Hibridna mobilna aplikacija za vremensku prognozu

Cerovec, Anja

Sveučilište Sjever

Završni rad br. 703/MM/2020

Hibridna mobilna aplikacija za vremensku prognozu

Anja Cerovec, 2224/336

Varaždin, rujan 2020. godine

Sveučilište
Sjever

Odjel za Multimediju, oblikovanje i primjenu

Završni rad br. 703/MM/2020

Hibridna mobilna aplikacija za vremensku prognozu

Student

Anja Cerovec, 2224/336

Mentor

Vladimir Stanisavljević, mr.sc.

Varaždin, rujan 2020. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL Odjel za multimediju

STUDIJ preddiplomski stručni studij Multimedija, oblikovanje i primjena

PRISTUPNIK Anja Cerovec | MATIČNI BROJ 2224/336

DATUM 14.09.2020. | KOLEGIJ Programski alati 3

NASLOV RADA Hibridna mobilna aplikacija za vremensku prognozu

NASLOV RADA NA ENGL. JEZIKU Hybrid mobile application for weather forecast

MENTOR mr.sc. Vladimir Stanisavljević | ZVANJE Viši predavač

ČLANOVI POVJERENSTVA

1. izv.prof.dr.sc. Dean Valdec - predsjednik
2. doc.dr.sc. Andrija Bernik, pred. - član
3. mr.sc. Vladimir Stanisavljević, v.pred. - mentor
4. pred. Snježana Ivančić Valenko, v.pred. - rezervni član
5.

Zadatak završnog rada

BROJ 703/MM/2020

OPIS

Kod hibridnih mobilne aplikacije koriste se HTML(5) tehnologije kao podloga za programiranje. Za iskorištenje svih ugrađenih mogućnosti pojedine platforme poput lokacijskih usluga ili kamere potrebno je mobilnu aplikaciju pisati u nekom specifičnom alatu koji iz HTML-a omogućuje pristup tim mogućnostima. Podaci se u aplikaciju mogu učitati iz vanjskih izvora na internetu tipično AJAX tehnologijama. U ovom radu potrebno je izraditi višeplatformsku aplikaciju za vremensku prognozu koristeći standardne web tehnologije i jedan takav alat.
U radu je potrebno:
• opisati osnovne Cordova programske zbirke i mogućnosti koje pružaju za razvoj aplikacija i usporediti Apache Cordova s konkurencijom,
• istražiti i opisati sučelje za pristup udaljenim izvorima podataka vremenske prognoze preko dokumentiranog sučelja,
• pomoću Cordove izraditi složeniju aplikaciju koja koristi ugrađene mogućnosti mobilnih platformi i prikazuje podatke pribavljene iz udaljenih izvora podataka vremenske prognoze
Detaljno opisati sve korištene tehnologije i korake u radu potrebne da bi se ostvarilo traženo te detaljno opisati stečena iskustva i postignute rezultate.

ZADATAK URUČEN 21.09.2020. | POTPIS MENTORA

SVEUČILIŠTE SJEVER

Predgovor

Tema ovog završnog rada odabrana je zbog osobne zainteresiranosti za programiranje i web dizajn. Istraživanjem moguće teme rada, naišla sam na pojam hibridne aplikacije pa tako i do Apache Cordove, koju sam na kraju i odabrala za izradu aplikacije. Cilj ovog rada je steći nova znanja u izradi hibridne mobilne aplikacije te na taj način približiti isti proces svima zainteresiranima.

Zahvaljujem se profesoru i mentoru Vladimiru Stanisavljeviću na savjetima i strpljenju tijekom izrade završnog rada. Također zahvaljujem obitelji na podršci tijekom cijelog školovanja.

Sažetak

Posljednjih godina mobilne aplikacije su uvelike promijenile načine na koje komuniciramo, slušamo glazbu, gledamo filmove, pratimo novosti pa i vremensku prognozu i tako postale sastavni dio našeg života do te mjere da se osjećamo nepotpuno ako zaboravimo mobitel kod kuće. Mobilne aplikacije dijele se na nativne, web i hibridne. Nativne aplikacije razvijene su korištenjem izvornog programskog jezika platforme za koju su namijenjene, a web aplikacije najčešće se razvijaju uz pomoć HTML-a, CSS-a i JavaScript-a dok su hibridne aplikacije kombinacija nativnih i web aplikacija. Hibridne aplikacije kodirane su pomoću standardnih web tehnologija, a uz pomoć dodataka i posebnih aplikacijskih spremnika pružaju osjećaj nativne aplikacije.

Pri izradi hibridnih aplikacija uvelike pomažu mnogi besplatni i komercijalni razvojni okviri koji se mogu naći na tržištu kao npr. Apache Cordova. Cordova je besplatan razvojni okvir u vlasništvu Apache zaklade kojim možemo razviti višeplatformsku hibridnu aplikaciju korištenjem web tehnologija koje se inače koriste za izradu web aplikacije. Cordova nudi set dodataka tj. API-ja koji omogućuju pristup nativnim značajkama uređaja kao što su kamera, lokacija, datotečni sustav, obavijesti i status baterije. Apache Cordova aplikaciju možemo razviti uz pomoć Cordova CLI-a ili koristeći se nekim od IDE-a koji olakšavaju cijeli proces.

U teorijskom dijelu rada opširnije je obrađena Apache Cordova i opisane su osnove njoj sličnih razvojnih okvira. Opisana je povijest Cordove, arhitektura, dodaci i proces instalacije Cordova CLI-a.

Praktični dio rada sadrži opis izrade aplikacije za prikaz trenutnih vremenskih uvjeta uz pomoć Apache Cordove i Visual Studija. Opisano je koje komponente je potrebno instalirati te su detaljno opisani dijelovi koda koji omogućuju funkcioniranje aplikacije.

Ključne riječi: Apache Cordova, hibridna aplikacija, web aplikacija, nativna aplikacija, API dodaci, HTML, CSS, JavaScript

Summary

In recent years, mobile applications have greatly changed the way we communicate, listen to music, watch movies, follow the news and event the weather forecast and thus became an integral part of our lives to the point that we feel incomplete if we forget our mobile phone at home. Mobile applications are divided into native, web and hybrid applications. Native applications are developed using native programming language of the platform for which they are intended, web applications are most often developed using HTML, CSS and JavaScript and hybrid applications are a combination of native and web applications. Hybrid applications are coded using standard web technologies and by using plugins and special application containers they provide a sense of native application.

There are many free and commercial frameworks which can be found on the market, such as Apache Cordova, that are helpful when building a hybrid application. Cordova is a free mobile development framework, owned by Apache, by which we can develop a multi-platform hybrid application using web technologies that are otherwise used to create a web application. Cordova offers a set of plugins, i.e. APIs that provide access to the native features of the device such as camera, location, file system, notifications and battery status. We can develop Apache Cordova application using Cordova CLI or one of many IDEs that make the whole process easier.

The theoretical part of the paper describes Apache Cordova and similar development frameworks. The history of Cordova, its architecture, plugins and the installation process of Cordova CLI are also described.

The practical part of the paper contains a description of weather application development by using Apache Cordova and Visual Studio. It is described which components need to be installed and the code that allows the application to function is described in detail.

Keywords: Apache Cordova, hybrid application, web application, native application, API Plugin, HTML, CSS, JavaScript

Popis korištenih kratica

WAP Wireless Application Protocol
- tehnički standard za pristup informacijama putem mobilne bežične mreže

API Application Programming Interface
- skup programskih pravila kojih se programeri moraju držati da bi ostvarili željene rezultate kod programa

HTML HyperText Markup Language
- opisni jezik koji služi za izradu web stranica

CSS Cascading Style Sheets
- stilski jezik koji opisuje izgled HTML dokumenta

ASF Apache Software Foundation
- Američka neprofitna organizacija koja podržava Apache softverske projekte, uključujući Apache HTTP Server

CLI Command Line Interface
- Sučelje naredbenog retka; oblik sučelja između operativnog sustava i korisnika u koji korisnik upisuje naredbe u obliku redaka teksta

SDK Software Development Kit
-Paket za razvoj softvera; skup alata za razvoj softvera u jednom instalacijskom paketu

CMD Command Prompt
- Naredbeni redak; cmd.exe je zadani prevoditelj naredbenog retka, komunicira s korisnikom preko CLI-a

UX User Experience
- Korisničko iskustvo; osjećaji i stavovi osobe o korištenju određenog proizvoda

UI User Interface
- Korisničko sučelje; mjesto interakcije korisnika i sustava

MVC Model View Controller
- Obrazac softverske arhitekture koji se koristi u softverskom inženjeringu za odvajanje pojedinih dijelova aplikacije u komponente ovisno o njihovoj namjeni

IDE Integrated Development Environment
- Integrirano razvojno okruženje; softverska aplikacija koja računalnim programerima pruža potrebne funkcionalnosti za razvoj softvera

Sadržaj

1. Uvod

Prvi komercijalno dostupan mobilni telefon bio je DynaTAC tvrtke Motorola proizveden 1983. godine. Imao je samo mogućnost poziva, a u operacijski sustav bila je uključena aplikacija za spremanje kontakata. Tek devedesetih godina mobiteli dobivaju mogućnost slanja SMS poruka, a zatim i alate poput kalendara, radija te alarma. [1]

Početkom 21. stoljeća dolazi do sve ubrzanijeg razvoja mobilnih telefona pa tako dolaze sa sve više mogućnosti i aplikacija. Prije nego što je bilo omogućeno povezivanje na Internet putem mobilnih mreža, morao se koristiti WAP (engl. *Wireless Application Protocol*). Uz pomoć WAP-a korisnici su mogli preuzeti bilo koje aplikacije koje im je ponudio proizvođač telefona. Međutim, potražnja za takve aplikacije bila je prevelika pa WAP nije mogao podnijeti opterećenje. Korisnici su morali čekati svoj red da uspiju preuzeti aplikaciju.

S najavom prvom iPhone-a 2007. godine započela je velika evolucija mobilnih aplikacija. Način na koji korisnici koriste mobilne aplikacije u potpunosti se promijenio. Godine 2008. Apple je na App Store-u objavio oko 550 aplikacija, od kojih je 135 bilo besplatno za preuzimanje. U samo tjedan dana korisnici iPhone-a preuzeli su oko deset milijuna aplikacija. [2]

Svakim danom mobilne aplikacije dobivaju nove mogućnosti, a mijenja se i način na koji ih koristimo. Nekada su aplikacije mogli razvijati samo zaposlenici određenih tvrtki, a danas to svatko može naučiti.

Postoje tri glavne vrste mobilnih aplikacija: nativne, web i hibridne aplikacije [3], a na nama je da odaberemo kakvu aplikaciju želimo izraditi.

Nativne aplikacije razvijene su za određeni operacijski sustav poput Androida ili iOS-a. Aplikacije izrađene za jednu platformu ne mogu se koristiti na bilo kojoj drugoj, npr. aplikacija izrađena specifično za iOS neće raditi na Androidu i obrnuto.

Glavne prednosti nativnih aplikacija su dobro korisničko iskustvo i visoke performanse. Osim toga, pristup širokom rasponu API-ja (engl. *Application Programming Interface*) ne postavlja nikakva ograničenja za upotrebu aplikacija. Nativne aplikacije dostupne su u trgovinama za aplikacije poput Google Play i App Store preko kojih ih možemo preuzeti na svoj mobilni telefon. Neki od nedostataka nativnih aplikacija su veći troškovi održavanja i odvojene korisničke podrške za svaku aplikaciju.

Web aplikacijama se pristupa preko internetskih preglednika i to su zapravo responzivne web stranice koje se prilagode zaslonu uređaja preko kojeg se pristupa. Obično su napisane u HTML-u, CSS-u i JavaScript-u.

Najveća prednost web aplikacija je ta što zahtijevaju najmanje memorije uređaja. Korisnici mogu pristupiti web aplikaciji s bilo kojeg uređaja koji je spojen na Internet, a troškovi održavanja su niski. Nedostatak je taj što korištenje web aplikacija sa lošom internetskom vezom obično rezultira sa vrlo lošim korisničkim iskustvom. Još jedan nedostatak je ograničeni pristup API-jevima.

Hibridne aplikacije izrađene su pomoću različitih višeplatformskih web tehnologija poput JavaScript-a i HTML5. Koriste elemente nativnih i web aplikacija.

Glavna prednost hibridnih aplikacija je ta što su relativno jednostavne za izradu, a uz to se mogu lako i jeftino održavati. Ne trebamo razviti aplikaciju za svaku platformu zasebno, nego se uz pomoć programske zbirke poput Apache Cordove ili Ionica aplikacija sama prilagodi uređaju i platformi na kojoj se koristi. Nedostaci hibridnih aplikacija su brzina, slabije performanse te sveukupna optimizacija u usporedbi s nativnim aplikacijama. Postoje i određeni problemi s prilagodbom dizajna aplikacije za različite platforme.

Kao temu ovog rada odabrala sam izradu hibridne aplikacije za vremensku prognozu, a pritom ću koristiti programski okvir Apache Cordovu. Cordova je mobilni razvojni okvir otvorenog koda koji omogućuje objedinjavanje CSS, HTML i JavaScript koda, ovisno o platformi uređaja. [4]

U prvom dijelu završnog rada teorijski se obrađuje Cordova i slični programski jezici. Opisana je povijest nastanka Apache Cordove, funkcionalnost, nabrojane komponente Cordove, proces instalacije Cordova CLI-a i izrada zadane aplikacije preko alata naredbenog retka te usporedba sa ostalim programskim okvirima za razvoj hibridnih aplikacija.

Praktični dio rada sadrži opis procesa izrade hibridne aplikacije za vremensku prognozu uz pomoć razvojnog okvira Apache Cordove, a za čiju realizaciju su korišteni alati za Apache Cordovu Visual Studija te OpenWeatherMap API sa kojeg su povučeni podaci o vremenskim uvjetima određenog mjesta.

2. Apache Cordova

Apache Cordova je besplatan mobilni razvojni okvir otvorenog koda koji omogućuje izradu višeplatformskih hibridnih aplikacija korištenjem web tehnologija kao što su HTML5, CSS3 i JavaScript. Kreiranje Cordove započelo je idejom o pronalasku jednostavnijeg načina izrade mobilnih aplikacija za više platformi te su kreatori odlučili da je najbolji pristup korištenje kombinacije nativnih i web tehnologija čime nastaje hibridna aplikacija. Aplikacije se izvršavaju na način da su usmjerene na svaku platformu zasebno i oslanjaju se na API-jeve za pristup mogućnostima svakog uređaja kao što su lokacija, podaci i mrežni status. [4]

Cordova funkcionira na način da programer implementira web sučelje koristeći iste web tehnologije koje se koriste za web razvoj, kao i biblioteke i razvojne okvire koji su potrebni za razvoj željene aplikacije. Cordova aplikacije se pokreću unutar klijentskog nativnog aplikacijskog spremnika koji pruža resurse potrebne operacijskom sustavu uređaja domaćina da učita aplikaciju. Ti resursi uključuju osnovne postavke aplikacije poput naziva i podržanih načina rada, zaslona učitavanja te ikone početnog zaslona. Svi resursi i postavke aplikacije zajedno sa kodom aplikacije pakiraju se u jedan aplikacijski paket koji se zatim može postaviti u trgovinu aplikacija.

Kako bismo izgradili Cordova aplikaciju, najprije kreiramo web aplikaciju koristeći standardne web tehnologije. Cordova web aplikacije su zapravo obične HTML5 aplikacije koje se pokreću preko *index.html* stranice, a jedina razlika je što se pri pokretanju također učita generirana *cordova.js* skriptna datoteka koja pruža sučelje za interne značajke Cordove od kojih je najvažniji Cordova upravitelj dodataka (engl. *Cordova plug-in manager*). Cordova.js datoteka je dinamički kreirana preko Cordove za svaku platformu prilikom izgradnje projekta. Kada se aplikacija pokrene, učita početnu stranicu pakirane web aplikacije u *Web View-u* (komponenta za prikaz web sadržaja unutar prozora aplikacije), a zatim se prebacuje kontrola na sam Web View kako bi korisniku omogućio interakciju sa web aplikacijom. Dok korisnik komunicira sa sadržajem aplikacije, linkovi ili JavaScript kod unutar aplikacije mogu učitati drugi sadržaj iz resursnih datoteka pakiranih s aplikacijom ili mogu doprijeti do mreže i povući sadržaj s weba ili poslužitelja aplikacije.

Cordova projekt pruža tri glavna obilježja: [5] alat naredbenog retka, pristup značajkama hardvera te mogućnost podrške budućih značajki. Alat naredbenog retka (engl. *Command Line Tool*) koristi se za stvaranje projekta i sastavljanje koda za mobilne platforme. Uzima naš izvorni HTML, CSS i JavaScript kod i pretvara ga u binarni format potreban za platforme koje podržava Cordova. Alat naredbenog retka također možemo koristiti za slanje koda simulatoru ili stvarnom uređaju.

2.1. Povijest

Cordova razvojni okvir stvorila je kanadska tvrtka Nitobi 2008. godine pod nazivom PhoneGap. Ubrzo je stekao veliku popularnost među programerima koji su željeli razvijati mobilne aplikacije, ali nisu znali kako programirati koristeći izvorni jezik uređaja. PhoneGap je omogućio korištenje standardnih tehnologija u razvoju web stranica za izradu mobilnih aplikacija. U početku, PhoneGap je pružao jednostavne JavaScript API-je za rad sa nekoliko izvornih značajki uređaja, a kasnije je obuhvaćena većina značajki na koje su se programeri mobilnih aplikacija navikli, uključujući kameru, datotečni sustav i obavijesti. S vremenom, projektu je počeo doprinositi IBM kao i mnoge druge tvrtke.

U listopadu 2011. godine Adobe je najavio kupnju tvrtke Nitobi. Istodobno, izvorni kod PhoneGap-a doniran je Apache zakladi (ASF – engl. *Apache Software Foundation*), gdje je najprije preimenovan u Apache Callback, zatim nakratko u Apache DeviceReady, a sada se zove Apache Cordova prema ulici u kojoj su bili smješteni Nitobi uredi kada je kreiran PhoneGap. Odlučeno je da će Apache Cordova uvijek ostati besplatna i otvorenog koda. [5] [6]

2.2. Adobe PhoneGap

Najjednostavnije, PhoneGap (https://phonegap.com/) je verzija Cordove sa nekim dodatnim mogućnostima. Za razvoj PhoneGap aplikacija također nije potrebno znanje nativnih programskih jezika, već je dovoljno poznavati web programske jezike HTML, CSS i JavaScript. Za izgradnju PhoneGap aplikacije koristi se usluga PhoneGap Build (https://build.phonegap.com/) koja omogućava izgradnju aplikacije u oblaku, a to znači da nema potrebe za instalacijom gomile dodatnih stavki te redovnim održavanjem nativnih SDK (engl. *Software Development Kit*) jer će se aplikacija automatski izgraditi na najnoviju verziju SDK ciljane platforme. PhoneGap Build također omogućava dodavanje članova tima i određivanje uloga unutar PhoneGap projekta te brže uklanjanje grešaka i implementaciju. [6]

U kolovozu 2020. Adobe je objavio da se 1. listopada 2020. ukida usluga PhoneGap Build, a time prestaje i daljnji razvoj PhoneGap-a te prestanak Adobe-ovog investiranja u Apache Cordovu. [7]

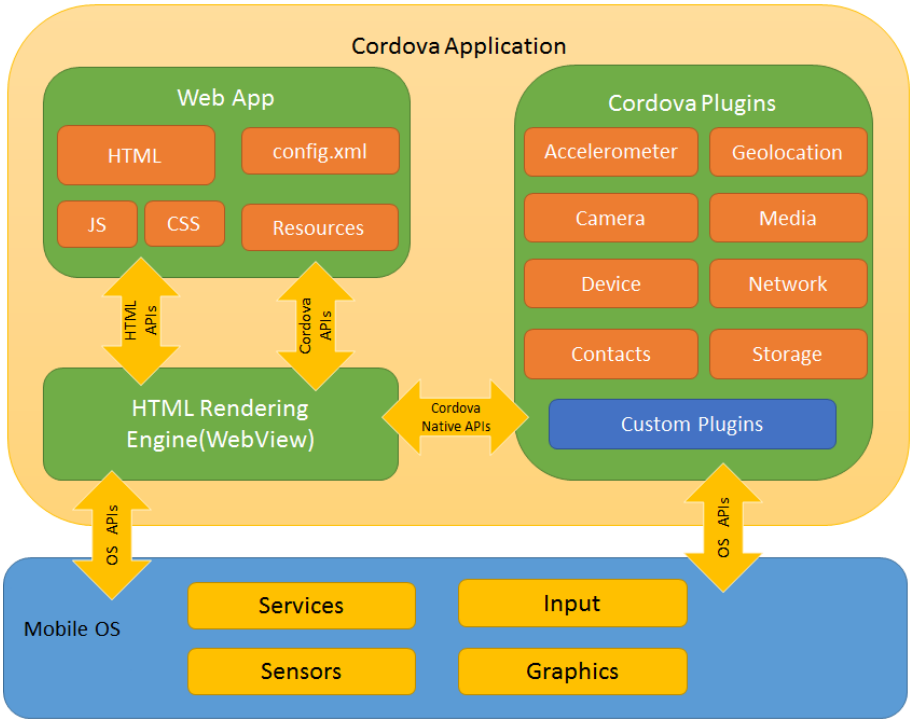
2.3. Razlike PhoneGap-a i Apache Cordove

Iako su PhoneGap i Cordova nastali iz istog projekta, postoje neke manje razlike između njih. Dok je Apache Cordova projekt otvorenog koda kojim se upravlja u sklopu ASF-a, PhoneGap je Adobe-ova implementacija Cordova projekta. PhoneGap je također otvorenog koda, ali Adobe mu je dodao usluge koje nisu besplatne.

PhoneGap i Cordova se također razlikuju u alatu naredbenog retka. Oba alata su dovoljno slična gdje lako možemo koristiti jedan ili drugi, ali osobitosti svakog alata naredbenog retka su različite. Najveća razlika je u tome što se alat naredbenog retka za PhoneGap integrira s Adobeovom uslugom PhoneGap Build.

2.4. Arhitektura Cordova aplikacije

Cordova aplikaciju čini nekoliko glavnih komponenti, a to su: web pogled, web aplikacija te programski dodaci. Slika 2.1 prikazuje detaljniji prikaz komponenti.



Slika 2.1 Arhitektura Cordova Aplikacije,
Izvor: https://cordova.apache.org/docs/en/latest/guide/overview/

Web pogled (engl. *Web View*) – izvorna komponenta aplikacije koja se koristi za prikazivanje web sadržaja (obično HTML stranice) unutar prozora ili zaslona nativne aplikacije. [6] Na nekim platformama također može biti komponenta unutar veće, hibridne aplikacije koja miješa web pogled s izvornim komponentama aplikacije.

Web aplikacija (engl. *Web App*) – Ovo je mjesto u kojem je smješten kod naše aplikacije. Sama aplikacija implementirana je kao web stranica, prema zadanom postavljena kao lokalna datoteka po imenu index.html, koja navodi CSS, JavaScript, slike, medijske datoteke ili druge resurse potrebne za pokretanje. Ovaj spremnik ima vrlo bitnu datoteku nazvanu config.xml koja pruža informacije o aplikaciji i specificira parametre koji utječu na njen rad, kao što je odgovor aplikacije na promjenu orijentacije uređaja.

2.5. Programski dodaci (engl. *Plugins*)

Dodaci su sastavni dio Cordova sustava. Pružaju mehanizam koji omogućuje JavaScript-u pristup značajkama kojima inače ne bi mogao pristupiti te omogućuju komunikaciju koda s uređajem. Ovisno o potrebi, u naš Cordova projekt možemo dodavati željene dodatke.

Apache Cordova projekt sadrži set dodataka nazvanih „Core Plugins“ koji se u raznoj literaturi mogu pronaći i pod nazivom „Core API“. To su osnovni dodaci koji omogućuju aplikaciji pristup mogućnostima uređaja kao što su baterija, kamera, kontakti, lokacija itd.

U nastavku su nabrojani osnovni dodaci te njihove uloge: [4]

- **Battery Status** (praćenje statusa baterije uređaja)
- **Camera** (pristup kameri uređaja)
- **Device** (sakupljanje informacija uređaja)
- **Dialogs** (pristup vizualnim obavijestima uređaja)
- **File** (*read/write* pristup datotekama uređaja)
- **Geolocation** (pristup lokaciji uređaja)
- **InAppBrowser** (pokretanje URL-a u drugoj instanci preglednika unutar aplikacije)
- **Media** (omogućuje snimanje i reproduciranje audio datoteka na uređaju)
- **Media Capture** (pristup mogućnostima fotografiranja te snimanja videa i zvuka)
- **Network Information** (provjera stanja mreže i informacije o mobilnoj mreži)
- **Screen Orientation** (postavljanje/zaključavanje orijentacije uređaja)

- **Splashscreen** (prikazivanje/sakrivanje zaslona učitavanja aplikacije)
- **Statusbar** (prikazivanje/sakrivanje/konfiguriranje statusne trake)
- **Vibration** (omogućuje vibraciju uređaja)
- **Whitelist** (odobrava mrežne zahtjeve aplikacije)

Kada programer implementira značajku u aplikaciju koja koristi jedan od Cordova API-ja, aplikacija poziva API pomoću JavaScript-a, a zatim poseban sloj unutar aplikacije prevodi Cordova API poziv u odgovarajući izvorni API za određenu značajku.

Programski dodatak je zapravo zbirka datoteka koja se sastoji od konfiguracijske datoteke nazvane *plugin.xml* i jedne ili više JavaScript datoteka, a ponekad sadrži datoteku izvornog koda, biblioteke te ostale sadržajne datoteke koje su potrebne dodatku.

Plugin.xml opisuje dodatak te poručuje sučelju naredbenog retka (CLI - engl. *Command Line Interface*) gdje kopirati komponente u projektu ciljane mobilne aplikacije. Unutar plugin.xml datoteke postoje postavke koje CLI koristi za namještanje postavki unutar config.xml datoteke specifičnih za platformu. [6]

Osim službenih dodataka Apache Cordove, možemo koristiti i dodatke koje su stvorili drugi programeri, a koji su dostupni na GitHub-u ili izraditi vlastiti programski dodatak. Dodatak mora imati barem jednu JavaScript datoteku koja definira metode, objekte i svojstva koja se koriste, a broj ostalih datoteka i biblioteka ovisi o tome što želimo da naš dodatak omogućuje.

2.6. Razvojni putevi Cordova aplikacije

Cordova pruža dva osnovna tijeka rada za izradu mobilne aplikacije [4]: *Cross-Platform* tijek rada i tijek rada usmjeren na jednu platformu. U većini slučajeva možemo koristiti bilo koji tijek rada za postizanje istog zadatka, a svaki ima svoje prednosti.

2.6.1. Cross-Platform tijek rada

Cross-Platform tijek rada koristi se kada želimo da se naša aplikacija pokreće na što više različitih platformi i operacijskih sustava bez potrebe za razvijanjem aplikacije za svaku platformu zasebno. Ovaj tijek rada vrti se oko Cordova CLI-a. CLI je alat visoke razine koji omogućuje izradu projekta za više platformi odjednom. Kopira zajednički skup web sredstava u poddirektorije za svaku mobilnu platformu, za svaku izvršava potrebne promjene konfiguracije te pokreće skripte za izradu kako bi generirao binarne aplikacije. CLI također pruža zajedničko sučelje za primjenu dodataka na našu aplikaciju.

2.6.2. Tijek rada usmjeren na jednu platformu

Tijek rada usmjeren na jednu platformu koristi se kada se želimo usredotočiti na izgradnju aplikacije za jednu platformu i trebamo ju moći modificirati na nižoj razini. Na primjer, moramo koristiti ovaj pristup ako želimo da naša aplikacija miješa prilagođene izvorne komponente s Cordova komponentama temeljenim na webu. U pravilu, ovaj tijek rada koristi se ako trebamo izmijeniti projekt unutar paketa za razvoj softvera (SDK).

2.7. Instalacija Cordova CLI-a

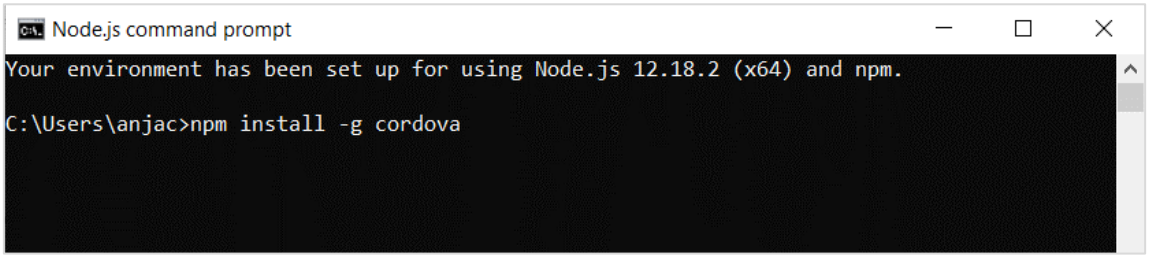
Kako bismo uopće mogli koristiti Apache Cordovu za izradu hibridnih aplikacija, moramo imati instaliran Cordova CLI (sučelje naredbenog retka) koji se distribuira kao *npm* paket (engl. *Node Package Manager*). Npm je upravitelj paketa za programski jezik JavaScript. Sastoji se od klijenta naredbenog retka koji komunicira s udaljenim registrom, a korisnicima omogućuje korištenje i distribuciju JavaScript modula koji su dostupni u registru. [8] Npm također pruža jednostavan način instaliranja softvera zajedno sa svim njegovim ovisnostima.

Na samom početku, moramo preuzeti i instalirati Node.js (https://nodejs.org/en/download/) kojim ćemo omogućiti pozivanje npm-a u naredbenom retku (CMD – engl. *Command Prompt*). Node.js je JavaScript *runtime* okruženje otvorenog koda koje izvršava JavaScript kod izvan web preglednika. Omogućuje stvaranje web poslužitelja i alata za umrežavanje pomoću JavaScript-a te zbirke modula koji upravljaju raznim osnovnim funkcionalnostima. [9]

Preporuča se i instaliranje Git klijenta koji će omogućiti pozivanje *git* naredbe u naredbenom retku. Git je distribuirani sustav za kontrolu verzija, tj. on prati promjene u izvornom kodu tijekom razvoja softvera.

Sljedeći korak je instalacija Cordova modula koristeći npm. Otvorimo Node.js command prompt te upišemo sljedeću naredbu (Slika 2.2):

```
npm install -g cordova
```



Slika 2.2 Node.js command prompt - Instalacija Cordove

Oznaka -g govori npm-u da instalira Cordovu globalno. U suprotnom, instalirat će se u poddirektoriju *node_modules* trenutne radne mape.

Daljnji koraci uključuju izradu direktorija buduće aplikacije, dodavanje platformi i dodataka te instalaciju zahtjeva za svaku dodanu platformu.

2.8. Osnovne naredbe Cordove

U nastavku su nabrojane naredbe koje se najčešće koriste za izradu projekta pomoću Cordova CLI-a. Naredbe se upisuju u Node.js command prompt.

Stvaranje novog projekta:

```
cordova create hello com.example.hello HelloWorld
```

Dodavanje i uklanjanje platformi:

```
cordova platform add android
cordova platform remove android
```

Provjera zahtjeva platformi:

```
cordova requirements
```

Dodavanje i uklanjanje dodataka:

```
cordova plugin add cordova-plugin-camera
cordova plugin remove cordova-plugin-camera
```

Izgradnja projekta za sve platforme:

```
cordova build
```

Testiranje aplikacije preko uređaja i simulatora:

```
cordova run android
cordova emulate android
```

Pregled liste platformi:

```
cordova platform ls
```

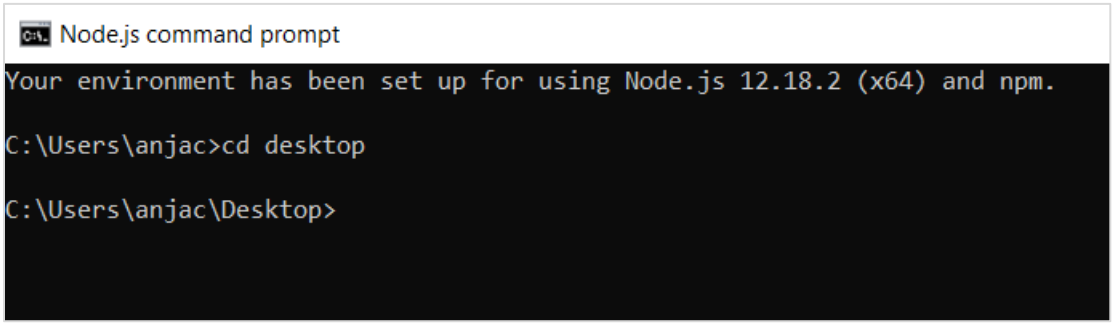
Pregled dodataka u projektu:

```
cordova plugin ls
```

2.9. Stvaranje osnovne Cordova aplikacije

Nakon što smo instalirali Cordova CLI na način kao što je objašnjeno u poglavlju 2.7, možemo krenuti sa stvaranjem osnovne Cordova aplikacije.

Prvi korak je odabir mjesta na kojem želimo stvoriti novi projekt koji će sadržavati datoteku izvornog koda te ostale potrebne datoteke. Na primjer, ako odlučimo stvoriti projekt na radnoj površini, u naredbeni redak potrebno je upisati ovu naredbu: `cd desktop` („cd“ označava *Change Directory*).

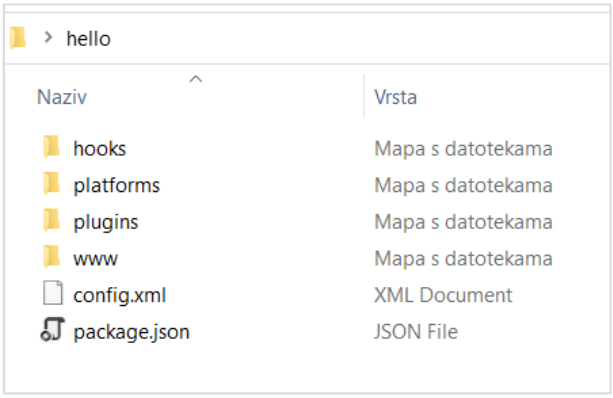


Slika 2.3 Cordova CLI - promjena direktorija

Zatim upišemo naredbu kojom će se kreirati novi projekt naziva *HelloWorld* u direktoriju *hello* sa identifikatorom *com.example.hello*:

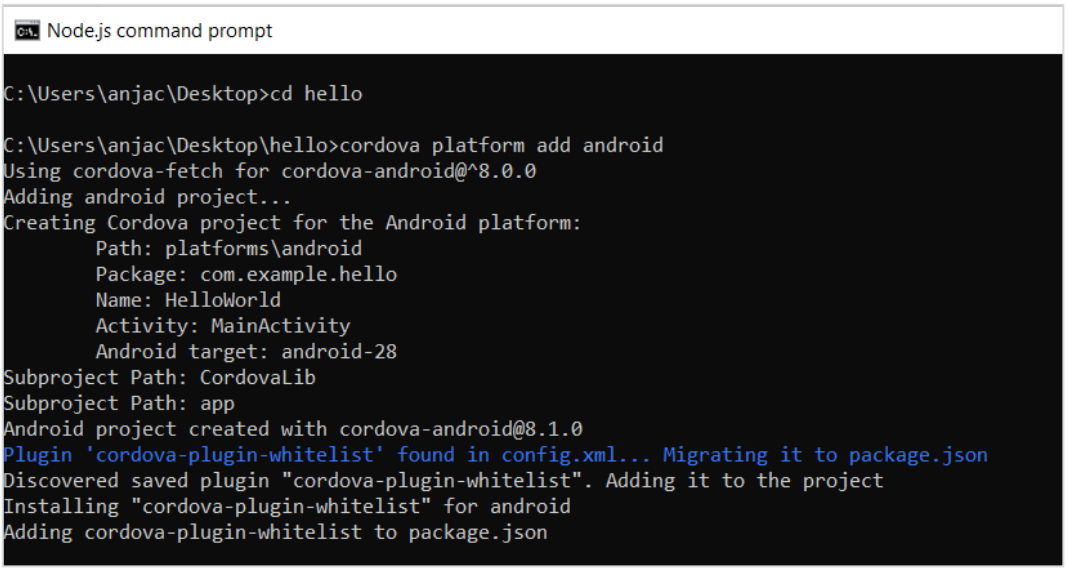
```
cordova create hello com.example.hello HelloWorld
```

Kreirani direktorij sadrži stavke prikazane na slici ispod:



Slika 2.4 Sadržaj direktorija "hello"

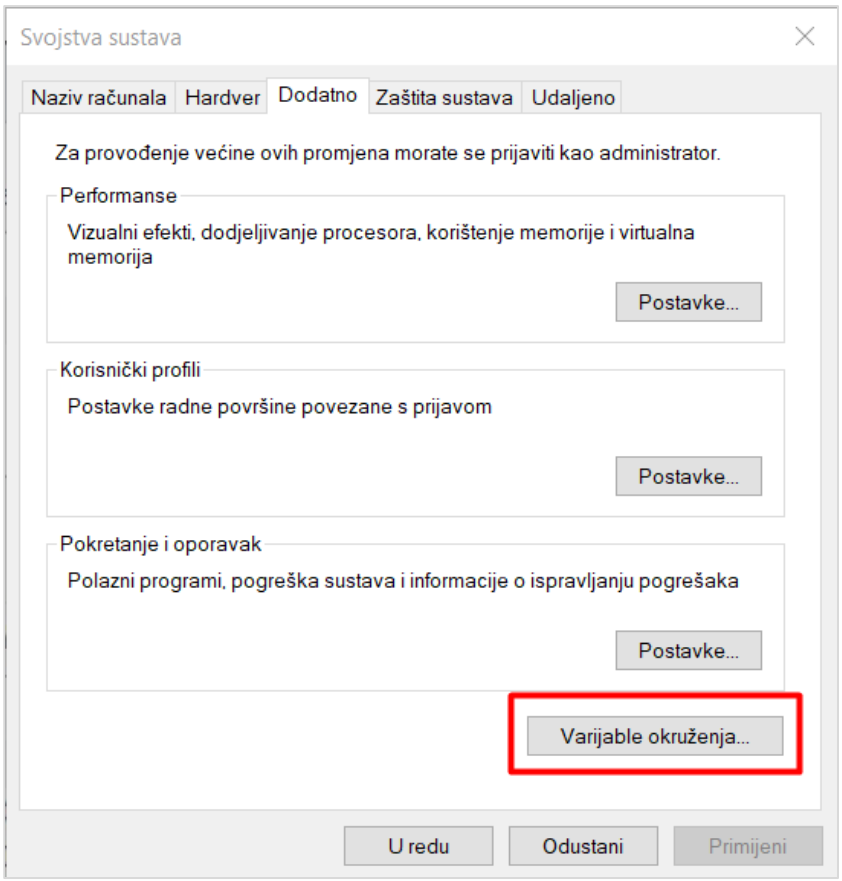
Idući korak je dodavanje platformi za koje želimo izgraditi aplikaciju, no prije toga moramo se prebaciti u novokreirani direktorij sa naredbom: `cd hello`. Sada kad se nalazimo u direktoriju *hello*, koristimo se naredbom `cordova platform add` i dodajemo željene platforme poput Androida, Windowsa ili iOS-a. Ja sam za ovaj primjer odlučila dodati Android platformu. Na slici Slika 2.5 možemo vidjeti da je ciljana verzija Android-28 tj. Android 9, a taj podatak će nam biti potreban kasnije kod instalacije Android SDK.



Slika 2.5 Cordova CLI - dodavanje Android platforme

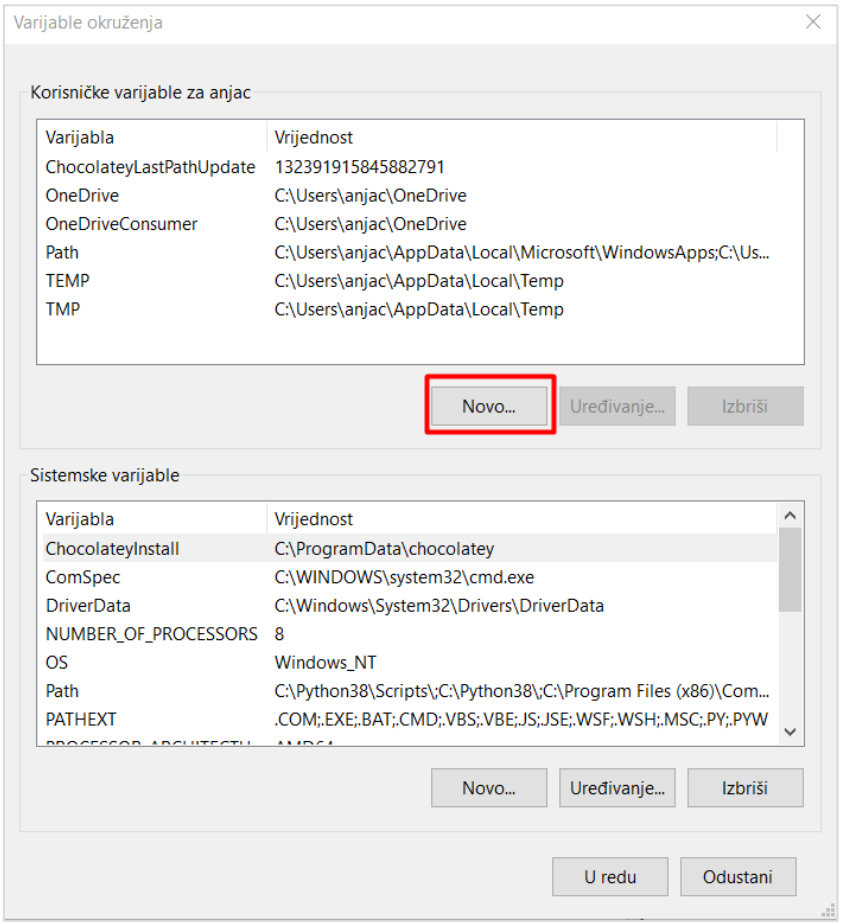
Kako bismo provjerili zadovoljavamo li sve uvjete izgradnje aplikacije za Android, upišemo naredbu `cordova requirements` i pojaviti će se popis zahtjeva platforme i što nam nedostaje.

Prvo moramo instalirati Java JDK 8 i ukoliko koristimo Windows računalo, postaviti novu varijablu okruženja (engl. *Environment Variable*) prema instalacijskom putu JDK. To napravimo na način da prvo u tražilicu Windowsa upišemo „Uređivanje varijabli okruženja sustava“ i kliknemo na ponuđeno. Zatim u otvorenom prozoru odaberemo „Varijable okruženja…“ i otvoriti će nam se prozor u kojem uređujemo korisničke varijable i sistemske varijable.



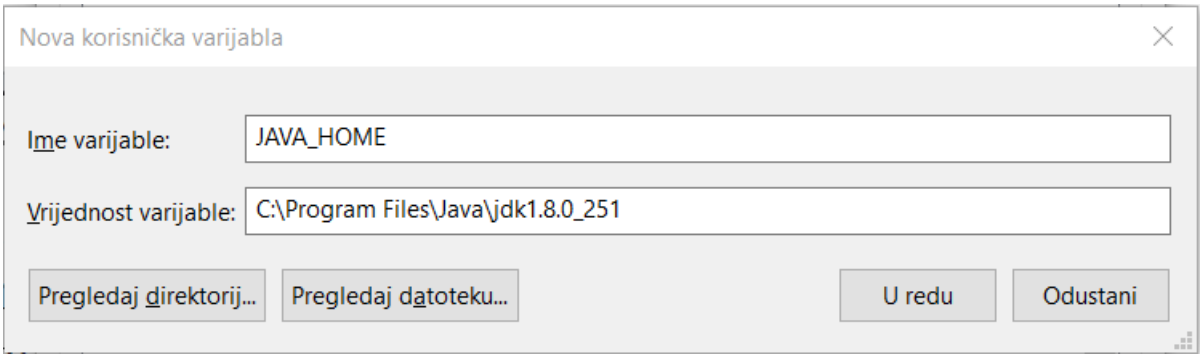
Slika 2.6 Varijable okruženja

Kada nam se otvori prozor „Varijable okruženja“ kliknemo *Novo* za dodavanje nove korisničke varijable (Slika 2.7).



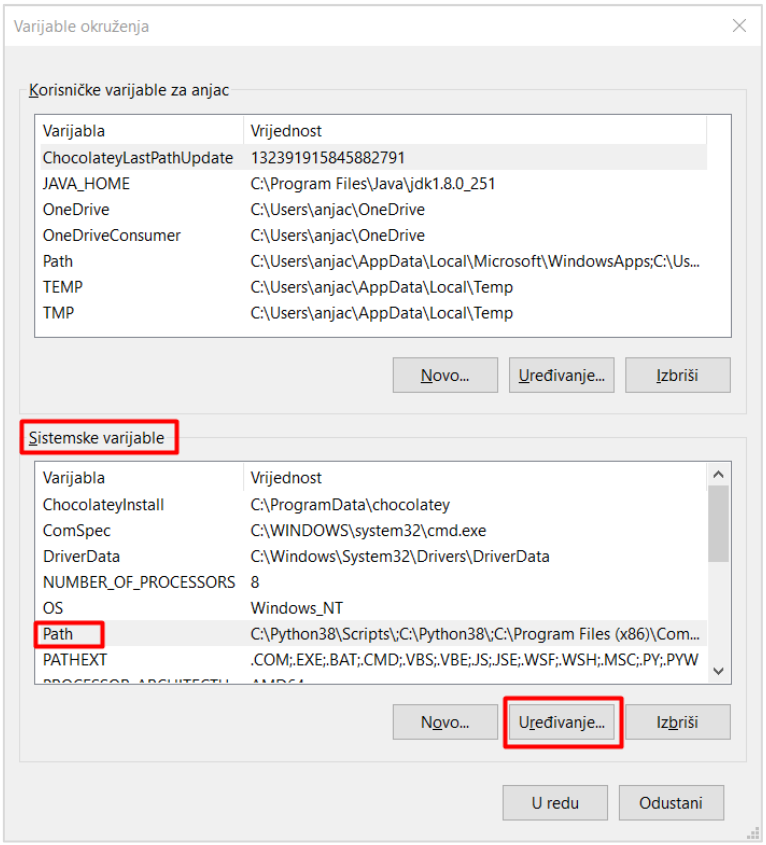
Slika 2.7 Dodavanje nove korisničke varijable okruženja

U novootvorenom prozoru ispunimo podatke kao što je prikazano na slici ispod s time da, ukoliko je potrebno, promijenimo lokaciju na kojoj se nalazi instalirani Java JDK. Klikom na „U redu“ kreirana je nova korisnička varijabla.



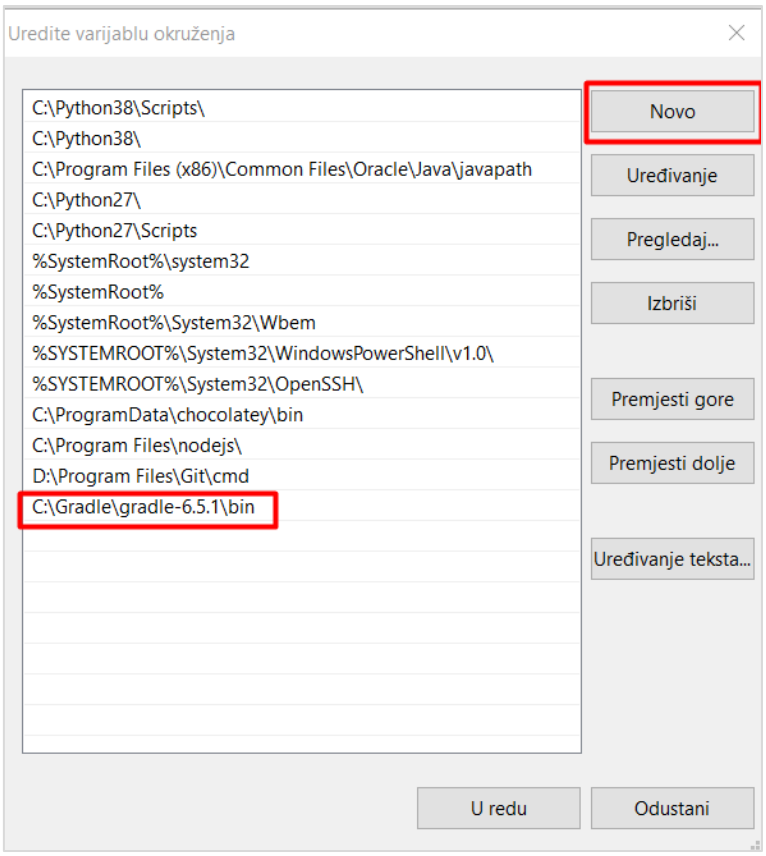
Slika 2.8 Dodavanje JAVA_HOME varijable

Dalje slijedi instalacija alata Gradle koji automatizira izgradnju aplikacije za višejezični razvoj softvera. Sa službene web stranice preuzmemo paket u .zip formatu, na C disku kreiramo novu mapu naziva „Gradle“ i u nju raspakiramo sadržaj preuzete komprimirane mape. Sada još moramo otvoriti „Varijable okruženja“ i pod sistemskim varijablama označiti *Path* i kliknuti na *Uređivanje.*



Slika 2.9 Uređivanje putanje sistemske varijable

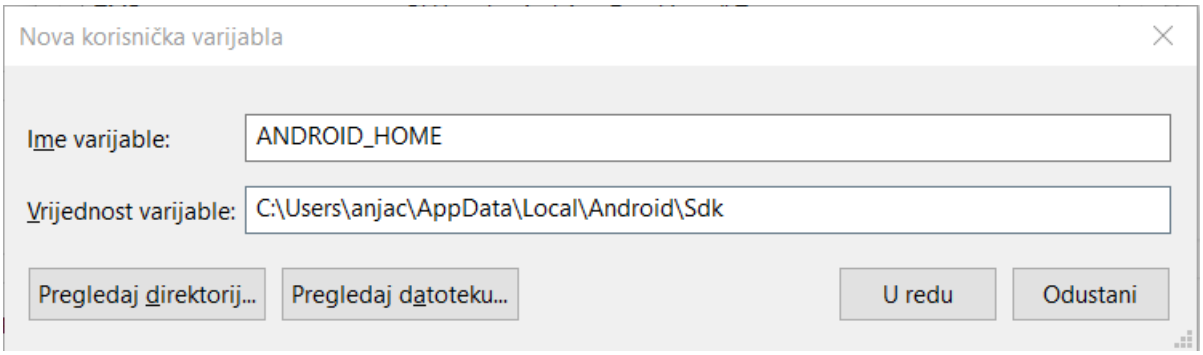
Otvoriti će se prozor u kojem možemo urediti putanju postojeće varijable okruženja ili dodati novu putanju. Kliknemo na gumb *Novo* i upišemo adresu prikazanu na slici Slika 2.10 sa ispravnom verzijom Gradle-a.



Slika 2.10 Dodavanje putanje Gradle-a

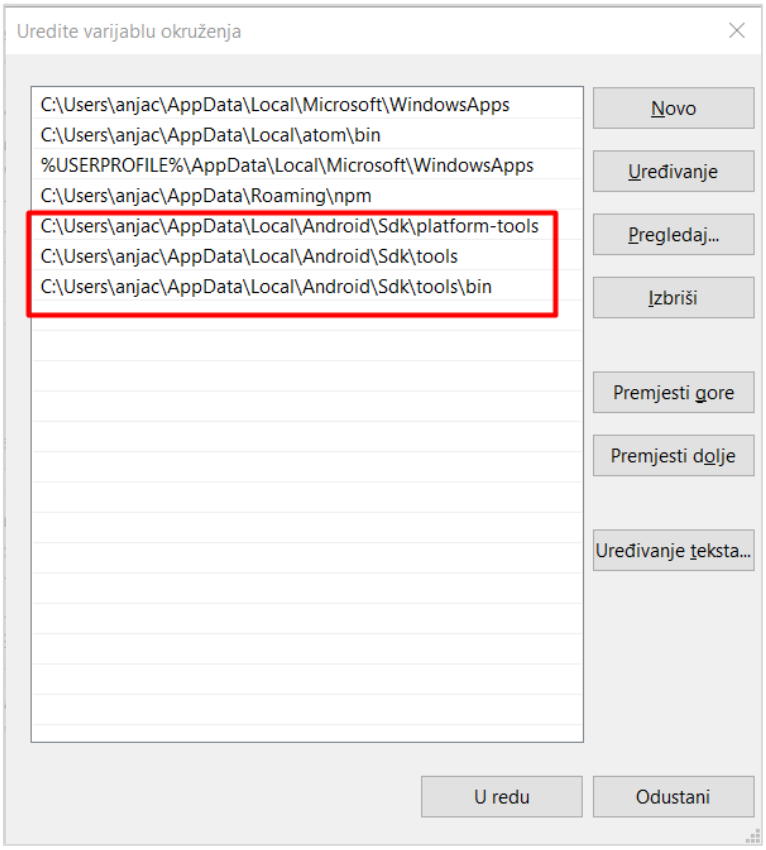
Posljednji zahtjev koji moramo zadovoljiti za izgradnju aplikacije na platformi Android je instalacija Android SDK alata i kreiranje simulatora. Proces započinje preuzimanjem i instalacijom razvojnog okruženja Android Studio. Kada je Android Studio instaliran i spreman za korištenje, otvorimo SDK Manager koji se nalazi na alatnoj traci Android Studija i označimo stavke koje želimo instalirati, a to su: Android 9.0 SDK, Android SDK Build-tools, Android Emulator, Android SDK Platform Tools, Android SDK Tools i HAXM Installer. HAXM je mehanizam za virtualizaciju široko korišten kao akcelerator za Android Emulator.

Nakon što su sve ove stavke instalirane, moramo dodati novu korisničku varijablu okruženja ANDROID_HOME sa adresom na kojoj se nalazi instalirani Android SDK (Slika 2.11).



Slika 2.11 Dodavanje ANDROID_HOME varijable

Sada trebamo označiti *Path* pod korisničkim varijablama, odabrati *Uređivanje* i dodati tri putanje prikazane na slici Slika 2.12 ispod.

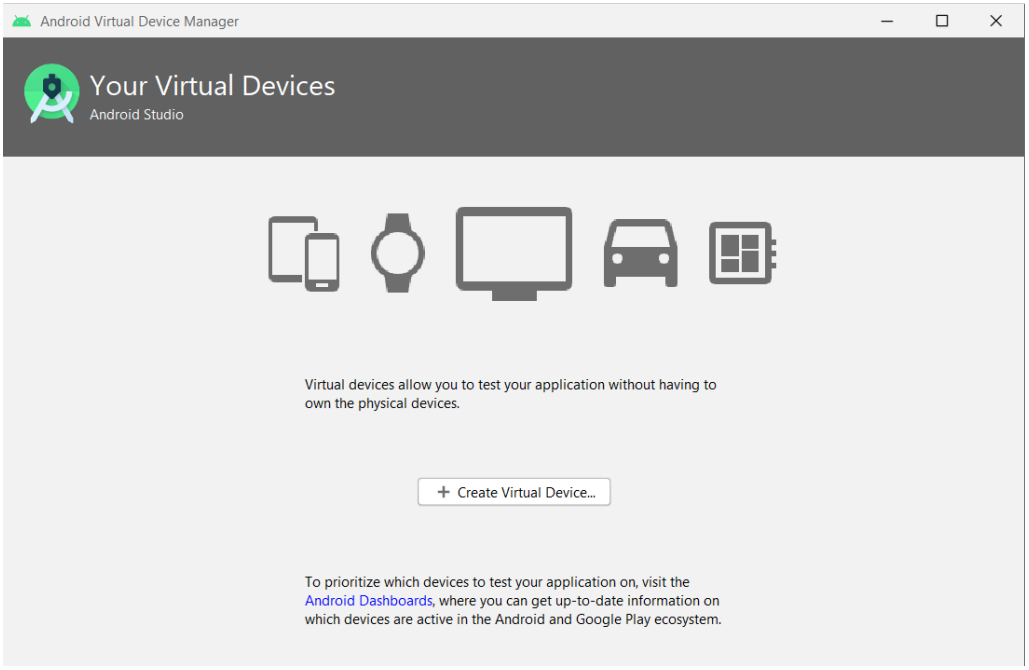


Slika 2.12 Dodavanje Android SDK putanji

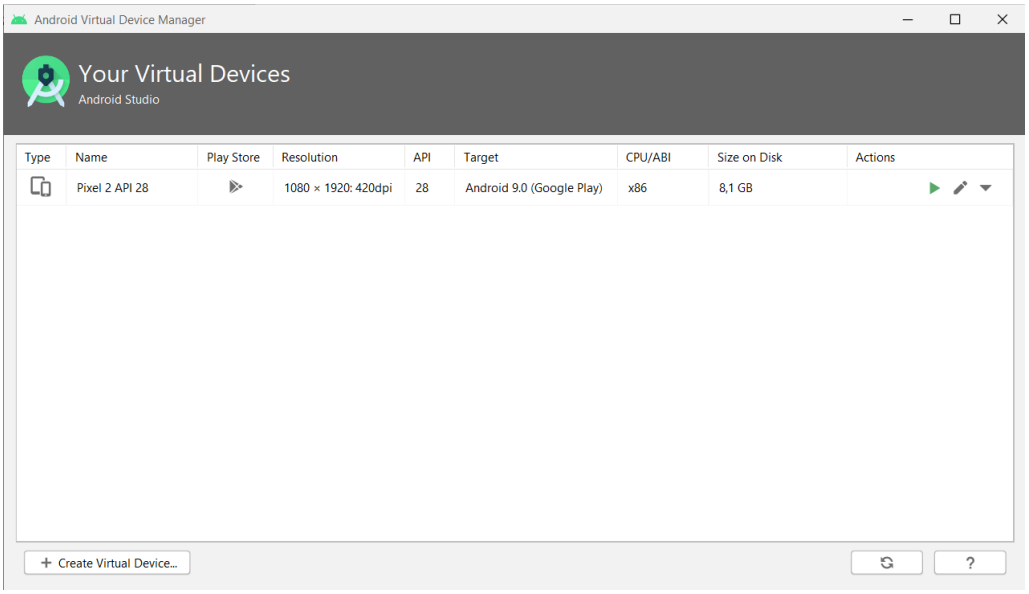
Vraćamo se u Node.js CMD i upišemo naredbu `cordova build` kako bismo izgradili aplikaciju za sve dodane platforme. Ukoliko je izgradnja bila uspješna, ispisati će nam se poruka „Build successful“

Sada je vrijeme za testiranje aplikacije. Aplikaciju možemo testirati preko internetskog preglednika pokretanjem datoteke *index.html* koja se nalazi u podmapi projekta naziva *www*, pokretanjem Android simulatora ili direktnim spajanjem našeg Android uređaja.

Za kreiranje novog virtualnog Android uređaja preko kojeg ćemo simulirati aplikaciju, otvorimo Android Studio te na alatnoj traci kliknemo na AVD Manager. Otvoriti će se sučelje na kojem odaberemo *Create Virtual Device*, a zatim pratimo upute i odabiremo vrstu uređaja, verziju Androida na uređaju te upišemo željeno ime simulatora.

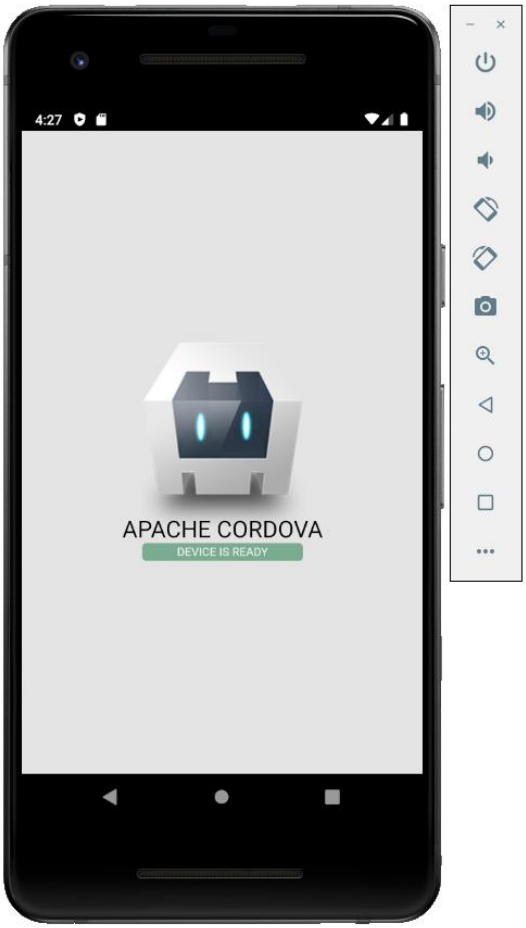


Slika 2.14 AVD Manager



Slika 2.13 Kreirani Android virtualni uređaj

Za testiranje preko simulatora najprije moramo pokrenuti virtualni uređaj preko AVD Managera u Android Studiju. Zatim u CMD upišemo naredbu `cordova emulate android` i ako smo sve korake napravili ispravno, aplikacija će se pokrenuti u simulatoru.



Slika 2.15 Simulacija Cordova aplikacije na virtualnom Android uređaju

Ako želimo testirati aplikaciju preko vlastitog Android uređaja, najprije moramo omogućiti opciju USB Debugging u postavkama uređaja te dopustiti instalaciju preko USB-a i preko nepoznatih izvora. Zatim spojimo Android uređaj sa računalom preko kabla i upišemo naredbu `cordova run android` kojom će se aplikacija instalirati i automatski pokrenuti na spojenom uređaju.

2.10. Zaključak – Apache Cordova

Apache Cordova je razvojni okvir koji omogućuje izradu hibridnih aplikacija uz pomoć često korištenih web tehnologija HTML-a, CSS-a i JavaScript-a. Time se pruža mogućnost razvoja mobilne aplikacije i onim programerima koji nemaju iskustva u izradi mobilnih aplikacija pomoću nativih jezika. Iako korištenjem Cordove možemo smanjiti obujam posla na način da ne moramo zasebno razvijati aplikaciju za svaku platformu i pritom koristiti nativni programski jezik, najveći problem bi mogao biti taj što kod Cordove i ostalih hibridnih rješenja izvršavanje koda neće biti brzo koliko bi bilo da je pisan nativnim jezikom uređaja. Također, ne preporuča se korištenje Cordove kod izrade grafički zahtjevnijih aplikacija i igara zbog problema sa brzinom, a što bi dovelo do lošeg korisničkog iskustva.

3. Razvojni okviri za hibridne aplikacije

Osim Apache Cordove, postoje još mnogi programski okviri koji koriste sličan pristup za izradu višeplatformskih aplikacija, a među najpoznatijima su:

- Ionic
- Xamarin
- Flutter
- Appcelerator Titanium

U nastavku su opisane značajke svakog od nabrojanih razvojnih programskih okvira.

3.1. Ionic

Najčešće korišten programski okvir za izradu hibridnih aplikacija je Ionic (https://ionicframework.com/), koji za razliku od Cordove dolazi sa ugrađenim korisničkim sučeljem. Fokusiran je na *frontend* interakciji korisničkog iskustva (UX) i korisničkog sučelja (UI) aplikacije. Može se integrirati sa ostalim bibliotekama ili razvojnim okvirima poput Angular, React ili Vue, a može se koristiti i samostalno. Ionic je dizajniran da radi i ispravno učitava aplikacije na svim novijim mobilnim uređajima i platformama te dolazi sa već pripremljenim komponentama, tipografijom i proširivom osnovnom temom koja se prilagodi svakoj platformi. Za pristup značajkama uređaja kao što su kamera, GPS ili svjetiljka, Ionic koristi dodatke Apache Cordove ili u novije vrijeme, dodatke CapacitorJS-a. [10] [11]

Uz UI okvir, Ionic ima i vlastiti CLI omot oko Cordova CLI-a, tako da imamo jednostavan način pokretanja novih Ionic projekata pomoću alata s kojim smo se već upoznali kod Cordove. Uz instaliranje i ažuriranje Ionica, CLI dolazi s ugrađenim razvojnim poslužiteljem, alatima za izgradnju te uklanjanje pogrešaka i još mnogo toga. Kako bismo mogli koristiti Ionic za našu Cordova aplikaciju, ne trebamo ništa dodatno preuzeti. Trebamo samo u naredbeni redak upisati sljedeću naredbu i na taj način instalirati Ionic CLI:

```
npm install -g ionic
```

Najbitnije značajke programskog okvira Ionic su [11]:

- **AngularJS**
  - Ionic koristi AngularJS MVC (*Model-View-Controller*) arhitekturu za izgradnju bogate *single-page* aplikacije optimizirane za mobilne uređaje
- **CSS komponente**
  - nude gotovo sve elemente potrebne mobilnoj aplikaciji; zadani stil može se prilagoditi vlastitim željama
- **JavaScript komponente**
  - proširuju CSS komponente sa JavaScript funkcionalnostima kako bi obuhvatile sve mobilne elemente koji se ne mogu napraviti samo korištenjem HTML-a i CSS-a
- **Cordova dodaci**
  - nude API potreban za korištenje nativnih funkcija uređaja JavaScript kodom
- **Ionic CLI**
  - ovo je uslužni program NodeJS-a sa naredbama za pokretanje, izgradnju i emulaciju Ionic aplikacija
- **Ionic pogled**
  - platforma za prijenos, dijeljenje i testiranje aplikacije na nativnim uređajima

3.2. Xamarin

Xamarin (https://dotnet.microsoft.com/apps/xamarin) je platforma otvorenog koda u vlasništvu Microsofta koja služi za razvoj nativnih višeplatformskih aplikacija pomoću .NET okvira i C# jezika. .NET je platforma za razvojne programere koja se sastoji od alata, programskih jezika i biblioteka koji služe za izgradnju različitih vrsta aplikacija, a sam Xamarin proširuje .NET sa alatima i bibliotekama potrebnim za razvoj aplikacija namijenjenih za Android, iOS, tvOS, watchOS, macOS i Windows. Neke od značajki koje Xamarin dodaje .NET platformi su [12]:

- Osnovni okvir za pristup izvornim značajkama
- *Extensible Markup Language* (XAML); jezik za izgradnju dinamičnih mobilnih aplikacija pomoću C#
- Biblioteke sa uobičajenim obrascima, kao što je MVVM (*Model View ViewModel*)
- Biblioteke specifične za platformu koje uključuju pristup API-jevima s Google-a, Apple-a ili Facebook-a za dodavanje bogatih mogućnosti
- Proširenja za uređivač koja pružaju označavanje sintakse, dovršavanje koda i ostale funkcionalnosti specifične za razvoj mobilnih aplikacija

Xamarin omogućava programerima da u prosjeku dijele 90% koda svoje aplikacije za različite platforme, a ostatak prilagode tako da postignu izvorne performanse, izgled i osjećaj za svaku platformu zasebno. Kombinirajući sposobnosti nativnih platformi, Xamarin dodaje niz značajki poput izravnog povezivanja Objective-C, Java, C i C++ biblioteka što daje mogućnost korištenja širokog raspona kodova trećih strana, moderne jezične konstrukcije i integrirano razvojno okruženje te podršku za tri najveće mobilne platforme i pristup nativnim API-jevima. Inženjeri Xamarina razvili su i nekoliko proizvoda koji pomažu kod razvoja višeplatformskih aplikacija: Xamarin.Android, Xamarin.iOS, Xamarin.Essentials i Xamarin.Forms. [13]

Xamarin.Essentials je biblioteka koja pruža višeplatformske API-je za pristup nativnim značajkama uređaja. Pojednostavljuje sam proces pristupa izvornim funkcionalnostima koje uključuju: informacije o uređaju, sustav datoteka, akcelerometar, biranje broja, pretvaranje teksta u govor i zaključavanje zaslona.

Xamarin.Forms je UI okvir otvorenog koda koji omogućava programerima izradu Xamarin.iOS, Xamarin.Android i Windows aplikacija iz jedne zajedničke baze kodova. Neke od značajki koje pruža Xamarin.Forms su: XAML jezik korisničkog sučelja, geste, efekti i stil.

3.3. Flutter

Flutter (https://flutter.dev/) je besplatan UI okvir otvorenog koda kojeg je razvio Google, a omogućava ponovnu upotrebu koda na operacijskim sustavima kao što su iOS i Android te istovremeno dopušta aplikacijama direktnu interakciju sa temeljnim uslugama platforme. Cilj je omogućiti programerima isporuku aplikacija visokih performansi koje pružaju nativan osjećaj na različitim platformama. [14]

Ovaj programski okvir čini nekoliko glavnih značajki [15]:

- **UI biblioteka temeljena na *widgetima***
  - Kombiniranjem različitih *widgeta* možemo izraditi čitavo korisničko sučelje. Svaki od widgeta definira strukturni element (npr. izbornik ili gumb), stilski element (npr. boje ili font), raspored stranice (npr. margine) i ostalo.
  - Flutter ne koristi izvorne widgete za svaku platformu, ali pruža programerima vlastite gotove widgete ili programeri mogu izraditi svoje widgete prema želji
- **Dart jezik**
  - Za razvoj aplikacije preko Flutter-a koristi se Google-ov programski jezik Dart koji je fokusiran na *frontend* razvoj. Dart je objektno-orijentiran programski jezik, a njegova sintaksa se može usporediti sa JavaScript-om
  - Dart dolazi sa repozitorijem softverskih paketa za proširenje mogućnosti aplikacija
- ***Hot reload***
  - Opcija „Hot reload“ omogućava brzu i jednostavnu izgradnju korisničkog sučelja, dodavanje raznih funkcionalnosti i ispravljanje grešaka
  - Automatski osvježava ažurirane dijelove aplikacije bez potrebe za ponovnim učitavanjem cijelog prozora

Flutter je zapravo sveobuhvatan SDK za stvaranje aplikacija. Pruža mehanizam za prikazivanje, UI komponente, testne okvire, razne alate i ostale značajke koje su potrebne za izradu aplikacije.

3.4. Appcelerator Titanium

Titanium (https://www.appcelerator.com/) je programski okvir otvorenog koda koji je razvila tvrtka Appcelerator 2008. godine, a omogućava stvaranje nativnih aplikacija za platforme poput iOS-a, Androida i Windowsa iz jedne JavaScript baze podataka. Koristi veliku količinu izvornih kontrola uređaja koje se mogu naći na iOS i Android platformama te time omogućuje programeru kreiranje aplikacije s bogatim funkcionalnostima kao i one za čiju se izradu koristi nativni jezik. Titanium-om se zapravo pokušava postići ponovna uporaba koda s objedinjenim JavaScript API-jem, sa značajkama specifičnih za platformu i izvornim performansama koje će ispuniti očekivanja korisnika. Osnovna komponenta Titanium-a je paket za razvoj softvera (SDK) pod Apacheovom licencom nazvan Titanium SDK. Za razvoj aplikacija pomoću Titanium-a, programer mora instalirati nativne alate za ciljane platforme, a zatim komunicira Titanium SDK CLI-jem koji je baziran na Node-u. Komunikacija se odvija direktno putem naredbenog retka ili preko Titanium Studija, razvojne okoline bazirane na Eclipse IDE. Za razvoj aplikacije pomoću Titanium-a ne koriste se HTML i CSS, nego MVC okvir Alloy koji organizira kod napisan u XML-u, JSON objekti te JavaScript. [16] [17]
Glavne značajke Titanium SDK uključuju:

- Višeplatformski API za pristup nativnim UI komponentama poput navigacijskih traka, izbornika i dijaloških okvira te nativnim funkcionalnostima uređaja uključujući datotečni sustav, mrežu, geolokaciju, akcelerometar i karte
- Transparentni pristup nativnim funkcionalnostima kojeg pokrivaju prevoditelj koda Hyperloop i nativni moduli
- Alloy okvir baziran na MVC obrascu

Titanium pruža API-je za:

- Upotrebu značajki specifičnih za hardver
- Korištenje kontrola specifičnih za operacijski sustav
- Sudjelovanje u ekosustavu platforme, npr. korištenje mehanizma obavijesti prikladnih za platformu

3.5. Zaključak – Ostali razvojni programski okviri

Osim Cordove, još neki od programskih okvira koji omogućuju izradu višeplatformske hibridne aplikacije su Ionic, Xamarin, Flutter i Titanium. Od nabrojanih okvira većina koristi HTML, CSS i JavaScript za razvoj aplikacije, osim Xamarina koji koristi .NET i C# te Titaniuma kod kojeg Alloy okvir organizira kod napisan u XML-u, JSON-u i JavaScript-u. Svaki okvir na neki način ima pristup nativnim značajkama uređaja. Ionic nativnim značajkama pristupa preko Cordove, a Flutter pristupa samo ograničenom broju značajki. Svaki programer može prema značajkama pojedinog okvira odabrati na koji način želi razviti hibridnu aplikaciju. Tablica 3-1 na jednostavniji način prikazuje usporedbu Apache Cordove sa ostalim programskim okvirima.

	Cordova	**Ionic**	**Xamarin**	**Flutter**	**Titanium**
Vlasnik	Apache	Drifty Co.	Microsoft	Google	Appcelerator
Prvo izdanje	2009. (PhoneGap)	2013.	2011.	2017.	2008.
Tehnologije za razvoj aplikacije	HTML, CSS, JavaScript	HTML, CSS, JavaScript, AngularJS	.NET, C#	Dart, HTML, CSS, JavaScript	XML, JSON, JavaScript
Osnovne značajke	API dodaci, klijentski nativni aplikacijski spremnik, Cordova CLI, WebView	UI okvir, integracija sa ostalim okvirima, Ionic CLI, CSS i JS komponente	Pristup izvornim značajkama, proširuje .NET, UI okvir	UI biblioteka, widgeti, *Hot reload*, testni okviri	Titanium SDK, Titanium Studio, višeplatformski API, Alloy okvir

Tablica 3-1 Usporedba Apache Cordove sa sličnim programskim okvirima

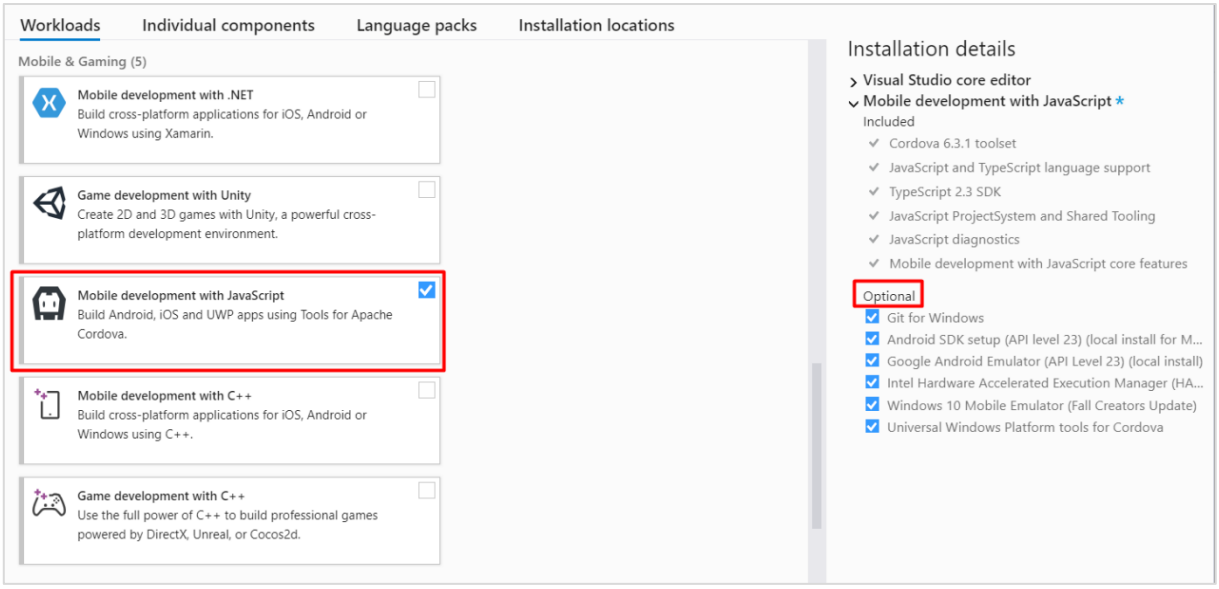
4. Izrada hibridne aplikacije za vremensku prognozu

Za praktični dio ovog rada odlučila sam napraviti aplikaciju za vremensku prognozu koja se temelji na Apache Cordovi. Odabrala sam Apache Cordovu jer nemam prijašnjeg iskustva u izradi nativnih mobilnih aplikacija, a okviri za razvoj hibridnih aplikacija uvelike pomažu u razvoju aplikacija za razne platforme. Aplikacija koristi geolokaciju uređaja pa sam to vidjela kao dobar primjer na kojemu mogu prikazati kako ubaciti i koristiti Cordova dodatake u aplikaciji. Korisnicima će uz automatsko pronalaženje lokacije uređaja, biti omogućeno i upisivanje željene lokacije za koju žele vidjeti trenutne vremenske uvjete. Umjesto načina razvoja Cordova projekta kod kojeg se naredbe upisuju u CMD, odabrala sam nešto moderniji pristup. Postoji više razvojnih okolina koje olakšavaju izradu Cordova projekta, među kojima je i Visual Studio 2017 kojeg sam ja odlučila koristiti. U Visual Studio mogu se dodati Apache Cordova alati koji omogućuju programerima stvaranje, upravljanje i testiranje Cordova projekata na Android-u, iOS-u i Windowsu. Pri izradi aplikacije koristila sam web tehnologije HTML, CSS, JavaScript, biblioteku jQuery i Bootstrap, alat Figmu te OpenWeatherMap API i Google Places API. U ovome radu prikazan je razvoj aplikacije za platforme Android i Windows, ali ne i za iOS jer bi nam za to bilo potrebno Macintosh računalo.

4.1. Instalacija Visual Studija

Visual Studio je Microsoftovo integrirano razvojno okruženje (IDE) koje se koristi za razvoj računalnih programa, web stranica, web i mobilnih aplikacija te web usluga. Dolazi u nekoliko verzija, a to su: Community, Professional i Enterprise. Community je besplatan i preporuča se za studente te za samostalan razvoj aplikacija, Professional je za manje timove i dolazi s dodatnim mogućnostima, a Enterprise je namijenjen većim tvrtkama. Professional i Enterprise se plaćaju te imaju besplatno probno razdoblje od nekoliko mjeseci. Odabrala sam Community upravo zbog toga što je besplatan.

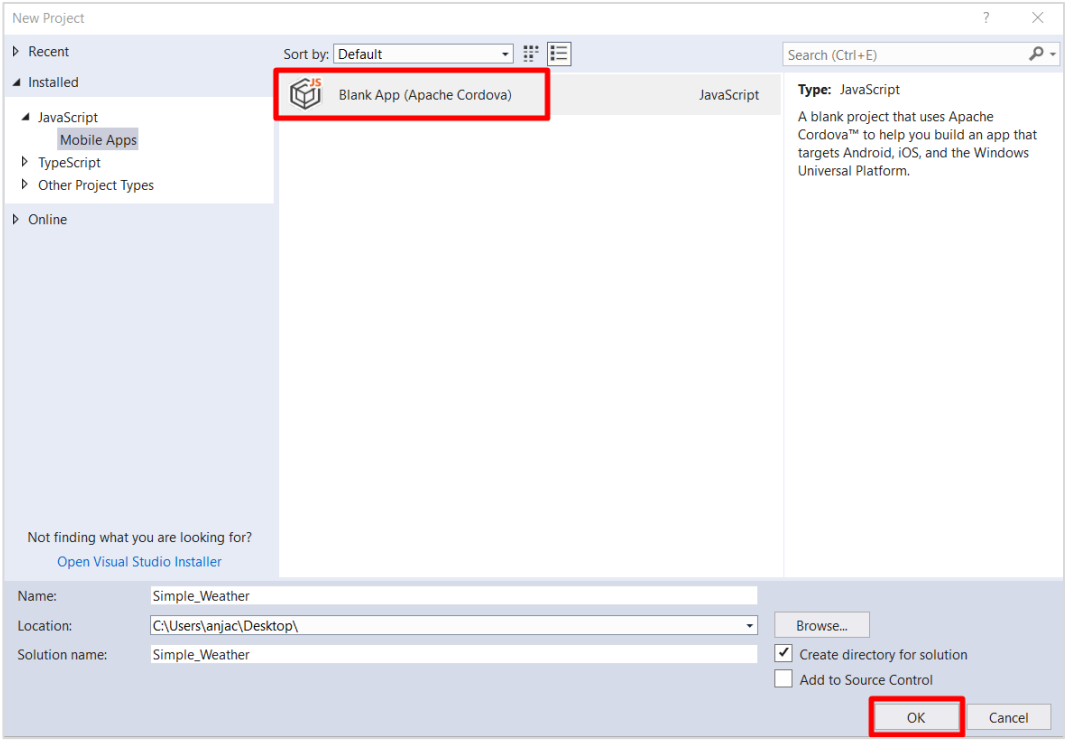
Proces instalacije Visual Studija počinje preuzimanjem željene verzije sa službene stranice. Kada prilikom instalacije dođemo do koraka gdje imamo ponuđen popis dodataka koje želimo dodati u Visual Studio, odaberemo „Mobile development with JavaScript“ (Slika 4.1) koji će nam omogućiti izgradnju aplikacije sa Apache Cordovom te na desnoj strani označimo da želimo instalirati i sve dodatne stavke koje su potrebne su da bismo mogli pokrenuti simulaciju Android ili Windows sustava. Zasebno moramo instalirati još Java JDK te dodati varijablu okruženja kao što je prethodno opisano u poglavlju „Stvaranje osnovne Cordova aplikacije“.



Slika 4.1 Visual Studio - instalacija alata za Apache Cordovu

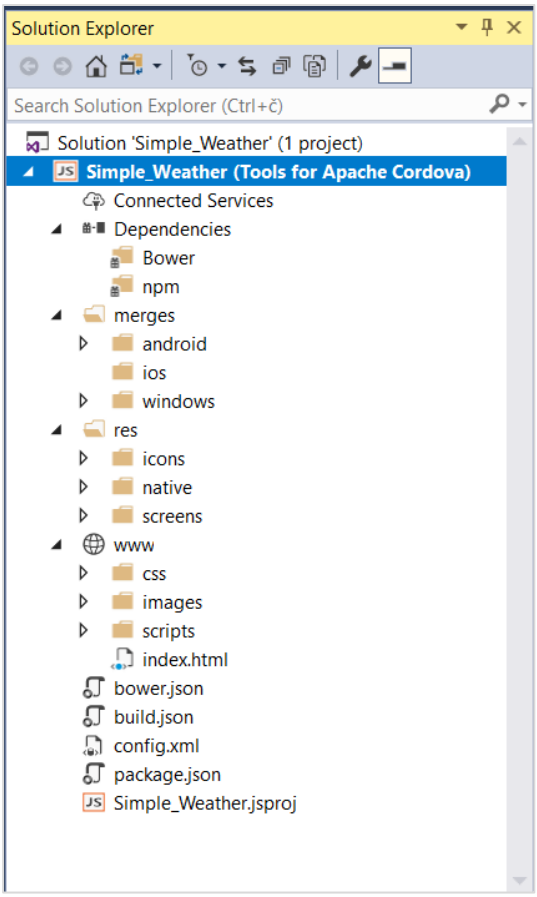
4.2. Kreiranje novog Apache Cordova projekta

Kako bismo kreirali novi Cordova projekt u Visual Studiju, odaberemo *File* > *New* > *Project* te u otvorenom prozoru odaberemo *Blank App (Apache Cordova)* i upišemo naziv našeg projekta (Slika 4.2).



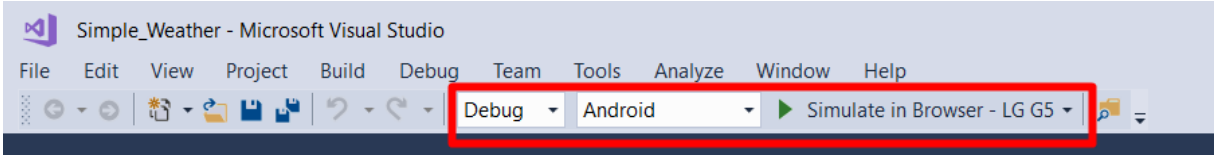
Slika 4.2 Visual Studio - kreiranje novog Apache Cordova projekta

Tijekom procesa stvaranja projekta, Visual Studio kreira rješenje (engl. *Solution*) za projekt, a zatim koristi Cordova CLI i ostale alate za kreiranje strukturirane mape projekta Cordova aplikacije, dodaje platforme (Android, iOS i Windows) i popunjava aplikaciju sa zadanim sadržajem iz predloška. Na slici ispod, prikazan je „Solution Explorer“ sa mapama i datotekama koje novokreirani projekt sadrži, a koje su detaljnije objašnjene u sljedećem poglavlju.



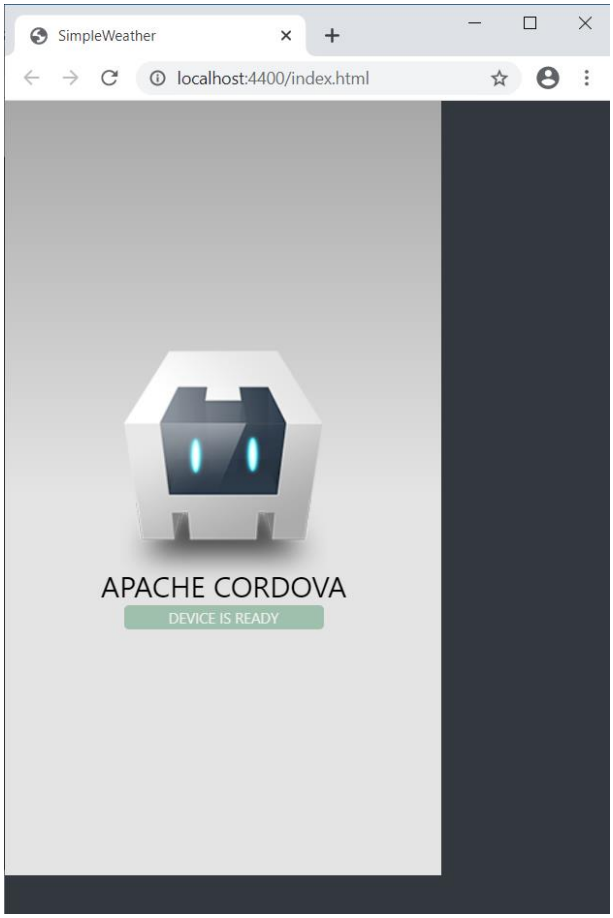
Slika 4.3 Visual Studio - Solution Explorer

U ovom trenutku možemo pokrenuti simulaciju projekta da bi vidjeli kako aplikacija trenutno izgleda i da bi provjerili da li se ispravno pokreće. Na alatnoj traci odaberemo simulator preko kojeg želimo pokrenuti aplikaciju i kliknemo zelenu strelicu za pokretanje (Slika 4.4).



Slika 4.4 Visual Studio - Pokretanje aplikacije preko simulatora

Ako odaberemo simulaciju preko web preglednika, prikazati će nam se sadržaj aplikacije u prozoru preglednika s veličinom odabranog uređaja. Prikazuje se zadana aplikacija koja nema nikakve mogućnosti interakcije, nego samo javlja da li se aplikacija ispravno inicijalizirala.



Slika 4.5 Visual Studio - Simulacija preko preglednika

Pokretanjem aplikacije u Visual Studiju otvara se *DOM Explorer* kojim možemo mijenjati različite elemente i atribute te mijenjati kod dok je aplikacija pokrenuta i vidjeti promjene uživo. Izmjene u DOM Exploreru ne utječu na „pravi“ kod pa možemo isprobavati kako bi razne izmjene koda utjecale na aplikaciju. Također, pokrenuti će se *JavaScript Console* gdje možemo vidjeti ukoliko ima kakvih grešaka u izvođenju programa. Simulacija se prekida klikom na gumb Stop na alatnoj traci ili zatvaranjem prozora u kojem je otvorena aplikacija.

4.3. Mape i datoteke projekta

Kao što je spomenuto u prethodnome poglavlju, u ovome poglavlju opisane su mape i datoteke koje su uključene u novokreirani Cordova projekt.

4.3.1. Mapa www

Ovo je nama najbitnija mapa jer će se unutar nje odvijati veći dio razvoja aplikacije. Sadrži HTML, CSS i JavaScript datoteke koje Cordova SDK pakira u mobilnu aplikaciju. Ovdje dodajemo i sve slike, ikone te ostali medijski sadržaj koji želimo uključiti u našu aplikaciju. Glavna ulazna točka aplikacije je datoteka index.html koja se prema zadanim postavkama učita kada se pokrene Cordova aplikacija.

CSS i JavaScript datoteke raspoređene su u mape „css“ i „scripts“. U mapi css nalazi se standardna index.css datoteka, a u nju također dodajemo i ostale .css datoteke koje želimo uključiti u projekt. U scripts mapu pohranjuju se sve .js datoteke koje koristi naš projekt, a na početku sadrži index.js datoteku čiji kod inicijalizira Cordova aplikaciju te datoteku platformOverrides.js koja funkcionira zajedno sa sličnim datotekama u mapi *merges*. [18]

4.3.2. Mapa merges

Mapu merges programeri mogu koristiti kako bi isporučili različiti sadržaj s obzirom na ciljanu mobilnu platformu. Sadrži tri podmape za svaku ciljanu platformu: Android, iOS i Windows. Sav sadržaj u tim mapama biti će kopiran u glavnu www mapu tijekom procesa pripreme, dodavanjem ili zamjenom sadržaja u mapi web aplikacije. Na primjer, ako dodamo sadržaj u *merges/android/scripts/platformOverrides.js*, taj sadržaj će se kopirati u *www/scripts/platformOverrides.js* i izmijeniti ili prebrisati postojeći sadržaj samo kada se aplikacija izgrađuje za Android platformu. [18]

4.3.3. Mapa res

U mapu res pohranjuju se resursi koji nisu dio web aplikacije, ali koristi ih nativna mobilna aplikacija poput ikona aplikacije, slike zaslona učitavanja, certifikata i ostalo. [18]

4.3.4. Datoteka bower.json

Bower.json je konfiguracijska datoteka za upravitelj paketa Bower koji se koristi za instalaciju paketa i biblioteka u web aplikaciji. Preporučeno je koristiti ovaj upravitelj paketa za biblioteke na strani klijenta jer ga aktivno održavaju autori raznih JavaScript i CSS biblioteka i dobro funkcionira na Windows, Mac OS X i Linux platformama. [18]

4.3.5. Datoteka build.json

Ovo je konfiguracijska datoteka za Android i iOS procese izgradnje aplikacije. Ne smijemo je obrisati jer ju Cordova SDK koristi za izgradnju potpisanih paketa mobilnih aplikacija koje tada možemo objaviti u trgovini aplikacija. U kasnijem poglavlju, detaljnije će biti objašnjen postupak izgradnje potpisanog paketa aplikacije. [18]

4.3.6. Datoteka config.xml

Datoteka config.xml sadrži postavke koje definiraju svojstva nativne Cordova mobilne aplikacije uključujući ime aplikacije, konfiguraciju dodataka, sigurnosne postavke i ostalo. Ovu datoteku također ne smijemo obrisati jer Cordova SDK neće biti u mogućnosti izgraditi i implementirati aplikaciju bez nje. U „Visual Studio alatima za Apache Cordovu“ ova datoteka dolazi sa posebnim uređivačem sa sučeljem na kojem lako možemo mijenjati postavke bez direktnog uređivanja koda. [18]

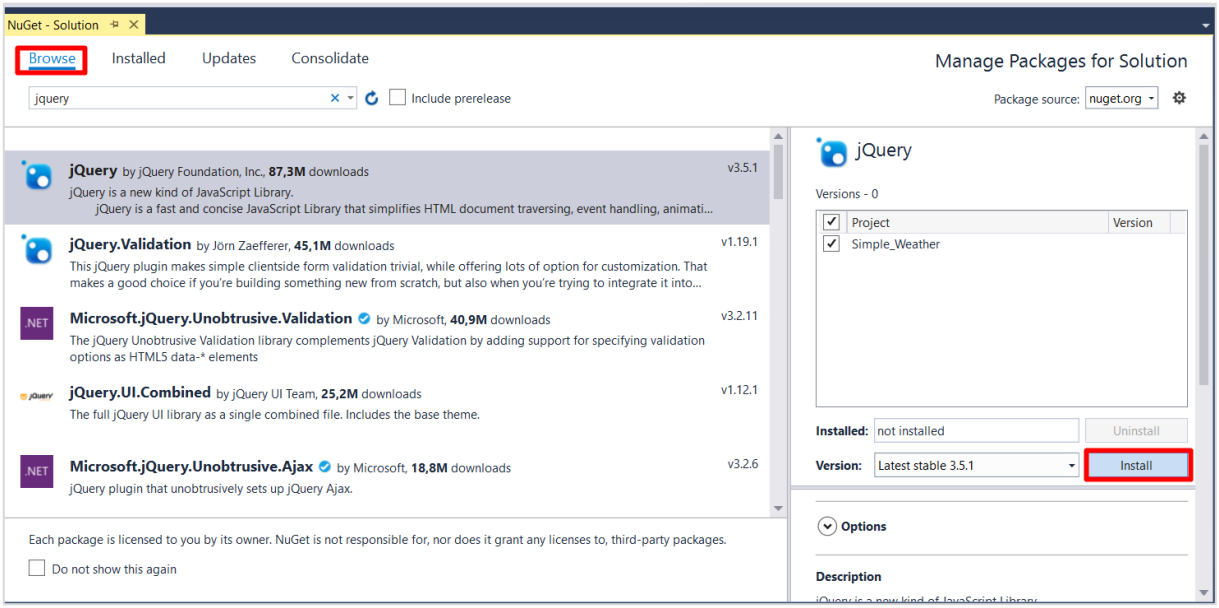
4.3.7. Datoteka package.json

Package.json je konfiguracijska datoteka za Node Package Manager (npm). Iako ovu datoteku Cordova projekti trenutno ne koriste često, s vremenom će postati važnija jer se planira zamjena datoteke config.xml sa package.json u budućoj verziji Cordove. [18]

4.4. Dodavanje jQuery biblioteke u projekt

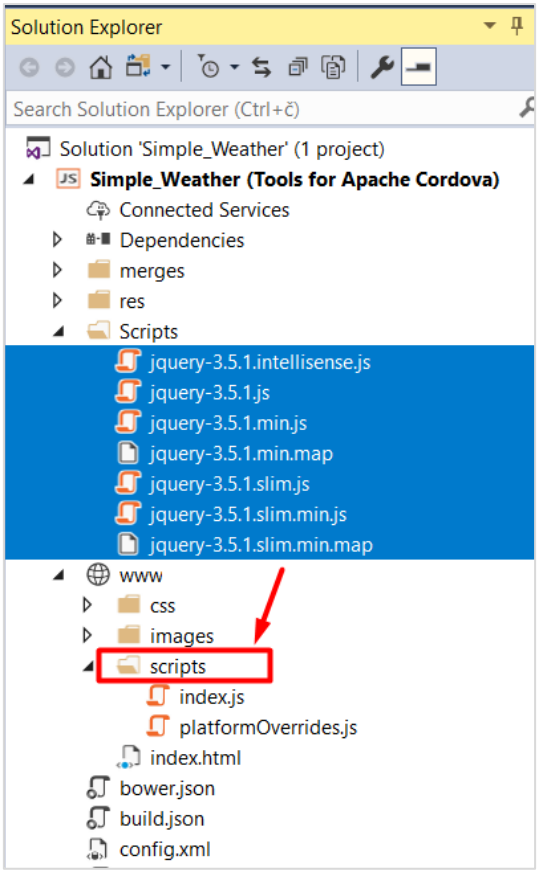
JQuery je JavaScript biblioteka koja smanjuje količinu koda kojeg je potrebno napisati, pojednostavljuje navigaciju dokumentom, odabir DOM elemenata, stvaranje animacija te omogućava stvaranje moćnih dinamičnih web stranica i aplikacija. [19] Iz tih razloga, jQuery se koristi i pri izradi ove aplikacije.

Kako bismo u projekt dodali jQuery, u izborniku Visual Studija odaberemo *Tools* > *NuGet Package Manager* > *Manage NuGet Packages for Solution.* Otvoriti će se NuGet Solution prozor u kojem u traku za pretraživanje upišemo „jQuery“ i odaberemo prvu ponuđenu opciju. Sa desne strane označimo naš projekt u koji želimo dodati jQuery te kliknemo Install.



Slika 4.6 Visual Studio - Instalacija jQuery biblioteke

Završetkom instalacije, NuGet u projekt dodaje novu mapu *Scripts* koja sadrži sve dodane jQuery datoteke. Te datoteke potrebno je premjestiti u *www/scripts*.



Slika 4.7 Premještaj instaliranih jQuery datoteka

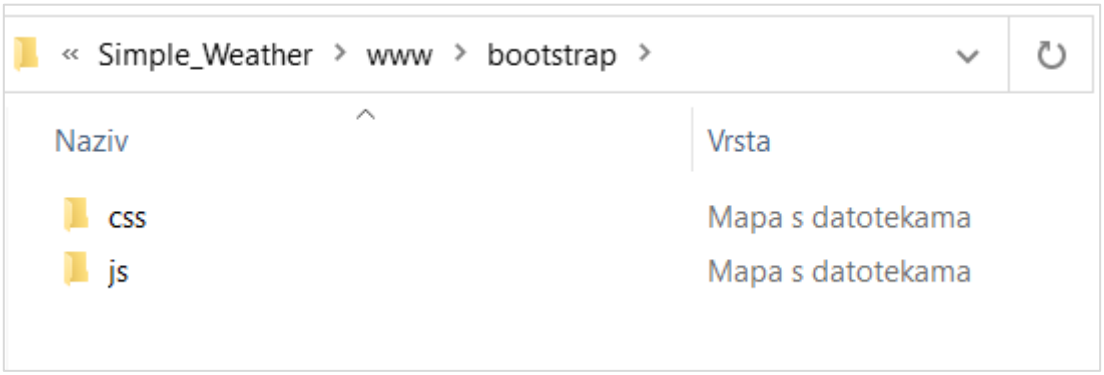
Zadnji korak u dodavanju jQuery-a u projekt je uključivanje biblioteke u HTML kod tj. u index.html datoteku koja se nalazi u *www* mapi. Na dno <body> sekcije, prije ostalih <script> referenci, dodamo ovaj redak:

```
<script src="scripts/jquery-3.5.1.min.js"></script>
```

4.5. Dodavanje Bootstrap-a u projekt

U ovome projektu koristiti će se i okvir Bootstrap koji služi za razvoj responzivnih web stranica sa *mobile-first* pristupom te će ubrzati proces dizajna aplikacije.

Bootstrap u naš projekt možemo dodati preko NuGet Package Managera kao što smo dodali i jQuery, no na taj način će se u projekt instalirati veliki broj datoteka koje nam nisu potrebne i koje će bespotrebno povećati veličinu projekta. Zato ćemo Bootstrap dodati na način da prvo preko službene stranice https://getbootstrap.com/ preuzmemo komprimiranu mapu koja sadrži .css i .js datoteke koje ćemo dodati u naš projekt. U www mapi napravimo novu podmapu naziva „bootstrap“, raspakiramo sadržaj preuzete komprimirane mape te podmape „js“ i „css“ prebacimo u novokreiranu mapu.



Slika 4.8 Sadržaj mape bootstrap

Kada smo napravili prethodno opisane korake, slijedi dodavanje potrebnih redaka u postojeći *index.html* dokument da bismo mogli koristiti Bootstrap pri izradi aplikacije.

U <head> sekciju potrebno je ubaciti ovaj redak iznad postojeće index.css reference:

```
<link rel="stylesheet" href="bootstrap/css/bootstrap.min.css" />
```

a na dno <body> sekcije ubacimo ova dva retka ispod prethodno dodane jQuery reference:

```
<script src="bootstrap/js/bootstrap.bundle.js"></script>
<script src="bootstrap/js/bootstrap.min.js"></script>
```

4.6. Dizajn aplikacije

Prije nego krenemo sa kodiranjem aplikacije, potrebno je napraviti skicu dizajna da odmah na početku znamo što želimo postići samim kodiranjem. Za izradu skice mogu se koristiti razni specijalizirani alati kao što su: Figma, Adobe XD ili Sketch. Za potrebe ove aplikacije, odlučila sam koristiti Figmu (https://www.figma.com/) koja je bazirana na webu pa je nije potrebno preuzimati i dostupna je svakome.

Dizajn ove aplikacije biti će jednostavan, kao i sama aplikacija. Sadržavati će podatke o trenutnoj temperaturi i vremenskim uvjetima, tlak, vlažnost zraka te brzinu vjetra. Uz sve to, pri vrhu će se nalaziti polje za unos mjesta za koje želimo dohvatiti podatke. Za pozadinu aplikacije odabrala sam plavi gradijent, tekst i ikone u bijeloj boji te font *Roboto*.

Slika 4.9 Figma - dizajn aplikacije

4.7. Ikona aplikacije

Kako bi vizualni identitet aplikacije bio potpun, još je potrebno izraditi ikonu same aplikacije. Pri izradi ikone korišten je Adobe Illustrator, a pomoću alata na stranici https://appiconmaker.co/ su generirane ikone u različitim dimenzijama kako bi odgovarale svakom Android i iOS uređaju. Generirane ikone preuzmu se i njima zamijenimo postojeće zadane Cordova ikone u mapama *android, ios i windows* koje se nalaze u *res/icons.*

Slika 4.10 ikona aplikacije

Ikone aplikacije raznih veličina za platforme Android, iOS i Windows definirane su u konfiguracijskoj datoteci config.xml. Na primjeru ispod (Kod 4-1) prikazano je kako su definirane ikone za Android:

```
<platform name="android">
    <icon src="res/icons/android/icon-36-ldpi.png" density="ldpi" />
    <icon src="res/icons/android/icon-48-mdpi.png" density="mdpi" />
    <icon src="res/icons/android/icon-72-hdpi.png" density="hdpi" />
    <icon src="res/icons/android/icon-96-xhdpi.png" density="xhdpi" />
</platform>
```

Kod 4-1 config.xml - implementacija Android ikona

4.8. Programiranje i kodiranje

Sada kada smo ubacili potrebne alate u projekt te napravili skicu dizajna aplikacije možemo krenuti sa uređivanjem postojećeg i dodavanjem novog koda, ali prvo trebamo sa weba preuzeti ikone pretraživanja, tlaka, vlažnosti i vjetra i ubaciti ih u mapu *www/images*. Ikone za ovu aplikaciju pronađene su i preuzete sa https://www.flaticon.com/. Te ikone ubaciti će se u HTML kod aplikacije.

4.8.1. HTML i CSS

Najprije krećemo sa uređivanjem *index.html* dokumenta. Prethodno smo u isti dokument dodali reference za jQuery i Bootstrap, a sada ćemo dodati dio koda kojim ćemo korisniku omogućiti upisivanje lokacije te prikaz podataka.

Dio koda u sekciji sa klasom *„app“* nam ne treba i moramo ga obrisati jer je namijenjen za zadanu Cordova aplikaciju:

```
<div class="app">
    <h1>Apache Cordova</h1>
    <div id="deviceready" class="blink">
        <p class="event listening">Connecting to Device</p>
        <p class="event received">Device is Ready</p>
    </div>
</div>
```

Kod 4-2 index.html - dio koda zadane aplikacije kojeg moramo obrisati

Umjesto njega u <body> upisujemo kod kojim ćemo u aplikaciju dodati obrazac za traženje željenog mjesta i podatke koje želimo da naša aplikacija prikazuje. Cijeli kod koji ćemo ubaciti u glavnu sekciju sa klasom *„container-fluid“* prikazan je na sljedećoj stranici, a u nastavku poglavlja opisani su dijelovi toga koda.

```
<div class="container-fluid">
       <div class="row">
            <div class="col">
                <div class="bg-white rounded-pill mx-3 my-2">
                    <div class="input-group">
                        <input type="search" placeholder="Unesite lokaciju..."
                               aria-describeby="find-weather-btn"
                               class="form-control border-0 "
                              id="city-name-input" name="city" required>
                        <div class="input-group-append">
                            <button id="find-weather-btn" type="submit"
                                    class="btn btn-link text-primary">
                                <img id="searchIcon" src="images/search.png" />
                            </button>
                        </div>
                    </div>
                </div>
            </div>
       </div>

       <div id="error-msg" class="not-displayed">
       </div>

       <div id="weather-data" class="not-displayed">
            <div class="d-flex align-items-center flex-column" id="basicData">
                <div class="row my-auto">
                    <div class="col-12" id="icon">
                        <!--ikona vremenskih uvjeta-->
                    </div>

                    <div class="col-12" id="temperature">/°C</div>

                    <div class="col-12 pt-2 pb-1" id="city-name">-</div>

                    <div class="col-12" id="description">-</div>
                </div>
            </div>

            <div id="detailsRow">
                <div class="row">
                    <div class="col-4">
                        <img class="imgIcon" src="images/indicator.png" />
                        <div class="pt-4" id="pressure">- hPa</div>
                    </div>
                    <div class="col-4">
                        <img class="imgIcon" src="images/humidity.png" />
                        <div class="pt-4" id="humidity">- %</div>
                    </div>
                    <div class="col-4">
                        <img class="imgIcon" src="images/wind.png" />
                        <div class="pt-4" id="wind">- km/h</div>
                    </div>
                </div>
            </div>
       </div>
</div>
```

Kod 4-3 index.html - <body>

Počinjemo sa dodavanjem glavne sekcije sa Bootstrap klasom *„container-fluid“* te sekcijama unutar glavne sa klasama *„row“* i *„col“* koje ćemo koristiti kod svakog elementa aplikacije. Ukupno će biti kreirano 3 retka, s time da će prvi redak imati jedan stupac pune širine, drugi redak će imati 4 stupca pune širine, a zadnji redak će imati 3 stupca od kojih svaki zauzima trećinu širine zaslona.

Popis i objašnjenje svih Bootstrap klasa mogu se pronaći u službenoj dokumentaciji na stranici https://getbootstrap.com/docs/4.5/getting-started/introduction/.

```
<div class="container-fluid">
      <div class="row">
            <div class="col">

            </div>
      </div>
</div>
```

Kod 4-4 index.html - sekcije sa klasama „container-fluid“, „row“ i „col“

- `container-fluid`
  - kreira spremnik aplikacije koji zauzima punu širinu zaslona uređaja
- `row`
  - kreira jedan redak u spremniku aplikacije
- `col`
  - kreira jedan stupac u retku

Prema dizajnu aplikacije, vidimo da se pri vrhu nalazi polje za pretraživanje lokacije sa ikonom pretraživanja s desne strane. To ćemo napraviti uz pomoć Bootstrap klasa koje oblikuju formu na način da spajaju polje i gumb u jednu cjelinu. Bootstrap klasama ćemo također urediti izgled polja i gumba, dodati željene margine te boju pozadine.

U prvu sekciju sa klasom *„col“* dodamo Kod 4-5:

```
<div class="bg-white rounded-pill mx-3 my-2">
    <div class="input-group">
        <input type="search" placeholder="Unesite lokaciju..."
               aria-describeby="find-weather-btn"
               class="form-control border-0 "
               id="city-name-input" name="city" required>
        <div class="input-group-append">
            <button id="find-weather-btn" type="submit"
                           class="btn btn-link text-primary">
                <img id="searchIcon" src="images/search.png"/>
            </button>
        </div>
    </div>
</div>
```

Kod 4-5 index.html - polje za unos lokacije sa gumbom za pretraživanje

- `placeholder`
  - Navodi kratki savjet korisniku koji opisuje očekivanu vrijednost polja

- `required`
  - Određuje da se prije slanja obrasca obavezno mora ispuniti polje za unos

- ```
  <button>
     <img src="" />
  </button>
  ```
  - Dodaje se slika koja se ponaša kao gumb
  ```

Nakon što smo dodali obrazac za unos lokacije, dodajemo novu sekciju unutar glavne sa identifikatorom *„error-msg“* koja će služiti za prikazivanje poruke ukoliko dođe do greške kod prikaza podataka. Ovaj element će biti sakriven prilikom učitavanja aplikacije i prikazivati će se samo prema potrebi.

```
<div id="error-msg" class="not-displayed">
</div>
```

Kod 4-6 index.html - sekcija za ispis poruke o pojavi greške

Sada dodajemo sekciju sa identifikatorom *„weather-data“* (Kod 4-7) koja će sadržavati sve podatke o trenutnom vremenu koje želimo da se prikažu korisniku. U toj sekciji kreiramo jednu sekciju sa identifikatorom *„basicData“* u koju grupiramo ikonu koja prikazuje vremenske uvjete, iznos temperature, naziv mjesta i opis vremenskih uvjeta te sekciju sa identifikatorom *„detailsRow“* u kojoj su grupirane ikone i iznosi tlaka, vlažnosti zraka i vjetra. Grupiranje elemenata nam pomaže da kasnije u CSS-u lakše stiliziramo te elemente.

Sekcija *„weather-data“* će također biti sakrivena jer želimo da u trenutku kada se aplikacija učitava piše samo poruka da je određivanje lokacije u tijeku. Uvjete u kojima želimo da se prikazuje poruka o grešci ili da se ispišu vremenski uvjeti odrediti ćemo JavaScript-om.

```
<div id="weather-data" class="not-displayed">
	<div class="d-flex align-items-center flex-column" id="basicData">
		<div class="row my-auto">
			<div class="col-12" id="icon">
				<!--ikona vremenskih uvjeta koja se
				izmjenjuje uz pomoć JavaScript-a
				s obzirom na podatke koji
				se povuku sa poslužitelja -->
			</div>

			<!--trenutna temperatura zraka-->
			<div class="col-12" id="temperature">/°C</div>

			<!--naziv mjesta-->
			<div class="col-12 pt-2 pb-1" id="city-name">-</div>

			<!--opis vremenskih uvjeta-->
			<div class="col-12" id="description">-</div>
		</div>
	</div>

	<div id="detailsRow">
		<div class="row">
			<div class="col-4">
				<img class="imgIcon" src="images/indicator.png"/>
				<div class="pt-4" id="pressure">- hPa</div>
			</div>
			<div class="col-4">
				<img class="imgIcon" src="images/humidity.png"/>
				<div class="pt-4" id="humidity">- %</div>
			</div>
			<div class="col-4">
				<img class="imgIcon" src="images/wind.png"/>
				<div class="pt-4" id="wind">- km/h</div>
			</div>
		</div>
	</div>
</div>
```

Kod 4-7 index.html - sekcija "weather-data"

Sada možemo krenuti i sa dodavanjem stila našoj aplikaciji. Otvorimo dokument index.css, obrišemo postojeći kod i unesemo novi prema primjeru koji prikazuje Kod 4-8.

```
body {
    min-height: 100%;
    font-family: 'Roboto', 'Open Sans', sans-serif;
    color: #fff;
    background: linear-gradient(180deg, #51D5FF 0%, #193A98 100%);
    background-repeat: no-repeat;
    background-attachment: fixed;
}

.container-fluid {
    height: 100vh;
}

.form-control, .btn {
    border-radius: 40px;
    outline: none;
    border: none;
}

.form-control {
    font-size: 18px;
}

    .form-control:focus, .btn:focus {
        box-shadow: none;
    }

#searchIcon {
    width: 24px;
}

#basicData {
    height: 60vh;
    text-align: center;
}

#icon img {
    width: 120px;
}

#temperature {
    margin-top: -10px;
    font-size: 80px;
    font-weight: 400;
}
```

```css
#city-name {
    margin-top: -10px;
    font-size: 24px;
    font-weight: 500;
}

#description {
    font-size: 22px;
    font-weight: 300;
}

.imgIcon {
    width: 32px;
}

#detailsRow {
    margin-top: 40px;
    text-align: center;
    font-size: 18px;
}

.not-displayed {
    display: none;
}

#error-msg {
    text-align: center;
    margin-top: 50%;
    font-size: 18px;
    font-weight: 400;
}

.refresh-btn {
    border: none;
    border-radius: 10px;
    font-size: 16px;
    padding: 4px 10px;
}
```

Kod 4-8 index.css - CSS kod aplikacije

Sljedeći korak je odabir API-ja koji će služiti za dohvat podataka o vremenskim uvjetima. Za potrebe ove aplikacije koristiti će se OpenWeatherMap API koji nudi besplatan dohvat trenutnih vremenskih uvjeta. Da bismo mogli koristiti API u našoj aplikaciji moramo kreirati korisnički račun na https://openweathermap.org/ i zatražiti API ključ kojeg je preporučeno spremiti na neko sigurno mjesto. API ključ je zapravo jedinstvena šifra koju aplikacija mora pozvati da bi se provjerila autentičnost korisnika i time spriječila zlouporaba samog API-ja.

Kako bismo aplikaciji omogućili povezivanje sa uslugama koje nudi OpenWeatherMap, moramo link na kojem se nalazi API ubaciti u *Content Security Policy*, meta oznaku HTML-a koja se koristi za definiranje odobrenih izvora sadržaja koji se smiju učitati u aplikaciji. U <head> sekciji dokumenta index.html u prvom retku nalazi se meta oznaka u koju moramo prije *https://ssl.gstatic.com* ubaciti http://api.openweathermap.org. Kada smo ubacili potreban link, <head> sekcija izgleda ovako:

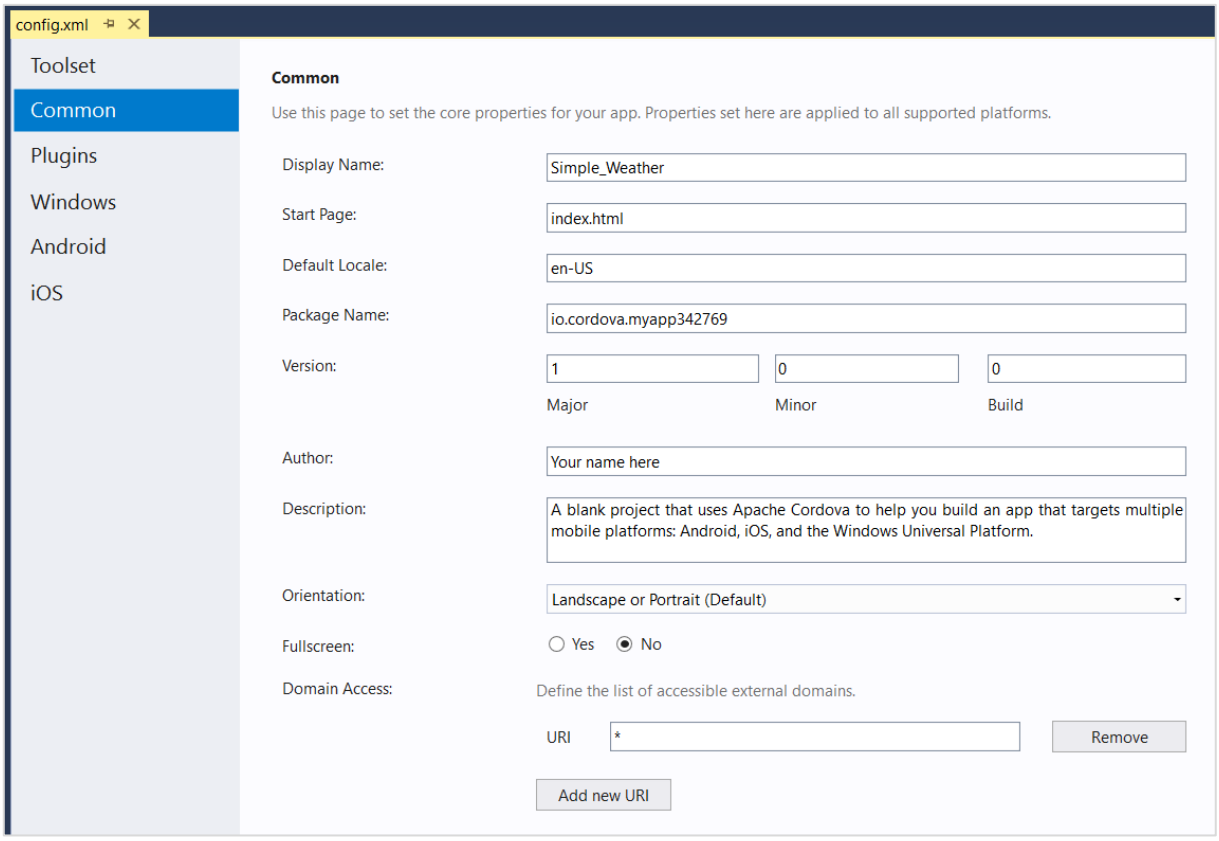
```
<head>
    <meta http-equiv="Content-Security-Policy" content="default-src
'self' data: gap: http://api.openweathermap.org https://ssl.gstatic.com
'unsafe-eval'; style-src 'self' 'unsafe-inline'; media-src *">

    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <meta name="format-detection" content="telephone=no">
    <meta name="msapplication-tap-highlight" content="no">
    <meta name="viewport" content="user-scalable=no, initial-scale=1,
maximum-scale=1, minimum-scale=1, width=device-width">
    <link rel="stylesheet" href="bootstrap/css/bootstrap.min.css" />
    <link rel="stylesheet" type="text/css" href="css/index.css">
    <title>SimpleWeather</title>
</head>
```

Kod 4-9 index.html - <head>

- `<meta>`
  - Oznaka koja definira metapodatke HTML dokumenta; uvijek se nalazi unutar <head> elementa
- `http-equiv`
  - pruža HTTP zaglavlje za informacije/vrijednost atributa sadržaja
- `content`
  - navodi vrijednost povezanu sa http-equiv ili atributom „name“

- `name`
  - specificira ime metapodatka
- `charset`
  - određuje kodiranje znakova za HTML dokument

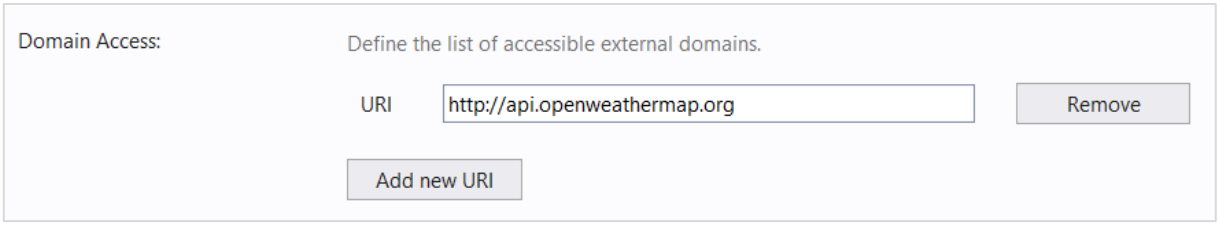
Za uspješno povezivanje sa OpenWeatherMap API-jem još je potrebno definirati pristup vanjskim domenama u Common kartici sučelja dokumenta config.xml.



Slika 4.11 config.xml - Common kartica

Prilikom kreiranja Cordova projekta automatski je dodan dodatak *Whitelist* koji dodaje određeni nivo sigurnosti Cordova aplikaciji dopuštajući programerima da definiraju domene kojima aplikacija može pristupiti. Prema zadanim postavkama, aplikacija može pristupiti bilo kojoj domeni upisivanjem '*' u polje, a to možemo vidjeti kod postavke „Domain Access“ (Slika 4.11). Iako nije greška ako dopustimo aplikaciji pristup bilo kojoj domeni, sigurnosna preporuka je dopustiti pristup samo domenama kojima naša aplikacija mora pristupati za ispravno funkcioniranje.

Kako bismo dodali domenu koja je potrebna za ovu aplikaciju, kliknemo gumb „Remove“ da bismo maknuli pristup svim domenama i dodamo istu domenu koju smo dodali u *Content Security Policy*: http://api.openweathermap.org i spremimo promjene u dokumentu.

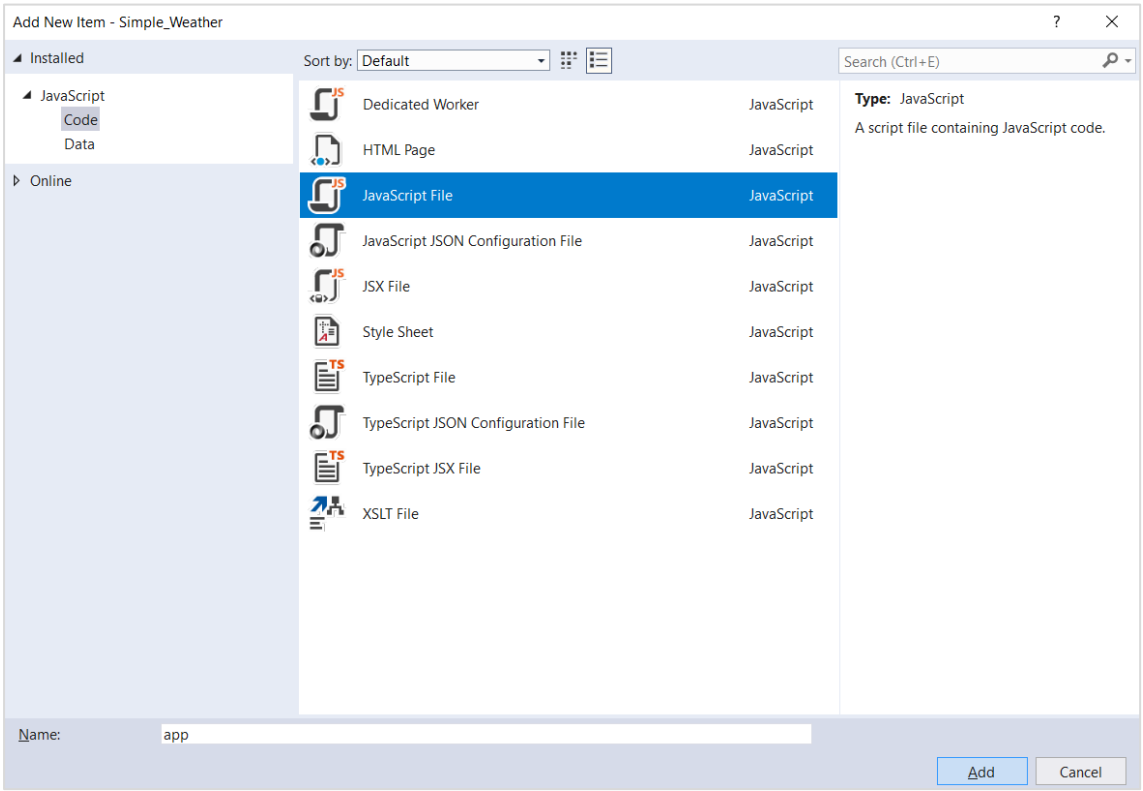


Slika 4.12 config.xml - definiranje pristupa OpenWeatherMap API domeni

4.8.2. JavaScript, jQuery

Kada smo zadovoljni sa izgledom naše aplikacije, krećemo sa dodavanjem JavaScript i jQuery koda kojim ćemo aplikaciji dodati funkcionalnost. Dva glavna dokumenta koja ćemo koristiti su „index.js“ koji već postoji u mapi *www/scripts* te „app.js“ kojeg ćemo kreirati u nastavku.

Novi .js dokument kreiramo na način da desnom tipkom miša kliknemo na mapu *www/scripts* i zatim odaberemo *Add* > *New JavaScript File*. U prozoru koji se otvorio odaberemo tip datoteke „JavaScript File“, upišemo naziv i kliknemo gumb *Add*.



Slika 4.13 kreiranje app.js dokumenta

Sada moramo dodati referencu novokreiranog dokumenta u <body> index.html dokumenta. Na dno ostalih .js referenci dodamo ovaj kod:

```
<script src="scripts/app.js"></script>
```

Popis dodanih .js referenci na kraju izgleda ovako:

```
<script src="scripts/jquery-3.5.1.min.js"></script>
<script src="bootstrap/js/bootstrap.bundle.js"></script>
<script src="bootstrap/js/bootstrap.min.js"></script>
<script type="text/javascript" src="cordova.js"></script>
<script type="text/javascript" src="scripts/platformOverrides.js">
</script>
<script type="text/javascript" src="scripts/index.js"></script>
<script src="scripts/app.js"></script>
```

Kod 4-10 index.html - <script> reference

Index.js

Prvo ćemo urediti kod koji se nalazi u index.js dokumentu. Dio koda koji je namijenjen za zadanu Cordova aplikaciju nam nije potreban i moramo ga maknuti:

```
var parentElement = document.getElementById('deviceready');
        var listeningElement = parentElement.querySelector('.listening');
        var receivedElement = parentElement.querySelector('.received');
        listeningElement.setAttribute('style', 'display:none;');
        receivedElement.setAttribute('style', 'display:block;');
```

Kod 4-11 index.js - dio koda koji moramo obrisati

Umjesto njega ubacimo kodKod 4-12 koji će pozivati funkcije za prikaz vremenskih uvjeta (funkcije ćemo definirati kasnije u dokumentu app.js) s obzirom na to je li korisnik sam upisao lokaciju ili je lokacija automatski određena:

```
getWeatherWithGeoLocation();

$('#find-weather-btn').click(getWeatherWithCityName);

$('#city-name-input').on("keyup", function (event) {
      if (event.keyCode === 13) {
            event.preventDefault();
            $('#find-weather-btn').click(getWeatherWithCityName());
      }
});
```

Kod 4-12 index.js - pozivanje funkcija

- `getWeatherWithGeoLocation()`
  - poziva funkciju getWeatherWithGeoLocation() kada se aplikacija učita

- `$('#find-weather-btn').click(getWeatherWithCityName)`
  - poziva funkciju getWeatherWithCityName svaki put kada kliknemo na gumb sa identifikatorom *„find-weather-btn“*

- ```
  $('#city-name-input').on("keyup", function (event) {
      if (event.keyCode === 13) {
            event.preventDefault();
            $('#find-weather-btn').click(getWeatherWithCityName());
      }
  });
  ```
  - Na pritisak tipke na tipkovnici sa ASCII kodom 13 (*Carriage return;* tipka „Enter“) unutar polja sa identifikatorom *city-name-input* aktivira se gumb sa identifikatorom *find-weather-btn* koji poziva funkciju getWeatherWithCityName()
  - Potvrda unijete lokacije pritiskom Entera na tipkovnici
  ```

U nastavku su opisani ostali dijelovi koda index.js dokumenta:

```
  (function () {
    "use strict";

    document.addEventListener( 'deviceready', onDeviceReady.bind( this ),
    false );

    function onDeviceReady() {
        document.addEventListener('pause', onPause.bind( this ),false);
        document.addEventListener('resume', onResume.bind( this ),false);

        getWeatherWithGeoLocation();

        $('#find-weather-btn').click(getWeatherWithCityName);

        $('#city-name-input').on("keyup", function (event) {
            if (event.keyCode === 13) {
                  event.preventDefault();
                  $('#find-weather-btn').click(getWeatherWithCityName());
            }
        });
    };

    function onPause() {
        // TODO: This application has been suspended. Save application
state here.
    };

    function onResume() {
        // TODO: This application has been reactivated. Restore
application state here.
    };
      } )();
```

Kod 4-13 index.js - cijeli kod

- ```
  (function () {
      //kod
  })();
  ```
  - Ovo je tzv. IIFE (engl. *Immediately Invoked Function Expression*) funkcija koja se pokreće automatski čim se datoteka učita

- ```
  "use strict";
  ```
  - Definira da se JavaScript kod mora izvoditi u „strogom načinu“ kod kojeg nije moguće koristiti nedefinirane varijable
  - „strogi način“ olakšava pisanje sigurnijeg JavaScript koda

- ```
  document.addEventListener( 'deviceready', onDeviceReady.bind( this ), false );
  ```
  - dodaje se slušač događaja (engl. *event listener*) *deviceready* koji signalizira da se Cordova u potpunosti učitala i da možemo na siguran način pozivati Cordova API-je tj. funkcija onDeviceReady može krenuti sa izvođenjem

- ```
  function onDeviceReady() {
     //kod
   };
  ```
  - ova funkcija se koristi za pokretanje bilo kojeg aplikacijskog procesa koji koristi neki od Cordova API-ja

App.js

Sada možemo krenuti sa dodavanjem koda u app.js. Prvi korak je kreiranje nove varijable pri vrhu dokumenta u koju ćemo ubaciti OpenWeatherMap API ključ kojeg smo prethodno zatražili:

```
var ApiKey = "api-kljuc";
```

U nastavku ćemo aplikaciji najprije dodati funkcionalnost kod koje korisnik sam upiše lokaciju za koju želi da se prikažu vremenski uvjeti, a zatim funkcionalnost koja će automatski odrediti lokaciju uređaja.

Ispod kreirane varijable u koju smo pohranili API ključ, kreiramo novu funkciju naziva „getWeatherWithCityName“ koja će dohvatiti lokaciju upisanu u polje sa identifikatorom „*city-name-input*“ i izvući podatke o vremenskim uvjetima lokacije sa OpenWeatherMap API-ja. Na web stranici https://openweathermap.org/current možemo vidjeti primjere API poziva prema nazivu grada, ID grada, geografskim koordinatama i poštanskom broju. Ukoliko dođe do greške pri dohvatu podataka, ispisati će se poruka „Error retrieving data“ sa dodatkom jqXHR (jQuery XMLHttpRequest) statusnog teksta koji će dodatno pojasniti grešku te gumbom za osvježavanje.

```
function getWeatherWithCityName() {
    var city = $('#city-name-input').val();
    var queryString =
        'http://api.openweathermap.org/data/2.5/weather?q=' + city +
'&appid=' + ApiKey + '&units=metric&lang=hr';
    $.getJSON(queryString, function (data) {
        showWeatherData(data);
    }).fail(function (jqXHR) {
        $('#weather-data').hide();
        $('#error-msg').show();
        $('#error-msg').html("<p>Error retrieving data.[1] " +
jqXHR.statusText + "</p>" + "<button class='refresh-btn'
onClick='window.location.reload();'>Osvježi</button>");
    });
    return false;
}
```

Kod 4-14 app.js - funkcija getWeatherWithCityName( )

- `var city = $('#city-name-input').val();`
  - kreiramo novu varijablu „city“ u koju spremamo vrijednost polja sa identifikatorom *„city-name-input“*

- `var queryString = 'http://api.openweathermap.org/data/2.5/weather?q=' + city + '&appid=' + ApiKey + '&units=metric&lang=hr';`
  - kreiramo novu varijablu „queryString“ i u nju spremimo API poziv
  - naziv grada dohvaća se preko varijable „city“
  - API ključ dohvaća se preko varijable „ApiKey“

- `units = metric` – koristi se metrički sustav
- `lang=hr` – koristi se hrvatski jezik

- ```
  $.getJSON(queryString, function (data) {
          showWeatherData(data);
  })
  ```
  - Dohvat JSON podataka sa OpenWeatherMap API-ja i specificiranje funkcije koja će se pokrenuti i prikazati podatke ako je dohvat uspješan
- ```
  .fail(function (jqXHR) {
          //kod
  });
  ```
  - Izvršava se ukoliko dođe do greške kod dohvata podataka sa poslužitelja
  - Naredbe unutar funkcije izvršavaju se prema redu po kojem su napisane

- `$('#weather-data').hide();`
  - Sakriva se sekcija sa identifikatorom *„weather-data“*

- `$('#error-msg').show();`
  - Prikazuje se sekcija sa identifikatorom „error-msg“

- ```
  $('#error-msg').html(" Error retrieving data.[1] " +
  jqXHR.statusText + " " + "<button class='refresh-btn'
  onClick='window.location.reload();'>Osvježi</button>");
  ```
  - Unutar sekcije sa identifikatorom *„error-msg“* dodaje se HTML kod koji ispisuje tekst sa dodatkom jqXHR statusnog teksta i gumbom za osvježavanje
- `return false`
  - sprječava zadano ponašanje preglednika

Sada nam treba funkcija koja će ispuniti našu aplikaciju sa trenutnim vremenskim uvjetima prethodno dohvaćenim sa poslužitelja OpenWeatherMap pa zato ispod prve funkcije napravimo novu funkciju „showWeatherData(data)“ u kojoj ćemo provjeriti ako su podaci uspješno definirani te ako jesu najprije sakrijemo poruku sa greškom ukoliko je prikazana i prikažemo sekciju u kojoj će biti ispisani podaci pomoću određenih parametara, a ako nisu ispisati poruku sa greškom. Popis vremenskih parametra koji se mogu povući iz API odgovora nalaze se na ovome linku https://openweathermap.org/current#parameter.
  ```

Pri ispisu podataka koristiti će se i parametar za dohvat ikone s obzirom na trenutne vremenske uvjete. Ovdje imamo dvije opcije: dohvatiti ikone koje nudi OpenWeatherMap ili koristiti neke druge koje smo ili preuzeli sa weba ili napravili svoje. Ako koristimo npr. ikone preuzete sa weba, u mapi *www* napravimo novu mapu naziva *icons* i u nju spremimo preuzete ikone. Preuzete ikone moramo preimenovati prema tome kako su imenovane na stranici https://openweathermap.org/weather-conditions#Icon-list tj. ako ikona za vedro nebo na OpenWeatherMap-u ima naziv 01d, onda tako moramo preimenovati i našu ikonu za vedro nebo.

```
function showWeatherData(data) {

    if (data.weather != undefined) {

        $('#error-msg').hide();
        $('#weather-data').show();

      //ikone iz mape 'icons'
        $('#icon').html("<img src='icons/" + data.weather[0].icon +
        ".png' alt=''>");

      //OpenWeatherMap ikone
      //$('#icon').html("<img src='http://api.openweathermap.org/img/w/"
      + data.weather[0].icon + ".png' alt=''>");

        var temp = Math.ceil(data.main.temp);
        $('#temperature').text(temp + "°C");

        $('#city-name').text(data.name + ", " + data.sys.country);
        $('#description').text(data.weather[0].description);
        $('#pressure').text(data.main.pressure + " hPa");
        $('#humidity').text(data.main.humidity + " %");

        var windSpeed = data.wind.speed;
        //pretvorba iz m/s u km/h
        $('#wind').text(Math.ceil(windSpeed * 3.6) + " km/h");

    } else {
        $('#weather-data').hide();
        $('#error-msg').show();
        $('#error-msg').html("<p>Error retrieving data.[3] " +
jqXHR.statusText + "</p>" + "<button onClick='window.location.reload();'>
Osvježi </button>");
    }
}
```

Kod 4-15 app.js - funkcija showWeatherData(data)

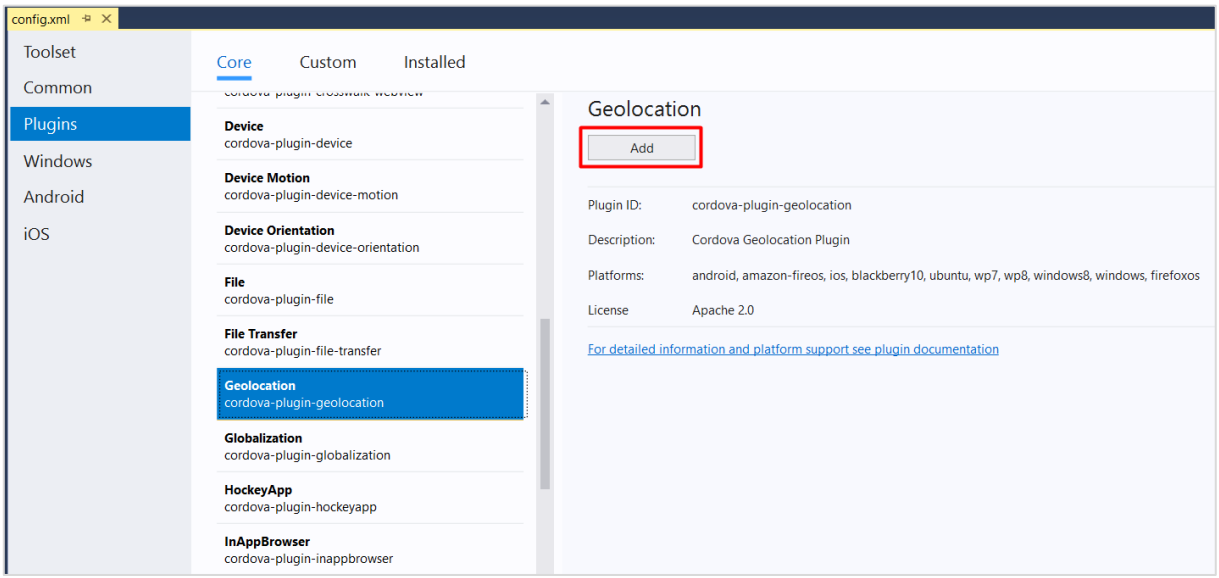
- ```
  if (data.weather != undefined) {
        //kod
  }
  ```
  - provjerava jesu li podaci unutar parametra *weather* definirani
  - `!= undefined` – „različito od nedefiniranog“

- ```
  $('#icon').html("<img src='icons/" + data.weather[0].icon +
  ".png' alt=''>");
  ```
  - u sekciju sa identifikatorom *„icon“* ubacuje sliku iz mape icons koja odgovara trenutnim vremenskim uvjetima

- `Math.ceil`
  - Zaokružuje broj na sljedeći najveći cijeli broj

Dodavanjem dviju opisanih funkcija imamo dodanu funkcionalnost aplikacije kod koje korisnik upiše lokaciju koju želi i ispišu se vremenski uvjeti. Sada slijedi dodavanje funkcionalnosti za automatsko određivanje lokacije. Ova funkcionalnost zahtijeva pristup lokaciji uređaja na kojem je instalirana aplikacija pa iz toga razloga u projekt moramo dodati Cordovin dodatak „Geolocation“.

Otvorimo config.xml uređivač, odaberemo karticu Plugins te iz popisa dodataka odaberemo Geolocation i kliknemo gumb Add.



Slika 4.14 config.xml - dodavanje Cordova dodatka „Geolocation“

Kada smo dodali Geolocation plugin krenemo sa dodavanjem potrebnog koda u app.js dokument. Sav kod koji dodajemo u nastavku mora se nalaziti između funkcije „getWeatherWithCityName“ i funkcije „showWeatherData“.

Prvo dodajemo funkciju naziva „getWeatherWithGeolocation“. Vodeći se službenom Cordova dokumentacijom (https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-geolocation/), trenutnu lokaciju dohvaćamo upisivanjem ovog koda:

```
navigator.geolocation.getCurrentPosition(geolocationSuccess,
                                         [geolocationError],
                                         [geolocationOptions]);
```

Primjećujemo da kod sadrži tri parametra:

- `geolocationSuccess` – funkcija koja će se pozvati ako je trenutna lokacija uspješno dohvaćena
- `geolocationError` - funkcija koja će se pozvati ako se pojavi greška
- `geolocationOptions` – specificiranje dodatnih postavki pri dohvatu lokacije
  - `enableHighAccuracy` – omogućavanje korištenja visoke preciznosti pri određivanju lokacije (koriste se WiFi, mobilne mreže i GPS)
  - `timeout` – maksimalno vrijeme (u milisekundama) koje može proći od poziva prema navigator.geolocation.getCurrentPosition do početka izvođenja geolocationSuccess
  - `maximumAge` – prihvaćanje predmemorirane pozicije čija starost nije veća od navedenog vremena u milisekundama

Navedeni kod ubacujemo u funkciju „getWeatherWithGeoLocation“ kao što je prikazano u primjeru ispod:

```
function getWeatherWithGeoLocation() {
    navigator.geolocation.getCurrentPosition(onGetLocationSuccess,
    onGetLocationError,{ timeout: 8000, enableHighAccuracy: true });

    $('#error-msg').show();
    $('#error-msg').text('Određivanje trenutne lokacije ...');
}
```

Kod 4-16 app.js - funkcija getWeatherWithGeoLocation( )

Sljedeća funkcija koju dodajemo je „onGetLocationSuccess(position)“ prikazana ispod, koja će početi sa izvođenjem ukoliko je trenutna lokacija uspješno dohvaćena.

```
function onGetLocationSuccess(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;

    var queryString =
    'http://api.openweathermap.org/data/2.5/weather?lat=' + latitude +
    '&lon=' + longitude + '&appid=' + ApiKey + '&units=metric&lang=hr';

    $.getJSON(queryString, function (data) {
        showWeatherData(data);
    }).fail(function (jqXHR) {
        $('#error-msg').show();
        $('#error-msg').html("<p>Error retrieving data.[2] " +
jqXHR.statusText + "</p>" + "<button onClick='window.location.reload();'>
Osvježi </button>");
    });
}
```

Kod 4-17 app.js - funkcija "onGetLocationSuccess(position)"

- `var latitude = position.coords.latitude;`
  - kreiramo novu varijablu „latitude“ u koju spremamo vrijednost koordinata geografske širine u decimalnim stupnjevima
- `var longitude = position.coords.longitude;`
  - kreiramo novu varijablu „latitude“ u koju spremamo vrijednost koordinata geografske dužine u decimalnim stupnjevima
- `var queryString = 'http://api.openweathermap.org/data/2.5/weather?lat=' + latitude + '&lon=' + longitude + '&appid=' + ApiKey + '&units=metric&lang=hr';`
  - u varijablu „queryString“ spremamo API poziv koji dohvaća geografsku dužinu i širinu
  - geografska širina dohvaća se preko varijable „latitude“
  - geografska dužina dohvaća se preko varijable „longitude“
  - API ključ dohvaća se preko varijable „ApiKey“

Zadnja funkcija koja nam je potrebna za automatsko određivanje lokacije je „onGetLocationError(error)“ koju prikazuje Kod 4-18, a sa izvođenjem će krenuti ako pri dohvatu lokacije dođe do greške.

```
function onGetLocationError(error) {
    $('#error-msg').html("<p>Nije moguće dohvatiti lokaciju. Provjerite
jeste li povezani na mrežu i uključite lokaciju uređaja.</p>"
       + "<button onClick='window.location.reload();'>Osvježi</button>");
}
```

Kod 4-18 app.js - funkcija "onGetLocationError(error)"

Na kraju u *app.js* dokument još dodamo funkciju koja će osvježiti podatke svaki određeni broj milisekundi. Iako se u ostalim aplikacijama za vremensku prognozu podaci osvježavaju otprilike svakog sata, za potrebe ovog rada stavljeno je na svake 2 minute.

```
setTimeout(function () {
    location.reload();
}, 120000);
```

Kod 4-19 app.js - periodičko osvježavanje podataka

4.9. Google Places API

Iako aplikacija sada radi sve što je na početku zamišljeno, primijetila sam da bi bilo dobro da nam se prilikom upisa lokacije pokazuju prijedlozi s obzirom na prvo slovo koje upišemo u polje za pretraživanje. To se može postići dodavanjem Google Places API-ja koji koristi ID mjesta za jedinstveno identificiranje mjesta te vraća informacije o mjestima koristeći HTTP zahtjeve. Prvi korak koji trebamo napraviti je kreiranje korisničkog računa na Google Cloud Platformi (https://cloud.google.com/), a zatim unutar Platforme kreirati novi projekt. Tek kada smo kreirali novi projekt, možemo zatražiti ključ Places API-ja i omogućiti njegovo korištenje. Više o radu s Google Maps Platformom koja se nalazi u sklopu Google Cloud Platforme i dohvatu API ključeva može se pročitati na https://developers.google.com/maps/gmp-get-started .

Nakon uspješno dohvaćenog Places API ključa, u *index.html* potrebno je najprije u *Content Security Policy*, na isto mjesto gdje je dodan link za OpenWeatherMap, dodati https://maps.googleapis.com i https://maps.gstatic.com. Zatim je potrebno u *config.xml* dokumentu definirati pristup istim domenama na kartici „Common“ u polju „Domain Access“. Kada smo dopustili pristup potrebnim domenama, u <head> sekciju HTML dokumenta dodajemo sljedeći kod, s time da umjesto „API_KLJUC“ upišemo naš Places API ključ koji smo prethodno dohvatili:

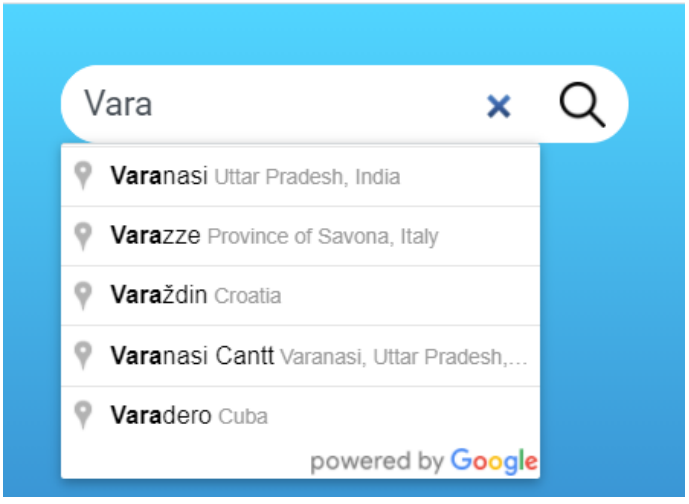
```
<script type="text/javascript"
src=
"https://maps.googleapis.com/maps/api/js?key=API_KLJUC&libraries=places">
</script>
```

Kod 4-20 index.html - učitavanje biblioteke s popisom mjesta

Sada još u <body> sekciju, iznad ostalih <script> referenci dodajemo kod Kod 4-21 kojim ćemo u polje sa identifikatorom *city-name-input* dodati mogućnost samodovršavanja upisa lokacije kao što je prikazano na slici Slika 4.15:

```
<script>
        var input = document.getElementById('city-name-input');
        var autocomplete = new google.maps.places.Autocomplete(input);
</script>
```

Kod 4-21 index.html - dodavanje opcije samodovršavanja upisa lokacije



Slika 4.15 Google Places API- prijedlog lokacije

Time je završen proces kodiranja Cordova aplikacije za vremensku prognozu. Aplikaciju je sada potrebno testirati na što više različitih uređaja i platformi da bismo bili sigurni da ispravno radi, a zatim ćemo ju pakirati da bi bila spremna za objavu u trgovini aplikacija.

4.10. Testiranje aplikacije

Postoje dva različita načina na koje možemo testirati našu aplikaciju: preko simulatora ili preko fizičkog uređaja. U nastavku je objašnjeno testiranje Android aplikacije i Windows aplikacije.

4.10.1. Testiranje aplikacije - Android

Za testiranje aplikacije preko Android simulatora imamo više opcija. Možemo odabrati između više besplatnih ili komercijalnih online simulatora, AVD Manager (Slika 2.13, Slika 2.14) ili direktno preko Visual Studija kojeg smo koristili za izradu aplikacije.

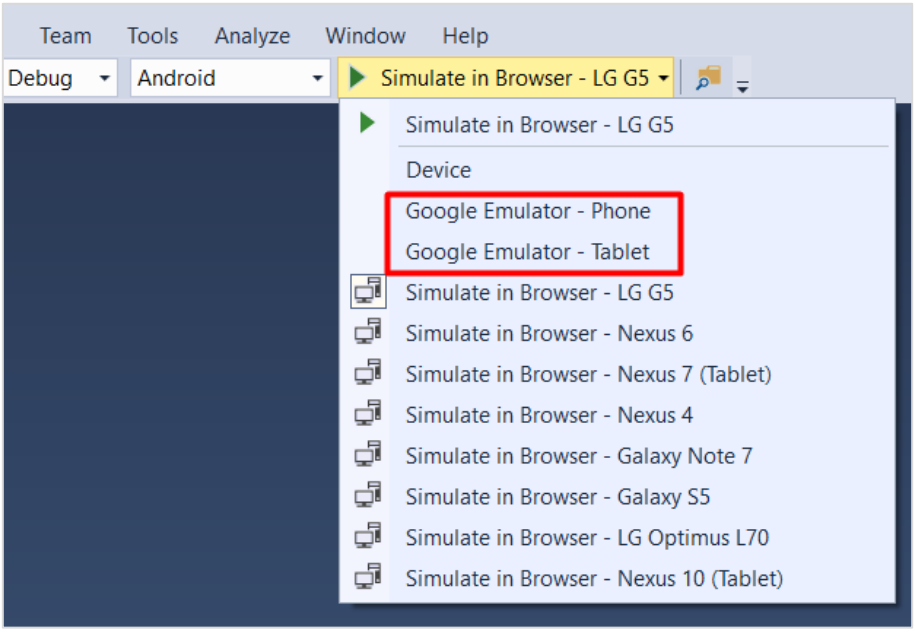
Prije nego krenemo sa testiranjem potrebno je u datoteci „GradleBuilder.js“ koja se nalazi u *platforms/android/cordova/lib/builders/* u sljedećoj liniji koda umjesto „http“ upisati „https“:

```
var distributionUrl
process.env['CORDOVA_ANDROID_GRADLE_DISTRIBUTION_URL']
'http\\://services.gradle.org/distributions/gradle-2.13-all.zip';
```

Ako ne promijenimo protokol u „https“, pojaviti će se greška koja će spriječiti instalaciju aplikacije na simulatoru ili Android uređaju.

Prilikom razvoja aplikacije, najbolje je testirati preko simulacije u pregledniku koju nudi Visual Studio. Na alatnoj traci odaberemo platformu Android te u padajućem izborniku „Simulate in Browser - *<uređaj>*“. Ovaj postupak prikazuje Slika 4.4 u poglavlju Kreiranje novog Apache Cordova projekta. Iako je ovo dobar način za testiranje aplikacije dok je još u procesu razvoja, za testiranje konačnog proizvoda ipak se preporuča koristiti ili simulator koji prikazuje i okvir uređaja ili fizički uređaj.

Prilikom instalacije Apache Cordova alata za Visual Studio, instalirane su i dodatne značajke koje omogućuju pokretanje Android simulatora uređaja. Simulator u Visual Studiju pokrećemo preko alatne trake gdje pokrećemo i simulaciju u pregledniku, ali u padajućem izborniku odaberemo između opcija „Google Emulator - Phone“ i „Google Emulator - Tablet“.

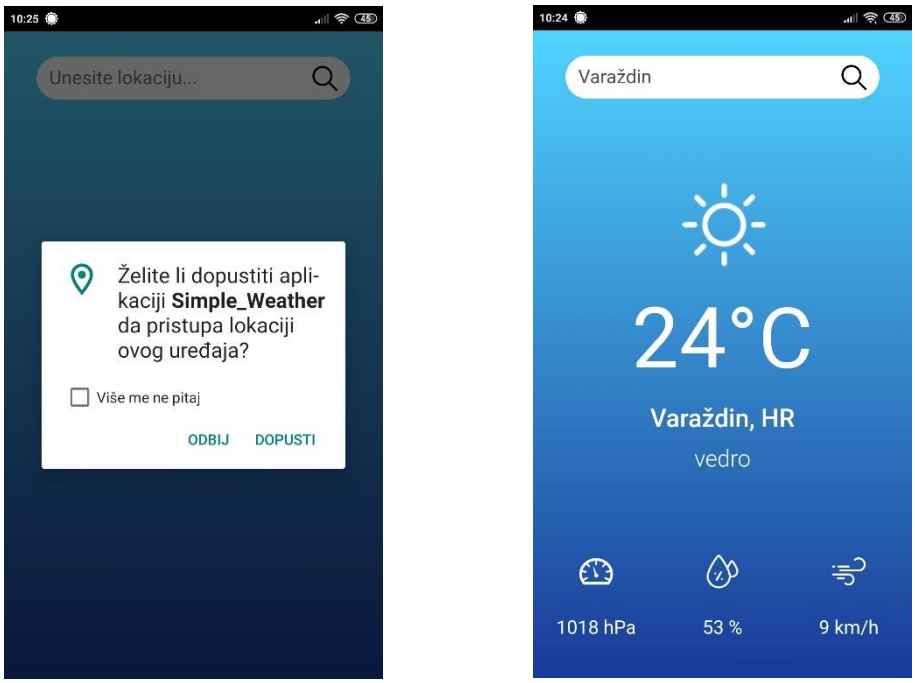


Slika 4.16 Visual Studio - pokretanje Google Emulator-a

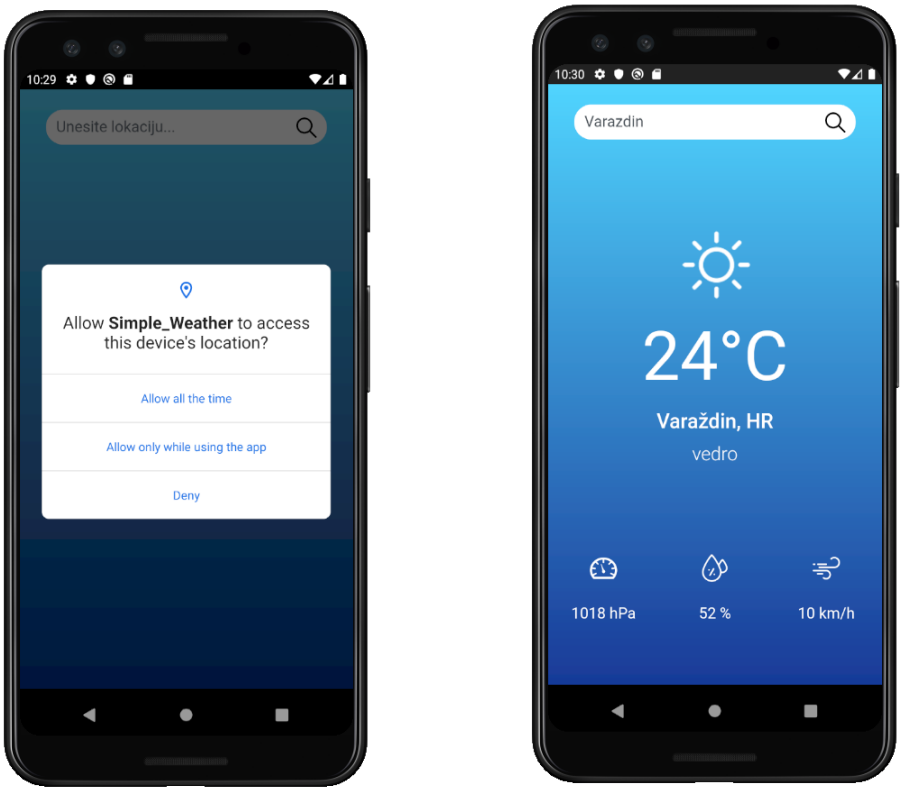
Ako želimo aplikaciju testirati na vlastitom Android uređaju, prvo na uređaju omogućimo opciju „USB Debugging“ i instalaciju preko USB-a, kablom spojimo uređaj sa računalom, a zatim na alatnoj traci Visual Studija odaberemo „Debug“, platformu „Android“ te u padajućem izborniku opciju „Device“ i instaliramo aplikaciju.

Kada želimo testirati preko AVD Managera najprije kreiramo novi virtualni Android uređaj u željenoj verziji Android-a. Zatim pokrenemo simulator i u simulator povučemo datoteku „android-debug.apk“ (*platforms/android/build/outputs/apk/*) koja nastaje prilikom testiranja preko simulatora uključenog u Visual Studio ili preko fizičkog uređaja.

Ova aplikacija je testirana na mobitelu Xiaomi Mi 6 sa verzijom Androida 9 te u AVD Manager-u na simulaciji uređaja Google Pixel 3 na verzijama Androida 6, 7, 8 i 10.



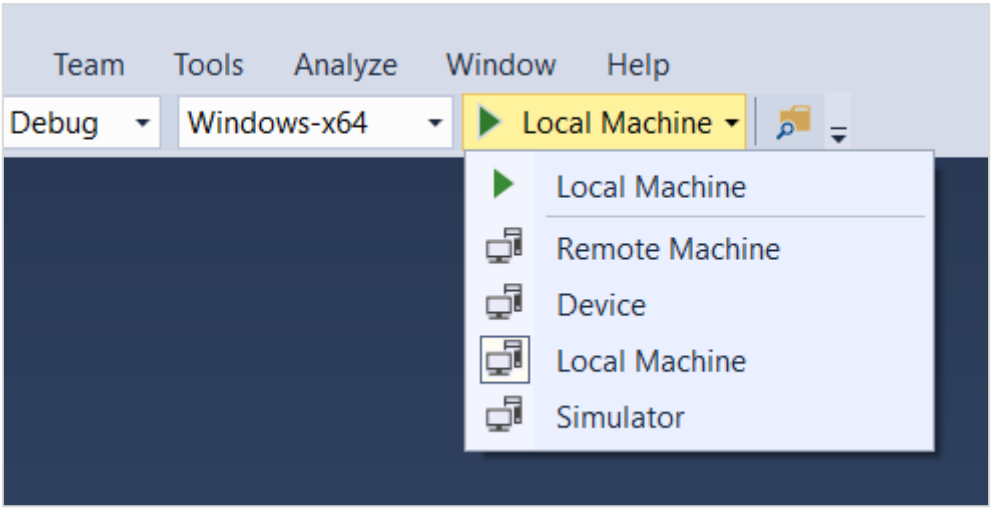
Slika 4.17 Testiranje aplikacije - Xiaomi Mi6, Android 9 (snimka zaslona)



Slika 4.18 Testiranje aplikacije - Google Pixel 3, Android 10

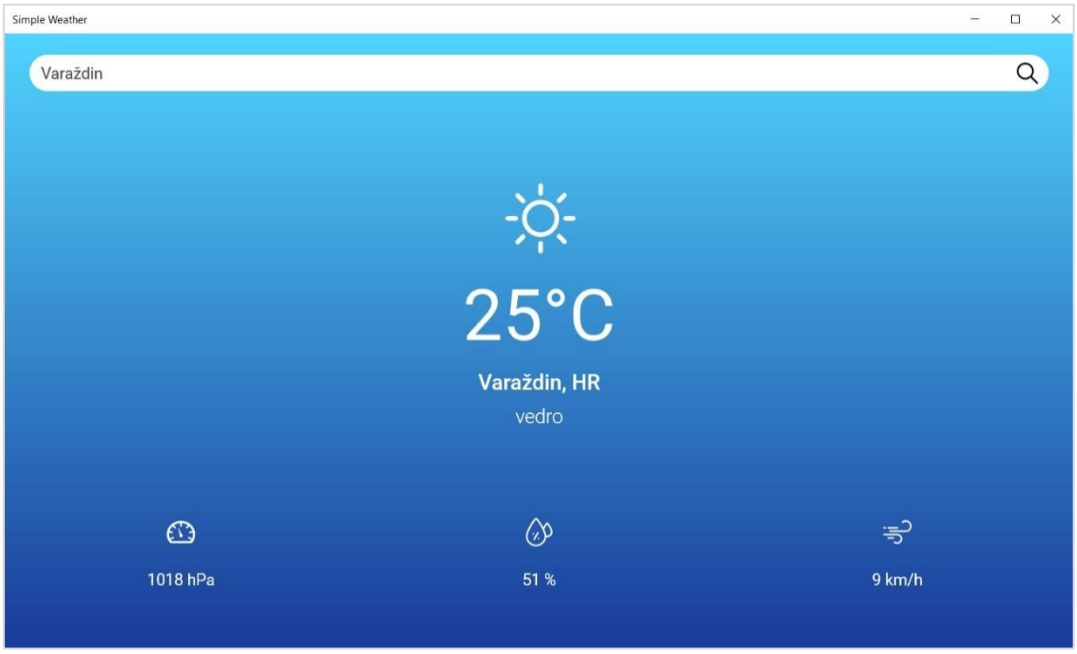
4.10.2. Testiranje aplikacije - Windows

Za testiranje Windows aplikacije moramo u alatnoj traci odabrati platformu Windows te odabrati želimo li aplikaciju instalirati na vlastito računalo (*Local Machine*), simulator, udaljeno računalo (*Remote Machine*) ili Windows uređaj spojen USB kablom (*Device*).



Slika 4.19 Visual Studio - testiranje Windows aplikacije

Aplikacija izrađena u ovome radu testirana je na vlastitom računalu sa Windows 10 operacijskim sustavom:

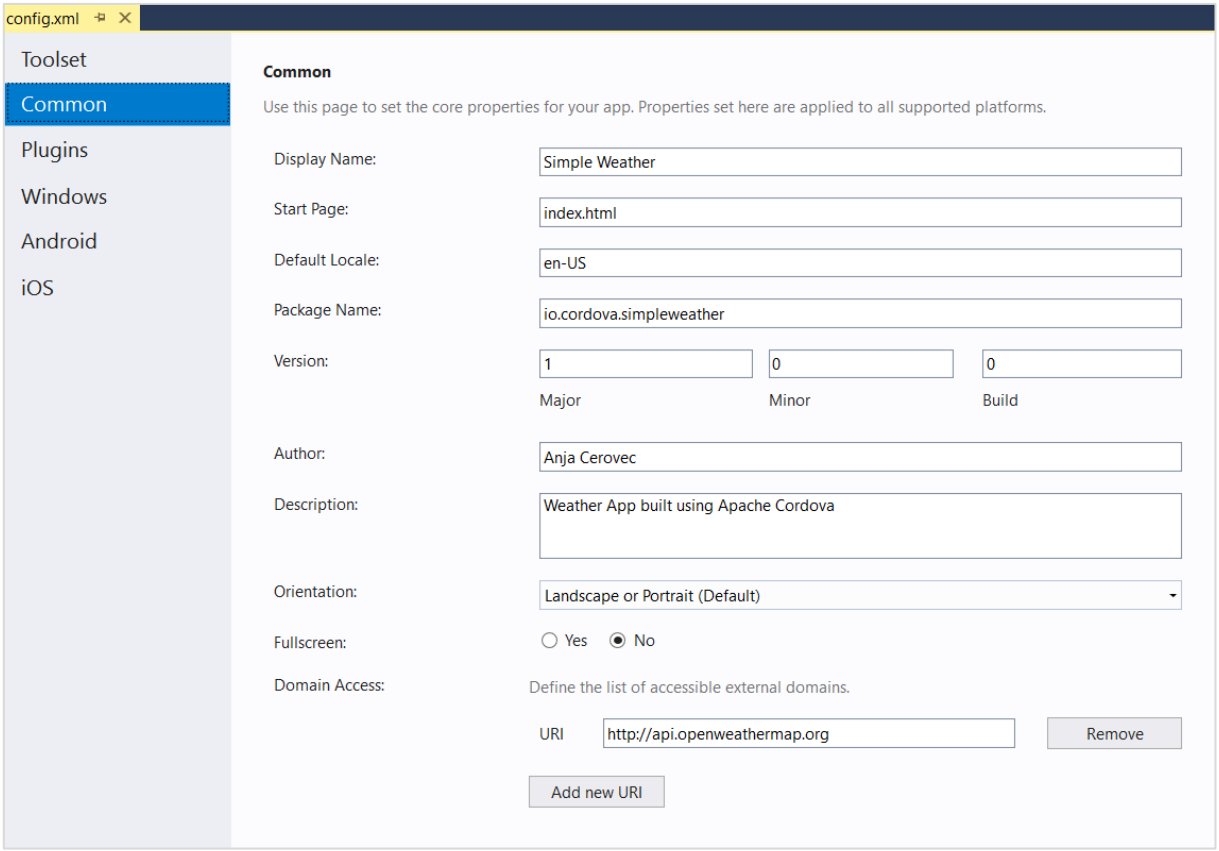


Slika 4.20 Testiranje aplikacije - računalo, Windows 10

4.11. Pakiranje Android aplikacije

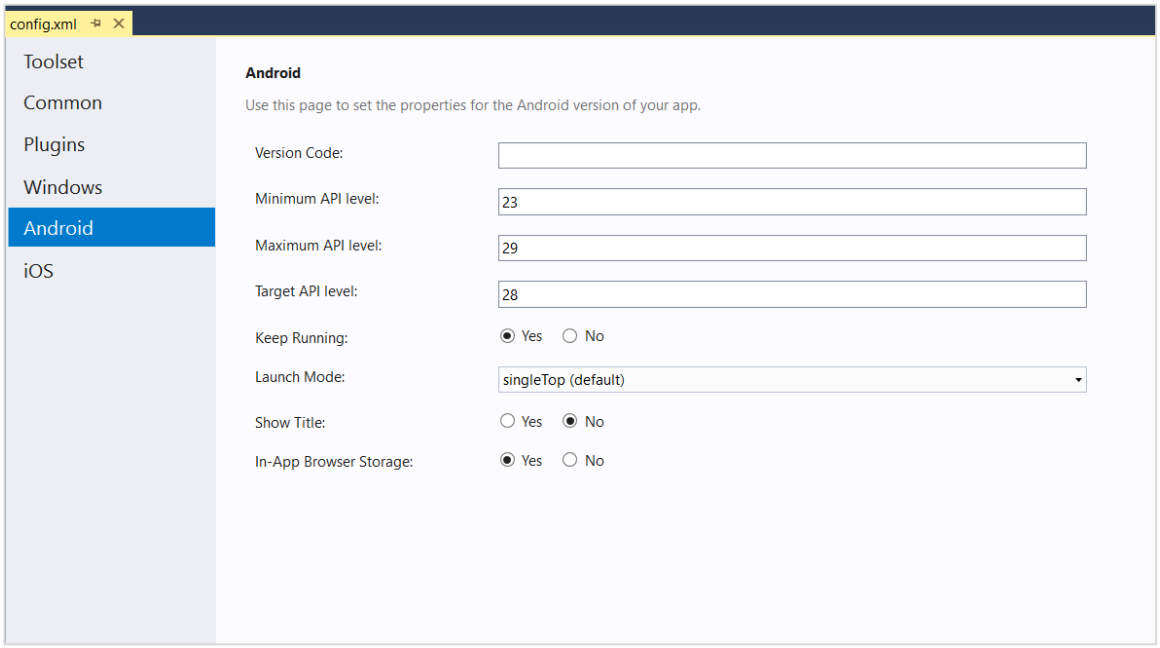
Sada kada je aplikacija uspješno testirana i funkcionira onako kako je zamišljeno, vrijeme je za pakiranje aplikacije kako bi bila spremna za objavu u trgovini aplikacija ili ako ne želimo postavljati aplikaciju u trgovinu, da bismo je mogli poslati drugim osobama koje žele isprobati aplikaciju na svome uređaju.

Prvi korak kod pakiranja Android aplikacije za objavu je popunjavanje potrebnih podataka preko config.xml uređivača. Otvorimo projekt u Visual Studiju i dvaput kliknemo na config.xml datoteku u Solution Exploreru da bismo otvorili uređivač. Odaberemo karticu Common te ispunimo osnovne podatke o aplikaciji prema primjeru na slici ispod:



Slika 4.21 config.xml - ispunjena kartica Common

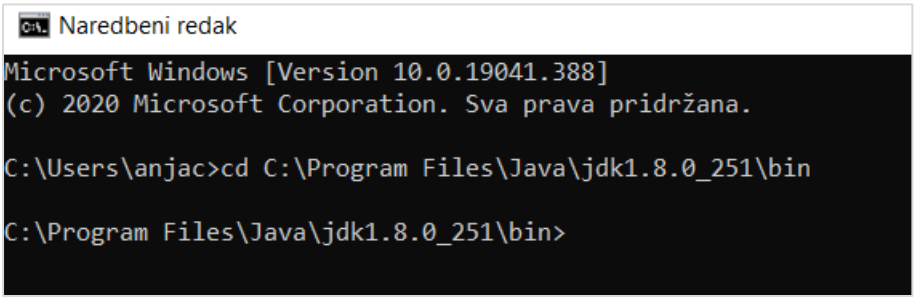
Kada smo ispunili karticu Common, prebacimo se na karticu Android. Postavke pod ovom karticom kontroliraju uvjete pod kojima se aplikacija pokreće na Android uređaju tj. određujemo verziju Androida koju želimo da naša aplikacija cilja, minimalnu verziju i maksimalnu. Minimalna verzija na kojoj je ova aplikacija testirana je API 23, a maksimalna na kojoj je testirano je API 29. Za ciljanu API verziju postavljen je API 28 tj. Android 9. Ovu karticu možemo ostaviti i praznu.



Slika 4.22 config.xml - kartica Android

Sljedeći korak za pakiranje aplikacije je generiranje *keystore-a*. U njega će se pohraniti certifikat kojim ćemo potpisati aplikaciju i time označiti da je aplikacija u našem vlasništvu. Keystore nam je potreban za stavljanje Android aplikacije u Google Play trgovinu jer veže aplikaciju uz jedinstveni ključ. Taj isti ključ koristi se kada želimo objaviti noviju verziju aplikacije pa je zato bitno keystore spremiti na sigurno mjesto i zapamtiti lozinku za pristup. Ako netko dođe do keystore-a i lozinke, ta osoba lako može promijeniti našu aplikaciju te objaviti lažnu aplikaciju. [5]

Za kreiranje keystore-a otvorimo CMD te navigiramo do *bin* mape instaliranog Java JDK koristeći naredbu `cd`:

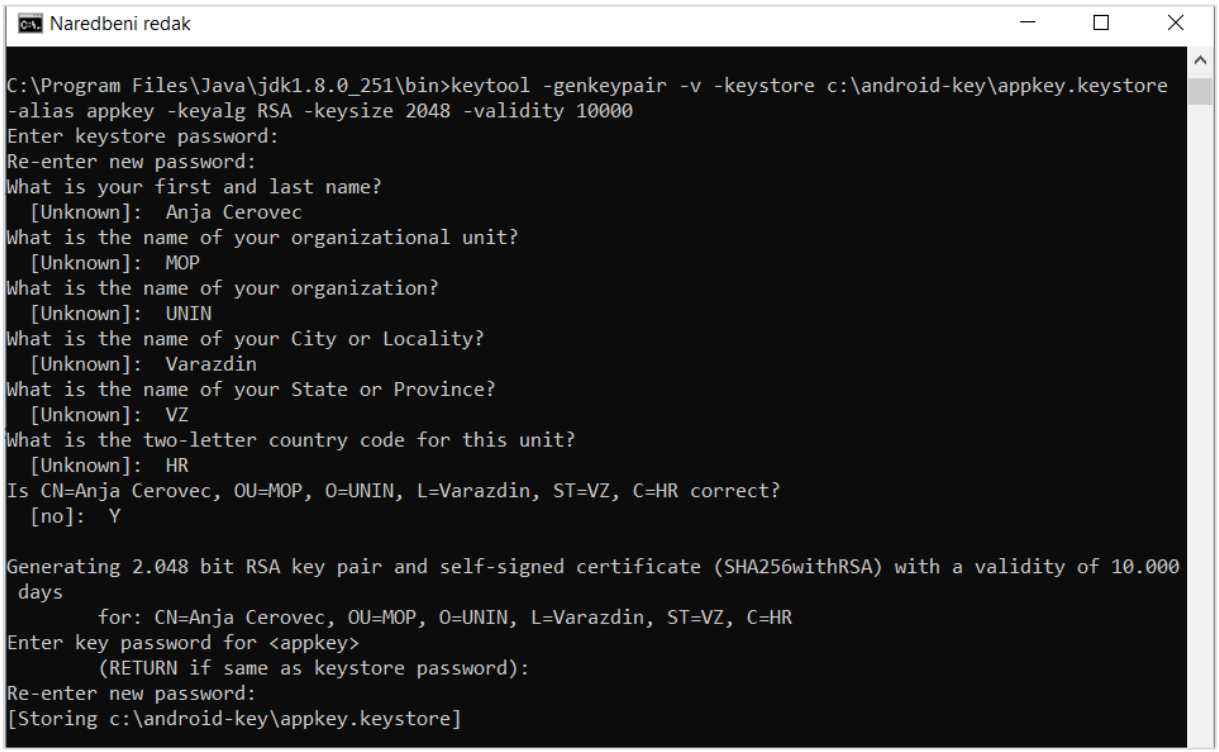


Slika 4.23 CMD - navigiranje do bin mape JDK

Zatim upišemo slijedeću naredbu s time da "`FILE-PATH/MY-RELEASE-KEY`" i „`ALIAS_NAME`" zamijenimo sa odgovarajućim podacima za naše potrebe :

```
keytool -genkeypair -v -keystore FILE-PATH\MY-RELEASE-KEY.keystore
-alias ALIAS_NAME -keyalg RSA -keysize 2048 -validity 10000
```

Nakon što smo unijeli ovu naredbu, pritisnemo tipku „Enter" te će nam se po redu ispisati upiti na koje je potrebno odgovoriti:

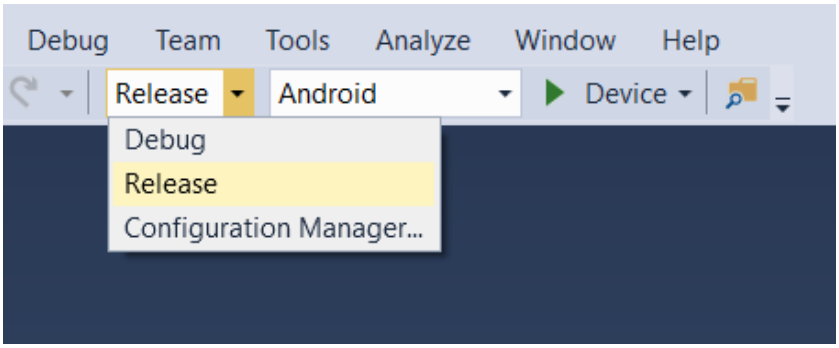


Slika 4.24 CMD - generiranje keystore-a

Kada smo uspješno generirali keystore moramo ga povezati sa našom aplikacijom. Otvorimo projekt u Visual Studiju te u Solution Exploreru pronađemo i otvorimo datoteku *build.json*. Datoteku ispunimo podacima koje smo unijeli tijekom generiranja keystore-a prema primjeru:

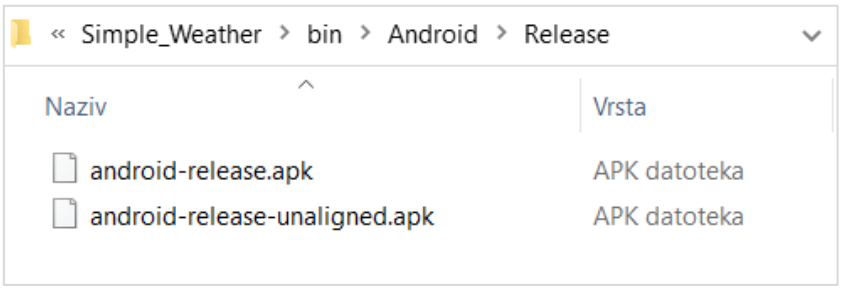
Slika 4.25 Ispunjena build.json datoteka

Zadnji korak za pakiranje aplikacije je izgradnja aplikacije na način da na alatnoj traci Visual Studija umjesto „Debug“ odaberemo „Release“, platformu Android te izgradnju prema fizičkom uređaju i klikom na zelenu strelicu pokrenemo stvaranje paketa.



Slika 4.26 Stvaranje paketa aplikacije

Ovim postupkom kreirana je datoteka *android-release.apk* koju sada možemo objaviti u trgovini aplikacija (detaljnije na: https://support.google.com/googleplay/android-developer/answer/113469?hl=en) ili poslati nekome da isproba našu aplikaciju.



Slika 4.27 Lokacija datoteke android-release.apk

4.12. Usporedba sa ostalim aplikacijama za vremensku prognozu

Aplikacije za vremensku prognozu danas se nalaze na gotovo svakom pametnom telefonu. Većina mobilnih brendova kreira vlastitu aplikaciju koja dolazi za pametnim telefonom, a mnogi korisnici sa trgovine aplikacija preuzmu i dodatnu aplikaciju za vremensku prognozu ako misle da postojeća nema dovoljno opcija ili možda traže i neku jednostavniju sa bržim pregledom trenutnih uvjeta. Postoji nekoliko vrsta aplikacija za vremensku prognozu: komercijalne, one kod kojih trebamo platiti ako želimo maknuti reklame te one koje su u potpunosti besplatne i bez reklama, a te su rijetke.

Aplikacija koja je izrađena u ovome radu pruža korisniku najosnovnije podatke te koristi besplatan API pa ne bi bilo pravedno naplaćivati korištenje takve aplikacije jer se u trgovini aplikacija nalaze konkurentske aplikacije sa puno više opcija, a da su također besplatne. Iako bi bilo dobro u aplikaciju dodati još neke opcije, mislim da i ovako ima svoje prednosti. Zbog prikaza samo osnovnih podataka, brže se učitava i ne zauzima puno memorije uređaja.

Tablica 4-1 prikazuje usporedbu najpopularnijih aplikacija za vremensku prognozu 2020. godine sa aplikacijom iz ovog rada:

	Simple Weather (moja aplikacija)	**AccuWeather**	**The Weather Channel**	**Dark Sky**
Besplatna/ komercijalna	Besplatna	Besplatna	Besplatna	Komercijalna
Hibridna/ nativna	Hibridna	Nativna	Nativna	Nativna
Reklame	Ne	Da	Da	Ne
Trenutni uvjeti	Da	Da	Da	Da
Dugoročna prognoza	Ne	Da	Da	Da
Satelitski prikaz	Ne	Da	Da	Da
Prikaz alergija	Ne	Da	Da	Ne
Widgeti	Ne	Da	Da	Ne
Android/iOS	Android	Android/iOS	Android/iOS	iOS

Tablica 4-1 Usporedba aplikacija za vremensku prognozu

5. Zaključak

Hibridne aplikacije u posljednje vrijeme postaju sve popularnije jer zbog lakoće izrade i manje potrošenog vremena više ljudi ima mogućnost razvoja vlastite aplikacije. Dok je kod nativnih aplikacija potrebno znanje programskih jezika posebno namijenjenih za svaku platformu, pri razvoju hibridne aplikacije uglavnom se koriste HTML, CSS i JavaScript, a rezultat je web aplikacija koja se pokreće kao samostalna aplikacija te pruža osjećaj kao da koristimo nativnu aplikaciju. Uz prednost korištenja standardnih web tehnologija, hibridne aplikacije omogućuju izradu višeplatformske aplikacije tj. aplikacije za koju se koristi isti kod bez obzira na platformu. Iako hibridne aplikacije imaju mnogo prednosti, postoje i neki bitni nedostatci: slabije performanse i brzina kod zahtjevnijih aplikacija te određena ograničenja kod pristupa hardveru uređaja.

U ovome radu, za razvoj hibridne višeplatformske aplikacije korištena je Apache Cordova, besplatan razvojni okvir koji koristi API-jeve za pristup hardverskim značajkama uređaja. Cordova je odabrana iz razloga što postoji široka zajednica korisnika te se na službenoj stranici nalazi opširna dokumentacija o korištenju, koja je pomogla pri uključivanju dodatka *Geolocation* u projekt. Dodatak Geolocation omogućava automatsko određivanje lokacije uređaja, što je jedan od bitnijih dijelova bilo koje današnje aplikacije za vremensku prognozu. Cordova dolazi sa vlastitim sučeljem naredbenog retka – Cordova CLI koje sam prvotno željela koristiti za razvoj aplikacije, ali naišla sam na *bug* (greška u programu koja uzrokuje neželjene rezultate) koji je sprječavao korištenje dodatka Geolocation na Android uređajima. Iz toga razloga odlučila sam koristiti Visual Studio u koji se mogu uključiti Apache Cordova alati koji pomažu pri izradi i testiranju Cordova projekta.

Cilj ovog rada bilo je upoznavanje sa procesom izrade hibridne aplikacije uz pomoć Apache Cordove te korištenje naučenog za izradu vlastite aplikacije za vremensku prognozu. Ovaj rad ukazuje na to da u današnje vrijeme svatko tko ima volje i želje može naučiti izraditi vlastitu mobilnu aplikaciju, bila ona nativna, web ili hibridna. Mislim da je korištenje Apache Cordove dobar početak za izradu vlastite aplikacije za programere koji imaju iskustva sa korištenjem standardnih web tehnologija, ali ne i sa nativnim programskim jezicima za pojedine platforme.

Očekujem da će popularnost hibridnih aplikacija i dalje rasti te da će s vremenom okviri za razvoj višeplatformskih hibridnih aplikacija omogućiti izradu kvalitetnijih i grafički zahtjevnijih aplikacija.

U Varaždinu, _____ _____

Potpis studentice

IZJAVA O AUTORSTVU I SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, ANJA CEROVEC *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom HIBRIDNA MOBILNA APLIKACIJA ZA VREMENSKU PROGNOZU *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

Anja Cerovec
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, ANJA CEROVEC *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom HIBRIDNA MOBILNA APLIKACIJA ZA VREMENSKU PROGNOZU *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

Anja Cerovec
(vlastoručni potpis)

6. Literatura

[1] https://www.knowyourmobile.com/phones/the-history-of-mobile-phones-from-1973-to-2008-the-handsets-that-made-it-all-happen-d58/. [Pokušaj pristupa 15. 06. 2020.].

[2] https://expertise.jetruby.com/brief-history-of-mobile-apps-286fbbf766a9. [Pokušaj pristupa 15. 06. 2020.].

[3] https://www.mobileaction.co/blog/app-business/types-of-apps/. [Pokušaj pristupa 15. 06. 2020.].

[4] https://cordova.apache.org/. [Pokušaj pristupa 15. 06. 2020.].

[5] R. K. Camden, Apache Cordova In Action, Manning Publications, 2016.

[6] J. M. Wargo, Apache Cordova 4 Programming, Addison-Wesley, 2015.

[7] https://blog.phonegap.com/update-for-customers-using-phonegap-and-phonegap-build-cc701c77502c. [Pokušaj pristupa 11. 09. 2020.].

[8] https://en.wikipedia.org/wiki/Npm_(software). [Pokušaj pristupa 01. 07. 2020.].

[9] https://en.wikipedia.org/wiki/Node.js. [Pokušaj pristupa 01. 07. 2020.].

[10] https://ionicframework.com/docs. [Pokušaj pristupa 05. 07. 2020.].

[11] https://www.tutorialspoint.com/ionic/index.htm. [Pokušaj pristupa 05. 07. 2020.].

[12] https://dotnet.microsoft.com/learn/xamarin/what-is-xamarin. [Pokušaj pristupa 07. 08. 2020.].

[13] https://docs.microsoft.com/hr-hr/xamarin/get-started/what-is-xamarin. [Pokušaj pristupa 07. 08. 2020.].

[14] https://flutter.dev/. [Pokušaj pristupa 10. 08. 2020.].

[15] https://medium.com/@concisesoftware/what-is-flutter-here-is-everything-you-should-know-faed3836253f. [Pokušaj pristupa 10. 08. 2020.].

[16] https://en.wikipedia.org/wiki/Appcelerator_Titanium.
[Pokušaj pristupa 09. 07. 2020.].

[17] https://devblog.axway.com/mobile-apps/comparing-titanium-and-phonegap/.
[Pokušaj pristupa 10. 07. 2020.].

[18] https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/first-steps/build-your-first-app?view=toolsforcordova-2017. [Pokušaj pristupa 27. 07. 2020.].

[19] https://en.wikipedia.org/wiki/JQuery. [Pokušaj pristupa 22. 07. 2020.].

Popis slika

Popis kodova

Popis tablica