mobilnog uređaja.

Tablica 8: Ocjene utrošenog vremena svih analiziranih okvira

Ionic	Nativna u Android s.	Xamarin	Materialize
10	5	7	8

Nakon detaljne analize svih okvira po odabranim faktorima može se vidjeti skup svih ocjena prema okviru i prema kriterijima te prosječna ocjenu svakog okvira u tablici 6.

Tablica 9: Ocjene svih kriterija i prosječna ocjena prema svakim analiziranim okvirima

	Ionic	Nativna u Android Studio-u	Xamarin	Materialize
Performansa	7	10	10	5
Zajednica	9	10	6	4
Grafičko sučelje	5	10	6	9
Potrošeno vrijeme	10	5	7	8
Ukupna ocjena	7.75	**8.75**	7.25	6.5

8. Zaključak

Nakon završetka analize i ocjenjivanja svih kriterija svakog okvira izračunata je prosječna ocjena za svaki okvir. Nativne aplikacije su na vrhu s ocjenom 8.75, a zatim slijedi Ionic sa 7.75 što je vrlo velika razlika, a može se reći i neočekivana prema početnim analizama i tezama. Na 3. mjestu je Xamarin s dosljednom ocjenom 7.25 i na kraju okvir za web aplikacije 6.5. Analiza potvrđuje da nativna aplikacija u Android Studio-u ima još uvijek znatnu prednost nad ostalim tipovima aplikacije. U svakom slučaju može se zaključiti da odabir razvijanja aplikacije u Android Studio-u neće biti pogreška. Ionic i Xamarin su opravdali svoju reputaciju te brojne članke s pozitivnim komentarima o tim okvirima te hibridnim okvirima općenito. Materialize CSS pokazuje neke vrlo dobre principe i ideje koje su iznenađujuće kvalitetne, ali upravo zbog tog razloga je još više iznenađujuće nezadovoljavajuća zajednica i podrška koju taj okvir ima. Upravo zbog takvih karakteristika ovaj okvir je izgubio korak za ostalim analiziranim okvirima.

Ovu temu sam izabrao najviše zato što sam se bavio hibridnim aplikacijama i u tom razdoblju sam uvidio kako se s hibridnim aplikacijama može u većini slučajeva napraviti aplikacije koje će biti podosta slične nativnim. Upravo zbog toga mi nije bilo jasno zašto bi netko radio dvostruki posao kad se može s vrlo laganom tehnologijom napraviti solidna, čak i više nego solidnu aplikaciju za obje platforme. Ova teza me potaknula da ovim radom dokažem kako je uistinu tako, međutim **moje mišljenje se uvelike promijenilo**.

Tehnologija se ne bira na takav način u današnjem svijetu, živimo u doba prestiža i novca gdje će velike „ribe“ gledati da plate što više i što veću kvalitetu, a male „ribe“ će gledati da to bude što oskudnije, da što bolje prođu i da aplikacija radi. Stoga dolazim do pitanja, kakva je moja želja –želim li biti mala tvrtka i manipulirati malim količinama novca ili želim biti velika tvrtka i da profit i organizacija cijele firme bude što veća. Naravno, svaka ta odluka i svaki taj put donosi i velike razlike u odgovornosti i riziku, ali to je sve na nama ovisno o tome tko želimo biti. Mislim da su te odluke i pitanja takvog tipa mnogo bitnija nego odabir same tehnologije. Tko želite biti i za koga želite raditi tek tada dolazimo do pitanja s čim želite raditi. Što se tiče tehnologija, uistinu je svaka dokazala svoju vjerodostojnost pri izradi i pokazala da se može napraviti kvalitetna i lako održiva aplikacija. Prednosti i mane postoje i uvijek će postojati, ali na kraju za koju god tehnologiju se odlučimo ne možemo reći da smo pogriješili ako odgovara namjeni našeg poslovnog plana. Posebno bih izdvojio web aplikacije i mobilne aplikacije jer mislim da je budućnost u mobilnim aplikacijama budući da je mobilni uređaj daleko više vezan uz korisnika nego računalo. Dok moj mobitel koristim

samo ja i 100% vremena je uz mene, računalo uvijek koristi više ljudi. Postavlja se pitanje: pa web aplikacija može izgledati isto kao i mobilna te smo uvelike uštedjeli na vremenu, zašto ne bismo imali web aplikaciju i osigurali korištenje na svim platformama i na svim uređajima uz to? Moj odgovor na to pitanje je da je mobilna aplikacija također daleko više personalizirana za korisnika te koristi punu snagu mobilnog uređaja, ali je, po meni, najbitnija stvar zašto odabrati mobilnu, a ne web aplikaciju, uz naravno namjenu aplikacije, korisničko iskustvo. Mislim da svaki korisnik želi imati ikonicu svoje aplikacije koje koristi na ekranu te da se jednostavnim klikom na ikonu otvara aplikacija koju on želi koristiti i gdje se njegov željeni sadržaj nalazi, dok nas web aplikacija „guši“ svojim upisivanjem url-a, dugim učitavanjem i traženjem željenog sadržaja. Spomenuo bih i u raznim člancima mnogo spominjanu performansu kod hibridnih aplikacija kao veliki nedostatak. Mislim da to vrijeme polako prolazi jer su performanse današnjih mobilnih uređaja kao kod računala pa mislim da će to svojstvo uskoro biti zanemareno. Za kraj mogu reći da je bilo vrlo zanimljivo prolaziti kroz sve okvire i njihove implementacije, svi primjeri aplikacija se mogu pronaći na sljedećem repozitoriju (https://github.com/fvukovic/Diplomski-rad). Mogu reći da su me sve implementacije mnogo naučile, gdje ne govorim samo o programiranju i o korištenju 4 popularna okvira, nego i velikoj razlici između svih tipova aplikacija.

Za kraj, želim zahvaliti tvrtki FINA - Financijska agencija, Sektoru informatike, te njihovim zaposlenicima Draženu Pondeljaku i Hrvoju Gluhaku koji su kroz sumentorstvo popratili rad na ovoj temi.

Literatura

1. Alexseyenko, A. (2017). *Angular 2 vs React. What to chose in 2017?* Službena stranica poduzeća Tech Magic. Preuzeto 25. svibnja 2018. s:https://blog.techmagic.co/angular-2-vs-react-what-to-chose-in-2017/
2. Angelini, E. (2012). *5 Pros and Cons of Appcelerator's Titanium.* Službena stranica poduzeća Spirales. Preuzeto 6.lipnja 2018 s: https://enricoangelini.com/2012/5-pros-and-cons-of-appcelerators-titanium/
3. Angular.org (2016). *AngularJs documentation*. Službena stranica AngularJS-a. Preuzeto 25. svibnja 2018. s: https://angularjs.org/
4. Apache Cordova (2018). Službena stranica Apache Cordove. Preuzeto 1. kolovoza 2018. s: https://cordova.apache.org/docs/en/latest/
5. AppDynamics (2018). *Službena stranica AppDynamicsa*. Preuzeto 8. rujna 2018. s:https://www.appdynamics.com/
6. Arsenault, C. (2018). Top 10 Front-End Frameworks of 2018. Preuzeto 10. lipnja 2018. s: https://www.keycdn.com/blog/front-end-frameworks/
7. Bristowe, J. (2015). *What is hybrid Mobile App.* Preuzeto 4. lipnja 2018. s: https://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/
8. Burnetter, E. (2011). *How Android works: The big picture.* Preuzeto 22. kolovoza 2018. s:https://www.zdnet.com/article/how-android-works-the-big-picture/
9. Chauhan, S. (2017). *Understanding Xamarin Forms.* Preuzeto 6.lipnja 2018. s:https://www.dotnettricks.com/learn/xamarin/xamarin-forms-build-cross-platform-mobile-apps
10. Daniel, S. F. (2017). *Mastering Xamarin UI Development*. Packt Publishing, Birmingham
11. Dumić, D. (2017). *Hibridne i nativne mobilne aplikacije.* Preuzeto 8. lipnja 2018.s:http://blog.king-ict.hr/dragutin-dumic/hibridne-i-nativne-mobilne-aplikacije
12. Firtman, M. (2010). *Programming the Mobile Web.* O'Reilly Media, Sebastopol
13. Gajotres.net (2015). *The Top 5 AngularJSHybrid Mobile App Frameworks (Pros/Cons.)* Preuzeto 6.lipnja 2018. s: https://www.gajotres.net/the-top-5-angularjs-hybrid-mobile-app-frameworks-proscons/
14. Google.com (2018). *Official Android Documentation.* Preuzeto 20.kolovoza 2018. s: https://developer.android.google.cn/

15. Gok, N., Kanna, N. (2013). *Building Hybrid Android Appswith Java andJavaScript.* O'Reilly Media, Sebastopol
16. Hanselman, S. (2014). *Xamarin.Forms – Write Once, Run Everywhere, AND Be Native?*. Preuzeto 1. lipnja 2018. s:https://www.hanselman.com/blog/XamarinFormsWriteOnceRunEverywhereANDBeNative.aspx
17. Ionic.com (2018). *Ionic Framework Documentation*. Službena stranica Ionic-a. Preuzeto 25. svibnja 2018. s:http://ionicframework.com/docs/overview/
18. Jscrambler.com (2017). *12 Frameworks for Mobile Hybrid Apps.* Preuzeto 6.lipnja 2018. s:https://blog.jscrambler.com/10-frameworks-for-mobile-hybrid-apps/
19. Kazman, R. (2012). *Software Architecture in Practice-3rd Edition.* Addison-Wesley Professional
20. Materializecss.com (2018). *Materialize Documentation*. Službena stranica. Preuzeto 22. kolovoza s: https://materializecss.com/
21. May, J. (2010). *Microsoft Visual Studio 2010: A Beginner'sGuide*. McGraw-Hill Education, New York
22. Mew, K. (2015). *Learning Material Design.* Packt Publishing, Birmingham
23. Microsoft.com (2018). *Xamarin documentation.* Službena stranica.Preuzeto 4. lipnja 2018.s:https://docs.microsoft.com/en-us/xamarin/
24. Reynolds, M. (2014). *Xamarin Essentials.* Packt Publishing, Birmingham
25. Schildt, H. (2014). *Java: A Beginner's Guide Sixth Edition.* McGraw-Hill Education, New York
26. Schmidt, D. (1997). *Applying Patterns and Frameworks to Develop Object-Oriented Communication Software.* Department of Computer Science
27. Singla, D. (2018). *Bootstrap vs Foundation: Who Has An Upper Hand?* TemplateToaster.com Preuzeto 22. kolovoza s: https://blog.templatetoaster.com/bootstrap-vs-foundation/
28. Zuniga, R. (2016). *Using C and C++ Code in an Android App with the NDK*. Preuzeto 10. lipnja s: https://www.sitepoint.com/using-c-and-c-code-in-an-android-app-with-the-ndk
29. Williamson, K. (2015). *LearningAngularJS: A Guide to AngularJS Development.* O'Reilly Media, Sebastopol

Popis ilustracija

Popis slika:

Popis tablica:

Popis programskih kôdova: