IZRADA iOS MOBILNE APLIKACIJE ZA VREMENSKU PROGNOZU

Zagorac, Domagoj

Veleučilište u Karlovcu
Strojarski odjel
Stručni prijediplomski studij Mehatronika

Domagoj Zagorac

IZRADA IOS MOBILNE APLIKACIJE ZA VREMENSKU PROGNOZU

ZAVRŠNI RAD

Karlovac, 2023

Karlovac University of Applied Sciences
Mechanical engineering Department
Professional undergraduate study of Mechatronics

Domagoj Zagorac

Development of an iOS mobile application for weather forecast

FINAL PAPER

Karlovac, 2023

Veleučilište u Karlovcu
Strojarski odjel
Stručni prijediplomski studij Mehatronika

Domagoj Zagorac

IZRADA IOS MOBILNE APLIKACIJE ZA VREMENSKU PROGNOZU

ZAVRŠNI RAD

Mentor:
dr. sc. Anamarija Kirin, pred

Karlovac, 2023

ZAVRŠNI ZADATAK

VELEUČILIŠTE U KARLOVCU

KARLOVAC UNIVERSITY OF APPLIED SCIENCES
Trg J.J.Strossmayera 9
HR-47000, Karlovac, Croatia
Tel. +385 - (0)47-843-510
Fax. +385 - (0)47-843-579

HRN EN ISO 9001:2015

VELEUČILIŠTE U KARLOVCU

Stručni/specijalistički studij: Mehatronika
(označiti)

Usmjerenje: / Karlovac,
2023.

ZADATAK ZAVRŠNOG RADA

Student: Domagoj Zagorac Matični broj: 0035211250

Naslov: Izrada iOS mobilne aplikacije za vremensku prognozu

Opis zadatka: Opisati sučelje za programiranje aplikacija (API), grafičko korisničko sučelje (GUI) te protokol dobivanja informacija preko API-ja. Izraditi iOS mobilnu aplikaciju za vremensku prognozu koja će prognozirati vrijeme do 7 dana uz mogućnost odabira grada. Uključiti animaciju trenutnih vremenskih prilika na odabranoj lokaciji.

Zadatak zadan:	Rok predaje rada:	Predviđeni datum obrane:
07/23.	09/23.	09/23.

Mentor: Predsjednik Ispitnog povjerenstva

dr. sc. Anamarija Kirin

PREDGOVOR

Ovaj rad je posvećen mome ocu, Elvisu, koji je imao, a i danas ima enormnu ulogu u mome životu. Ocu koji je uvijek nesebično stavljao mene ispred sebe samoga, koji mi je uvijek davao neizmjernu podršku i ljubav, bodrio me, vjerovao u mene daleko više nego sam ja tada mogao u samoga sebe. Tata, od srca ti hvala na svemu!

Želio bih iskazati iskrenu i duboku zahvalnost mentorici, dr. sc. Anamariji Kirin na vrijednom vodstvu i angažmanu tijekom izrade i realizacije mog završnog rada. Radi Vaše stručnosti, predanosti i strpljenja izvukao sam ono najbolje iz svog rada.

Posebnu bih zahvalu uputio Karli što je velikodušno, požrtvovno, inspirativno i nesebično svakog dana bila uz mene. S tobom je život lakši.

Najdublju zahvalnost bih uputio sestri i majci, ne samo zbog toga što su požrtvovno bile uvijek tu za mene, nego što me neprestano bodre, guraju i navijaju za mene svakoga dana.

Najiskrenije Vam hvala!

Sažetak

Ovaj diplomski rad fokusira se na razvoj iOS mobilne aplikacije za vremensku prognozu koristeći openweathermap.org API. Aplikacija će omogućiti korisnicima da prate vremenske prognoze za različite gradove do 7 dana unaprijed. Implementirat će se grafičko korisničko sučelje (GUI) koje će omogućiti odabir grada i prikaz trenutnih vremenskih prilika putem animacija. Kroz ovaj rad, analizirat će se sučelje za programiranje aplikacija (API), dizajniranje GUI-a te protokol dobivanja informacija putem openweathermap.org API-a.

KLJUČNE RIJEČI; APLIKACIJSKO PROGRAMSKA SUČELJA (API), JSON, GRAFIČKO KORISNIČKO SUČELJE (GUI), HTTP SWIFTUI, OPENWEATHERMAP

Summary

This master's thesis focuses on the development of an iOS mobile application for weather forecasting using the openweathermap.org API. The application will enable users to track weather forecasts for different cities up to 7 days in advance. A graphical user interface (GUI) will be implemented, allowing city selection and display of current weather conditions through animations. Throughout this thesis, the application programming interface (API) will be analyzed, GUI design will be addressed, and the protocol for retrieving information via the openweathermap.org API will be discussed.

KEYWORDS: APPLICATION PROGRAMMING INTERFACES (API), JSON, GRAPHICAL USER INTERFACE (GUI), HTTP, SWIFTUI, OPENWEATHER

SADRŽAJ

Stranica

1. UVOD

Mobilne aplikacije postale su sastavni dio svakodnevnog života, pružajući korisnicima brz pristup informacijama i funkcionalnostima. Vremenska prognoza je jedna od najtraženijih informacija, s potrebom za preciznim i ažuriranim podacima. U ovom završnom radu istražuje se proces izrade iOS mobilne aplikacije za vremensku prognozu. Rad će se usredotočiti na opis sučelja za programiranje aplikacija (API), grafičko korisničko sučelje (GUI) te protokol dobivanja informacija putem iOS aplikacije koja koristi openweathermap.org API za prikaz trenutnih vremenskih podataka i prognoza. *OpenWeatherMap* API omogućava pristup bogatom skupu vremenskih podataka.

U radu će se analizirati dostupni endpointovi, njihova struktura i parametri te način komunikacije s API-jem putem HTTP zahtjeva. Kreiranje korisničkog sučelja bit će izvedeno korištenjem *Swift* programskog jezika i *Xcode* razvojnog okruženja. GUI će omogućiti korisnicima unos željenog grada te će prikazivati trenutne vremenske informacije i prognoze za narednih 7 dana. Poseban naglasak bit će stavljen na intuitivan dizajn i korisničko iskustvo. Aplikacija će koristiti HTTP zahtjeve za komunikaciju s openweathermap.org API-jem. Bit će detaljno opisani koraci slanja zahtjeva, obrada odgovora te parsiranje podataka za daljnje prikazivanje u aplikaciji. Jedna od ključnih značajki aplikacije bit će animacija trenutnih vremenskih prilika na odabranoj lokaciji. Koristeći animacijske tehnike dostupne u Swiftu, prikazivat će se stvarna vremenska situacija u gradu korisnika.

2. SUČELJE ZA PROGRAMIRANJE APLIKACIJA (API)

API (engl. *Application Programming Interface*) omogućava interakciju između aplikacije i vanjskog sustava, omogućujući dohvaćanje i razmjenu podataka. Za ovaj projekt, koristit ćemo vremenski API koji pruža ažurirane vremenske informacije za različite lokacije. API će se koristiti za dobivanje podataka kao što su temperatura, uvjeti vjetra, vlaga, i drugi relevantni parametri. Prije nego što dublje istražimo ulogu vremenskog API-ja, važno je razumjeti osnove API-ja. API je skup definiranih pravila i protokola koji omogućavaju komunikaciju između različitih aplikacija, servisa ili sustava [1]. API-ji djeluju kao posrednici koji omogućavaju aplikacijama da zahtijevaju određene usluge ili podatke od drugih aplikacija ili servisa, čime se omogućuje integracija funkcionalnosti i razmjena podataka bez potrebe za dubokim razumijevanjem unutarnjeg rada drugog sustava.

2.1. Aplikacijsko programsko sučelje (API)

Suvremeno društvo koristi izuzetno brze kanale komunikacije i razmjene podataka zahvaljujući globalnoj povezanosti putem interneta. No, usprkos ogromnom kapacitetu interneta, njegova struktura ne omogućuje uvijek učinkovit pristup podacima zbog njihove razbacanosti i skrivenosti. Strojno čitljivi podaci predstavljaju novi vid resursa koji podiže vrijednost podataka na novu razinu, zahvaljujući njihovoj strukturiranoj prirodi koja olakšava njihovu računalnu obradu i analizu.

U kontekstu razmjene podataka, a posebno onih koji su čitljivi kako za strojeve tako i za ljude, ključnu ulogu igraju formati kao što su JSON (engl. *JavaScript Object Notation)* i XML (engl. *Extensible Markup Language*). Ovi formati omogućuju elegantnu i strogo definiranu strukturu podataka, što čini proces razmjene podataka efikasnijim i preciznijim. Ovakvi formati su iznimno napredni i dovoljno prilagodljivi da se mogu implementirati kroz poslužitelje u aplikacijskim sučeljima kojima se pristupa putem programskog koda.

API omogućava korisnicima programski pristup podacima putem datoteka koje poslužitelj stavlja na raspolaganje preko interneta. Međutim, API je više od samog sučelja ili pristupa poslužitelju - to je skup resursa, metoda i dostupnih podataka koji omogućavaju interakciju s određenim poslužiteljem na programski način. Osim toga, API posjeduje ključna svojstva kao što su pristupačnost, jednostavnost i kompatibilnost s različitim programskim pristupima, te često pruža mogućnost personaliziranog prikaza podataka. Datoteke koje se prenose putem API-ja obično su strukturirane i strojno čitljive te predstavljaju sadržaj koji je dostupan na web sjedištu, no programski se generira ili oblikuje na strani poslužitelja. API sučelje se također sastoji od funkcija i procedura koje omogućuju korisnicima pristup podacima nižeg nivoa, fokusiranim na odabranu temu koju sami definiraju.

Dosta web aplikacija i internetskih stranica u svijetu koristi aplikacijska programska sučelja, bilo da su korisnici svjesni tog sučelja ili ne [2]. Bez obzira na veličinu, od velikih korporacija do malih obiteljskih poduzeća, vjerojatno koriste neki oblik API sučelja kako bi svojim korisnicima omogućili brz i jednostavan pristup informacijama koje im trebaju.

Stručnjaci prepoznaju važnost računalne tehnologije i interneta kao ključnog kanala za razmjenu podataka [3]. API sučelja su široko rasprostranjena, čak i u znanstvenoj zajednici. Primjerice, postoje besplatna API sučelja koja pružaju raznovrsne strukturirane podatke, poput botaničkih informacija o vrstama biljaka ili temeljnih astronomskih informacija. Također, postoje API sučelja koja pružaju informacije dobivene iz naprednih algoritama strojnog učenja za obradu slika, omogućujući prepoznavanje sadržaja slika. Većina API sučelja dolazi od organizacija koje su stručne u istim područjima kao i pruženi podaci, poput otvorenih vladinih organizacija ili specijaliziranih privatnih tvrtki. Posebno u znanstvenom svijetu, gdje se često suočavamo s ogromnom količinom podataka, potrebno je učinkovito rješenje za obradu i analizu tih podataka. Za programski pristup podacima, kako bi se olakšala obrada i pretvorba podataka u željeni izlazni format, korištenje API sučelja je izvrsna opcija.

2.2. Formati za razmjenu informacija putem API sučelja JSON

Formati za razmjenu informacija putem API sučelja igraju ključnu ulogu u olakšavanju komunikacije i razmjene podataka između aplikacija i poslužitelja. Iako postoji široka paleta formata koji se koriste za ovu svrhu, naglasak će biti na najčešće korištenim formatima - XML i JSON, uz YAML kao čitljiviju alternativu JSON-u, te CSV kao format za razmjenu tabličnih podataka. Nama za rad bitan JSON (engl. *JavaScript Object Notation*) je popularan, strojno čitljiv format podataka koji je jednostavan za čitanje i pisanje kako za ljude, tako i za računalne sustave [4]. Podaci u JSON formatu organizirani su u hijerarhijsku strukturu koristeći ključ-vrijednost parove. Ovi podaci su često formatirani između vitičastih zagrada, a indentacija (tabovi) se često koristi kako bi se olakšala čitljivost ljudima. JSON je prilagodljiv različitim programskim strukturama, što ga čini valjanim na mnogim platformama, uključujući JavaScript i Python [3].

Njegova jednostavnost, čitljivost i brzina ručnog pisanja su čimbenici koji ga čine izuzetno popularnim za razmjenu podataka. JSON je standardiziran prema ECMA-404 standardu, proizašavši iz temeljnog ECMA-262 standarda objavljenog 1999. godine. On je neovisan o specifičnom programskom jeziku, no temelji se na *ECMAScript* programskom jeziku. ECMA-404 definira sintaksu JSON-a, detaljno opisuje njegovu strukturu, upotrebu i svrhu, te precizira oblik JSON objekata i njihovih vrijednosti. Formati kao što su XML, JSON i YAML igraju ključnu ulogu u omogućavanju strukturirane razmjene podataka između aplikacija i poslužitelja putem API sučelja [5]. Njihova prilagodljivost i čitljivost čine ih izvrsnim izborom za različite scenarije i platforme, doprinoseći efikasnoj komunikaciji i razmjeni podataka u modernom digitalnom okruženju.

2.3. Ključna uloga API-ja u osiguravanju ažuriranih vremenskih informacija za aplikacije

Istražit ćemo važnost API-ja koristeći vremenski API kao primjer, fokusirajući se na načine na koje API pruža ažurirane vremenske informacije za različite lokacije te kako te informacije obogate funkcionalnost aplikacija [5].

Vremenski API, kao primjer specifičnog tipa API-ja, omogućava aplikacijama pristup ažuriranim vremenskim informacijama za različite geografske lokacije. Ovaj API pruža korisnicima mogućnost da dobiju informacije kao što su temperatura, uvjeti vjetra, vlaga, prognoze, i drugi relevantni parametri. Ova vrsta informacija izuzetno je korisna za različite industrije i aplikacije, uključujući vremenske aplikacije, web stranice, aplikacije za planiranje putovanja, poljoprivredne sustave i još mnogo toga. Vremenski API pruža niz prednosti koje obogate iskustvo korisnika i optimiziraju funkcionalnosti aplikacija koje koriste ove podatke.

Jedna od glavnih prednosti vremenskog API-ja je pružanje ažuriranih i preciznih vremenskih informacija. Kroz ovaj API, aplikacije dobivaju pristup trenutačnim meteorološkim podacima koji su od ključne važnosti za pružanje relevantnih informacija korisnicima u stvarnom vremenu. Ovo je posebno korisno u situacijama kada korisnici trebaju znati trenutačne vremenske uvjete kako bi donijeli odluke ili se pripremili za određene aktivnosti. Personalizacija je još jedna značajna prednost koju vremenski API pruža.

Na temelju trenutačnih vremenskih uvjeta, aplikacije mogu pružiti korisnicima relevantne savjete, preporuke ili obavijesti koje su prilagođene njihovim potrebama i preferencijama. Osim toga, vremenski API ima ključnu ulogu u procesima odlučivanja, posebno za poslovne aplikacije kao što su poljoprivredni sustavi [6]. Pravodobni vremenski podaci omogućuju bolje planiranje i donošenje odluka. Primjerice, poljoprivrednici mogu koristiti vremenske prognoze kako bi bolje organizirali aktivnosti kao što su zalijevanje ili sjetva te time povećali učinkovitost svojih operacija.

Integracija s drugim uslugama također je važna prednost koju vremenski API pruža. Aplikacije imaju mogućnost povezivanja s različitim uslugama i izvorima podataka. Na primjer, aplikacija za putovanja može iskoristiti vremenske podatke kako bi korisnicima prikazala trenutačne uvjete na njihovim odredištima, poboljšavajući njihovo iskustvo i planiranje.

3. GRAFIČKO KORISNIČKO SUČELJE (GUI)

Grafičko korisničko sučelje ima ključnu ulogu u modernim aplikacijama, omogućujući korisnicima da intuitivno i učinkovito integriraju s kompleksnim informacijama i funkcionalnostima. Primjena GUI-a u aplikaciji za prikaz vremenske prognoze pruža korisnicima brz, vizualan i personaliziran pristup vremenskim informacijama, poboljšavajući njihovo korisničko iskustvo i olakšavajući donošenje informiranih odluka. Kroz primjere kao što je ovaj, vidimo kako grafička korisnička sučelja igraju ključnu ulogu u povezivanju tehnologije s korisnicima na način koji je pristupačan i koristan.

Grafičko korisničko sučelje je ono što korisnik vidi i način na koji koristi aplikaciju. U našem slučaju, GUI će omogućiti korisnicima odabir grada za koji žele vidjeti vremensku prognozu. GUI će također prikazati ažurirane informacije o vremenskim uvjetima, uključujući temperaturu, ikonu vremena i druge relevantne podatke. Za bolje korisničko iskustvo, može se implementirati i mogućnost odabira razdoblja prognoze do 7 dana.

3.1. Općenito o grafičkom korisničkom sučelju

Razvoj digitalne tehnologije bitno je transformirao način interakcije između ljudi i računala. U suvremenom okruženju, komunikacija s računalnim sustavima ostvaruje se putem korisničkih sučelja. Grafička korisnička sučelja, poznata kao GUI (engl. *Graphical User Interface*), prepoznatljiva su forma ovih sučelja. Ona su prilagođena različitim računalnim uređajima i sposobnostima prikaza [7]. Temeljna funkcionalnost GUI-a leži u integraciji teksta, vizualnih elemenata, interaktivnosti i stilizacije kako bi se korisnicima pružio pregledan prikaz informacija.

Povijest grafičkih korisničkih sučelja počinje revolucionarno 1981. godine kada je tvrtka Xerox razvila sučelje „Star interface“ za osobna računala. Ovo sučelje je preuzelo konceptualni model uredskih aktivnosti, omogućujući korisnicima da simuliraju radnje ureda [8]. Ova digitalizacija uključivala je manipulaciju

pokazivačem miša umjesto premještanja papira, pristup printera, organizaciju dokumenata i slične radnje. Ova transformacija omogućila je korisnicima obavljanje tih aktivnosti putem digitalnih uređaja, stvarajući novo iskustvo. WIMP (*windows*, *icons*, *menus*, *pointer*) koncept, preteča modernih grafičkih korisničkih sučelja, temelji se na sučeljima s kvadratnim prozorima, kliznim trakama, panelima i dijaloškim okvirima. Iako je ovakav pristup bio ograničavajući za programere, omogućio je manje iskusnim korisnicima lakše upravljanje računalom. Sučelja su se tijekom vremena prilagodila mobilnim uređajima i dodirnim zaslonima, gdje je osnovni način interakcije postao dodir. Iako su se neka svojstva promijenila, osnovne pretpostavke WIMP pristupa zadržane su s određenim modifikacijama [9].

Grafičko korisničko sučelje ima sposobnost prikaza informacija kroz različite vizualne elemente. Vrednovanje vizualnog dojma u GUI-u često se temelji na intuiciji, osjećajima i svijesti, aspektima koje računala još uvijek ne mogu potpuno shvatiti. Manualno testiranje ima ključnu važnost u osiguravanju kvalitete grafičkih korisničkih sučelja jer čovjek može dodijeliti osvrt temeljen na emocionalnom dojmu nakon obavljenih aktivnosti. Raznolikost elemenata u GUI-u omogućava različite pristupe testiranju kako bi se osigurala kvaliteta. Iako se GUI temelji na vizualnom dojmu, mnoge komponente mogu učinkovito biti testirane putem automatiziranih alata.

Važno je napomenuti da računala ne mogu ocijeniti atraktivnost izgleda pojedinih elemenata GUI-a, ali automatizirani alati mogu biti iskorišteni za testiranje funkcionalnosti tih komponenata. Automatizacija testiranja GUI komponenata obično uključuje simulaciju različitih aktivnosti poput dodira, miša ili tipkovnice. Ova metoda omogućuje brzu reprodukciju velikog broja scenarija testiranja u kratkom vremenskom periodu. Osiguranje kvalitete grafičkih korisničkih sučelja može se najbolje postići kombinacijom manualnog i automatiziranog testiranja.

Manualno testiranje se koristi za određivanje atraktivnosti, dojma i korisničkog iskustva svih elemenata GUI-a, dok se automatizirano testiranje fokusira na provjeru temeljnih funkcionalnosti i performansi sučelja na efikasniji način [10]. Ključno je usmjeriti pažnju na integraciju automatskog testiranja u manualni

proces testiranja, kako bi se postigla potpuna pokrivenost i osigurala kvalitetna validacija vizualnih elemenata i programske logike cijelog sustava.

3.2. Primjena GUI-a u aplikaciji za vremensku prognozu

Aplikacija za vremensku prognozu pruža idealno okruženje za primjenu grafičkog korisničkog sučelja (GUI) kako bi korisnicima omogućila jednostavan i interaktivan način pristupa vremenskim informacijama. Kroz GUI, korisnici će moći interaktivno odabrati željeni grad za koji žele vidjeti vremensku prognozu. To se može postići kroz padajući izbornik ili pretraživač, čime će se olakšati korisnicima pronalaženje željenog grada. Nakon što korisnik odabere grad, GUI će prikazati ažurirane informacije o vremenskim uvjetima za taj grad. Podaci kao što su temperatura, vlažnost zraka, brzina vjetra, stvarna i osjećana temperatura te drugi relevantni parametri bit će prikazani na korisniku razumljiv način.

Korištenjem vizualnih ikona, GUI će korisnicima omogućiti brz i lak način za prepoznavanje različitih vremenskih uvjeta. Ikone kao što su sunce, oblaci, kiša, snijeg i drugi brzo će prenijeti informacije o trenutačnom vremenu. Dodatna funkcionalnost GUI-a bit će povezana s prikazom prognoze za odabrani grad. Korisnici će moći odabrati razdoblje prognoze, uključujući opciju prikaza vremenske prognoze do 7 dana unaprijed. GUI će također uključivati interaktivne elemente koji će omogućiti korisnicima da mijenjaju prikaz podataka, zumiraju karte, pretražuju dane prognoze te na taj način prilagode pregled informacija prema vlastitim potrebama.

Personalizacija GUI-a omogućava korisnicima prilagodbu prikaza vremenskih podataka prema svojim željama, što dodatno poboljšava njihovo korisničko iskustvo. Kvalitetan GUI pomaže korisnicima da brže donose informirane odluke o planiranju svojih aktivnosti na temelju vremenske prognoze, čime se povećava korisničko zadovoljstvo i funkcionalnost aplikacije.

4. PROTOKOL DOBIVANJA INFORMACIJA PREKO API-JA

Za dohvaćanje informacija putem API-ja, aplikacija će koristiti HTTP zahtjeve. Kroz API ključ (engl. *API key*), aplikacija će se autentificirati kako bi dobila pristup vremenskim podacima. Zahtjev će sadržavati informacije o odabranoj lokaciji (gradu), a odgovor će biti u JSON formatu. Nakon obrade odgovora, podaci će biti prikazani korisniku putem GUI-a.

4.1. HTTP zahtjevi i komunikacija putem API-ja

HTTP (engl. *Hypertext Transfer Protocol*) je osnovni protokol koji omogućuje komunikaciju između klijenta (najčešće web preglednika) i servera na *World Wide Webu*. To je protokol za prijenos hipertekstualnih dokumenata, kao što su HTML stranice, ali se također koristi za razmjenu različitih vrsta podataka putem API-ja. HTTP zahtjevi se koriste kako bi klijenti (aplikacije, preglednici) tražili od servera da izvrši neku radnju ili da im dostavi određene informacije. Najčešće korišteni HTTP zahtjevi su [10]:

- *GET* zahtjev se koristi za dohvaćanje informacija sa servera. Klijent traži od servera određene resurse ili podatke. Na primjer, kad unesete URL u svoj preglednik, preglednik šalje *GET* zahtjev kako bi dobio HTML stranicu.
- *POST* zahtjev se koristi za slanje podataka serveru. Klijent šalje podatke koje želi poslati na server, kao što su formulari ili korisnički unosi. Ovaj zahtjev često koristi aplikacija za slanje novih podataka na server, npr. prilikom kreiranja novog korisničkog računa.
- *PUT* zahtjev se koristi za ažuriranje postojećih resursa na serveru. Klijent šalje nove informacije koje trebaju zamijeniti postojeće podatke na serveru.
- *DELETE* zahtjev se koristi za brisanje resursa na serveru. Klijent šalje ovaj zahtjev kako bi zatražio brisanje određenog resursa.

- *PATCH* zahtjev se koristi za parcijalno ažuriranje resursa. Umjesto zamjene cijelog resursa, kao kod *PUT* zahtjeva, *PATCH* omogućuje ažuriranje samo određenih dijelova resursa.

HTTP zahtjevi se sastoje od različitih dijelova, kao što su URL (engl. *Uniform Resource Locator*), zaglavlja (engl. *headers*) i tijelo (engl. *body*). URL definira adresu resursa koji se traži ili manipulira. Zaglavlja sadrže dodatne informacije o zahtjevu, kao što su informacije o korisniku, format podataka koji se šalje itd. Tijelo sadrži stvarne podatke koje klijent šalje serveru, kao npr. podaci forme u POST zahtjevu.

Kako bi aplikacija komunicirala s API-jem putem HTTP-a, ona šalje odgovarajuće HTTP zahtjeve serveru i čeka na odgovor. Odgovor servera sadrži statusni kod koji govori o uspješnosti zahtjeva (npr. *200 OK* za uspješan zahtjev, *404 Not Found* za nepostojeći resurs) te odgovarajuće podatke ili informacije koje su tražene ili odgovarajuće greške u slučaju problema [11]. HTTP je protokol bez stanja, što znači da svaki zahtjev serveru dolazi neovisno o prethodnim zahtjevima. To znači da server ne pohranjuje podatke o prethodnim zahtjevima klijenta između različitih zahtjeva, osim ako se to ne postigne korištenjem dodatnih tehnika poput kolačića (engl. *cookies*) ili sesija.

API-ji omogućuju razmjenu podataka i funkcioniranje aplikacija na način da jedna aplikacija može koristiti usluge i funkcionalnosti koje pruža druga aplikacija ili servis. Za komunikaciju s API-jem putem HTTP-a, aplikacija prvo mora znati URL API-ja kojim se želi komunicirati. Također, aplikacija mora biti svjesna tipova zahtjeva koje API podržava (*GET, POST, PUT, DELETE, PATCH*) te kako formatirati podatke u tijelu zahtjeva ako je potrebno [12].

Uz URL, aplikacija također šalje odgovarajuće zaglavlje u zahtjevu. Zaglavlje može sadržavati informacije o autentifikaciji (kako bi se potvrdilo da je klijent ovlašten pristupiti API-ju), formatu podataka koje klijent očekuje (npr. JSON ili XML), jeziku koji se koristi, i slično. Odgovor servera također sadrži zaglavlje s dodatnim informacijama o odgovoru, kao što su vrsta sadržaja (engl. *Content-Type*), duljina odgovora i statusni kod koji označava ishod zahtjeva.

Tijelo odgovora sadrži stvarne podatke koje je server poslao kao odgovor na zahtjev. To može biti i u obliku običnog teksta (*plain text*) ili nekog formata za razmjenu podataka (JSON, XML i razni drugi). U svrhu sigurnosti i autentifikacije, API-ji često koriste različite mehanizme kao što su API ključevi, tokeni, OAuth i slično kako bi osigurali da samo ovlaštene aplikacije ili korisnici pristupaju njihovim resursima.

4.2. Upravljanje pristupom podacima API ključevi

API ključ je važan alat koji omogućava aplikacijama da se sigurno autentificiraju i pristupe uslugama i podacima putem API-ja. On pruža sigurnost, kontrolu i praćenje u komunikacija između aplikacija i API-ja, kao i način da se identificira i autorizira aplikacija prije nego što joj se omogući pristup određenim podacima ili uslugama, čineći cijeli proces integracije i razmjene podataka pouzdanim i efikasnim. Kroz API ključ (engl. *API key*), aplikacija će se autentificirati kako bi dobila pristup vremenskim podacima.

API ključ je kao digitalna „lozinka“ koju aplikacija koristi prilikom slanja HTTP zahtjeva API-ju. Kada se aplikacija registrira za pristup određenom API-ju, dobiva jedinstveni API ključ koji će koristiti u svakom zahtjevu koji šalje serveru [13]. Ovaj ključ se obično uključuje u zaglavlje HTTP zahtjeva kako bi se potvrdila autentičnost aplikacije.

Jedna od glavnih prednosti API ključa leži u aspektu sigurnosti. Implementacija API ključa osigurava da samo autorizirane aplikacije imaju pristup API-ju. Ovaj mehanizam zaštićuje osjetljive informacije od potencijalnih zloupotreba ili hakerskih napada. Kontrola pristupa vlasnicima API-ja da pažljivo prate i upravljaju koje aplikacije imaju pristup njihovoj usluzi putem API-ja. Ovo uključuje kvote za broj zahtjeva koje aplikacija može poslati u određenom vremenskom razdoblju. Ograničenja osiguravaju ravnomjernu raspodjelu resursa te sprječavaju preopterećenje sustava.

Ovaj nivo kontrole omogućava vlasnicima da postave ograničenja pristupa za svaku aplikaciju, uključujući kvote za upite ili definiranje specifičnih uvjeta pristupa. Analitika je treći ključni aspekt koji se postiže kroz korištenje API ključa. Vlasnici API-ja mogu koristiti ključ kako bi pratili kako se njihova usluga koristi od strane različitih aplikacija. To im omogućava da prikupljaju analitičke podatke o korištenju, omiljenim značajkama i drugim relevantnim informacijama. Ovi analitički podaci pomažu vlasnicima API-ja da bolje razumiju svoje korisnike i prilagode svoju uslugu kako bi bolje odgovarala njihovim potrebama.

Kako bi koristili API ključ za autentifikaciju, aplikacija će uključiti ključ u zaglavlje svakog HTTP zahtjeva koji šalje serveru [14]. Primjer zaglavlja sa sadržanim API ključem prikazan je na Slika 1.

Slika 1. Primjer zaglavlja sa API ključem [15]

X-API-Key je naziv zaglavlja koje prenosi API ključ. Server će provjeriti ovaj ključ i odobriti ili odbiti zahtjev na temelju njegove valjanosti. Kroz ovakav mehanizam autentifikacije, API ključ pruža siguran način da se aplikacija identificira i pristupi određenim uslugama ili podacima koje pruža API, čime se osigurava pouzdana i kontrolirana komunikacija između aplikacije i servera [16].

Rotacija ključeva je preporučena praksa koja dodatno povećava sigurnost. Ovaj proces uključuje redovitu promjenu API ključeva, što otežava potencijalnim zlonamjernim osobama da pristupe API-ju. Zamjena starog ključa novim zahtijeva ažuriranje aplikacije, čime se osigurava integritet pristupa.

Besplatni i plaćeni planovi su česta praksa kod API-ja. Ovisno o potrebama i resursima, API-ji često nude različite planove pretplate. Besplatni planovi često nude ograničene resurse u pogledu podataka i brzine zahtjeva, dok plaćeni

planovi omogućavaju više resursa i naprednije mogućnosti. Upravljanje dozvolama je još jedna korisna značajka. Vlasnici API-ja mogu odrediti kojim specifičnim uslugama ili resursima svaki API ključ može pristupiti, prilagođavajući pristup prema potrebama. Pružatelji API-ja mogu pratiti tko i kako koristi njihovu uslugu kroz API ključeve. Ako se primijeti nepravilna upotreba ili sumnjive aktivnosti, ključ se može suspendirati ili deaktivirati kako bi se zaštitili resursi i osigurala sigurnost.

4.3. Integracija Vremenskih API-ja

Zahtjev će sadržavati informacije o odabranoj lokaciji (gradu), a odgovor će biti u JSON formatu. Ovaj način komunikacije omogućuje aplikaciji da dinamički dohvati vremenske podatke za specifičan grad putem API-ja. JSON je popularan format za razmjenu podataka jer je lako čitljiv i parsiran od strane različitih programskih jezika. Kada aplikacija želi dobiti vremenske podatke za određeni grad putem API-ja, ona će oblikovati *HTTP GET* zahtjev s informacijom o lokaciji koja je odabrana (Slika 2.) [13].

```vbnet
GET /api/vremenski-podaci?lokacija=Zagreb HTTP/1.1
Host: api.weather-service.com
X-API-Key: your-api-key-here
```

Slika 2. Zahtjev za vremenske podatke za grad Zagreb [15]

U ovom primjeru, zahtjev traži vremenske podatke za grad Zagreb. *X-API-Key* je zaglavlje koje sadrži API ključ za autentifikaciju. Odgovor servera bit će u JSON formatu i sadržavat će informacije o vremenskim uvjetima za odabrani grad (Slika 3) [13].

Slika 3. Odgovor na zahtjev sa Slika 2 [15]

U ovom primjeru, odgovor sadrži informacije o temperaturi, vremenskim uvjetima, vlazi i lokaciji za grad Zagreb. Aplikacija može parsirati ovaj JSON odgovor kako bi prikazala relevantne podatke korisniku ili ih koristila za daljnje obrade. Kroz ovakav način komunikacije, aplikacija može dinamički dohvatiti vremenske podatke za različite gradove koristeći isti API putem različitih zahtjeva. JSON format omogućava laku razmjenu podataka između aplikacije i API-ja, olakšavajući integraciju i upotrebu podataka u aplikaciji.

Ovi API-ji često pružaju informacije kao što su temperatura, vlažnost, brzina vjetra, uvjeti oblačnosti i drugi relevantni meteorološki podaci. Osim osnovnih vremenskih podataka, neki vremenski API-ji također pružaju prognozu za buduće dane ili tjedne, omogućavajući aplikacijama da prikazuju dugoročne vremenske uvjete. Pri korištenju vremenskog API-ja, važno je uzeti u obzir sljedeće [14]:

- Brzina osvježavanja podataka: API-ji često imaju određene kvote ili ograničenja u brzini osvježavanja podataka. To znači da aplikacija neće uvijek moći dobiti trenutne podatke u stvarnom vremenu, već mora uzeti u obzir interval osvježavanja koje pruža API.
- Vremenska zona: Prilikom dobivanja vremenskih podataka, važno je uzeti u obzir vremensku zonu za odabranu lokaciju. Vremenski podaci se često daju u vremenskoj zoni određenoj od strane API-ja, pa je važno pravilno interpretirati ove informacije u lokalnoj vremenskoj zoni korisnika.
- Greške i stabilnost: API-ji mogu iskusiti greške ili privremene poteškoće u pružanju podataka. Stoga je dobra praksa implementirati odgovarajuće

rukovanje greškama u aplikaciji kako bi se korisnicima pružila pouzdana i stabilna usluga.

- Ažuriranja i dokumentacija: Vlasnici API-ja često ažuriraju svoje usluge i dokumentaciju. Ako planirate dugoročno koristiti vremenski API, važno je redovito provjeravati dokumentaciju i ažurirati svoj kod kako biste pratili eventualne promjene u API-ju.

Integracija s ovakvim API-jima zahtijeva razumijevanje njihovih karakteristika, kako bi se osigurala točnost, stabilnost i dobro iskustvo korisnika prilikom pristupa vremenskim informacijama.

4.4. Prikaz vremenskih podataka kroz interaktivno grafičko korisničko sučelje (GUI)

Nakon obrade odgovora, podaci će biti proslijeđeni do GUI-a. Podaci dobiveni iz JSON odgovora koriste se za oblikovanje prezentacije na grafičkom korisničkom sučelju (GUI) [16].

Način na koji se ti podaci prikazuju ovisi o dizajnu aplikacije i potrebama korisnika. Vremenski podaci mogu biti prikazani na nekoliko načina, uključujući tekstualni prikaz, grafičke elemente, mape i animacije. Tekstualni prikaz pruža osnovne informacije poput temperature, vlažnosti i vremenskih uvjeta putem jasnih tekstualnih elemenata na ekranu. Grafički prikazi, kao što su grafikoni ili ikone, omogućuju vizualizaciju promjena, poput temperaturnih trendova ili vremenskih ikona. Mape s oznakama koriste se ako je fokus aplikacije na prikazu vremenskih podataka za različite lokacije.

Na karti se označavaju različite lokacije s odgovarajućim oznakama koje pružaju informacije o vremenskim uvjetima na tim mjestima. Animacije se koriste za prikazivanje promjena u vremenskim uvjetima kroz vrijeme, što je korisno za prognoze i vizualizaciju dinamičnih promjena u uvjetima. Konačni odabir načina prikaza vremenskih podataka ovisi o dizajnu aplikacije i preferencijama korisnika. Cilj je pružiti informacije na način koji je lako razumljiv, informativan i estetski

privlačan, osiguravajući da korisnici dobiju sveobuhvatan uvid u meteorološke uvjete na intuitivan način. Napredniji GUI-ji često omogućavaju interakciju korisnika s prikazanim podacima. Na primjer, korisnici mogu odabrati različite gradove ili datume kako bi vidjeli vremenske podatke za različite trenutke ili lokacije.

Cjelokupan proces uključuje nekoliko ključnih koraka i aspekata. Kreiranje GUI-a je početni korak gdje se koriste grafičke komponente iz odgovarajućih biblioteka ili okvira. Ovisno o programskom jeziku i platformi, primjenjuju se alati poput *Tkinter*-a za Python, *JavaFX*-a za Java, ili *SwiftUI*-a za Swift, kako bi se stvorio osnovni okvir za prikaz podataka [17].

Prikazivanje podataka znači da se oblikovani vremenski podaci umetnu u odgovarajuće sekcije GUI-a. Primjerice, temeljne informacije poput temperature i trenutnih uvjeta mogu biti prikazane na vidljivom mjestu, dok se dodatni detalji kao što su vlaga i brzina vjetra mogu pružiti u nižim dijelovima sučelja.

Ažuriranje podataka je bitno za osiguravanje aktualnosti informacija. Ako se vremenski podaci mijenjaju tijekom vremena, aplikacija će redovito osvježavati prikazane podatke kako bi korisnicima pružila najnovije informacije i prognoze. Uz sve ovo, važno je osigurati da GUI bude odgovarajuće prilagođen platformi na kojoj aplikacija radi. Na primjer, GUI za mobilne uređaje treba biti prilagođen načinu na koji korisnici koriste dodirne zaslone.

5. IZRADA MOBILNE APLIKACIJE U PROGRAMSKOM JEZIKU *SWIFT*

Nakon što je sučelje i protokol aplikacije jasno definiran, slijedi faza implementacije mobilne aplikacije. Ova faza uključuje niz ključnih koraka koji omogućuju funkcionalnost, praktičnost i estetski privlačan dizajn krajnjeg proizvoda. Prva faza implementacije usredotočuje se na razvijanje osnovne strukture aplikacije. To uključuje stvaranje korisničkog sučelja (UI) prema definiranim smjernicama dizajna. Koristeći relevantne alate i okvire za razvoj mobilnih aplikacija, poput *React Native-a* za višestruke platforme ili *Swift-a* za iOS, razvijaju se komponente sučelja kao što su unosno polje za odabir grada i gumb za pokretanje traženja vremenskih podataka [18].

Nakon stvaranja sučelja, faza slanja zahtjeva prema API-ju uslijedila bi kao ključan korak. Aplikacija će koristiti uneseni grad kako bi generirala zahtjev prema odgovarajućem vremenskom API-ju. Ovaj zahtjev sadrži podatke o željenom gradu i očekuje odgovor s vremenskim informacijama u JSON formatu. Obrada odgovora API-ja sljedeći je korak. Kada se odgovor primi, aplikacija će izvršiti parsiranje JSON podataka kako bi izvukla relevantne informacije poput temperature, uvjeta, vlage i brzine vjetra. Ovi podaci zatim se pripremaju za prikazivanje na korisničkom sučelju. Nakon obrade odgovora API-ja, podaci se dinamički ubacuju u odgovarajuće dijelove korisničkog sučelja.

U završnoj fazi, aplikacija se testira kako bi se osiguralo da sve komponente ispravno funkcioniraju i da korisničko iskustvo ispunjava očekivanja. Eventualne greške ili nedostaci ispravljaju se prije konačnog izdanja aplikacije. Kroz cjelokupan proces, implementacija mobilne aplikacije za prikaz vremenskih informacija zahtijeva pažljivo planiranje, suradnju dizajnera i programera te usmjerenost na stvaranje korisničkog sučelja koje je intuitivno, funkcionalno i atraktivno.

5.1. *Swift* programski jezik

Swift je moderni objektno orijentirani programski jezik koji je razvijen za pisanje softverskih aplikacija i programa za razne Apple proizvode. Predstavljen je 2014. godine na *Apple Worldwide Developers Conference* (WWDC) kao odgovor na potrebu za naprednijim i efikasnijim programskim jezikom za razvoj aplikacija za macOS, iOS, watchOS i tvOS platforme. *Swift* je osmišljen kako bi poboljšao produktivnost programera, omogućio brže izvođenje koda i pružio jednostavnije i sigurnije okruženje za razvoj [19]. Jedan od ključnih aspekata Swifta je njegova sigurnost i moderan dizajn. Ovaj jezik je dizajniran s naglaskom na minimiziranje mogućih programskih grešaka i olakšavanje otkrivanja i ispravljanja tih grešaka.

Nudi napredne mehanizme zaštite kao što su opcionalni tipovi i automatsko upravljanje memorijom, što pomaže smanjiti broj grešaka povezanih s upravljanjem memorijom koje su česte u drugim jezicima. Osim toga, *Swift* također posjeduje čitljiv sintaksni stil koji je lak za razumijevanje i pisanje. Značajke kao što su inferencija tipova olakšavaju programerima da napišu čist i razuman kod, smanjujući potrebu za eksplicitnim navođenjem tipova varijabli ili konstanti. Značajna prednost *Swift-a* je i njegova iznimno brza izvedba. *Swift* je optimiziran za rad na Appleovim uređajima što omogućava aplikacijama napisanim u ovom jeziku da brže rade i bolje iskorištavaju resurse uređaja.

Tablica. 1 Tipovi podataka, konstante i varijable [20]

Tip podataka	Opis
Int	Koristi se za prikaz cijelih brojeva
Double	Koristi se za prikaz 64-bitnih decimalnih brojeva
Float	Koristi se za prikaz 32-bitnih decimalnih brojeva
Character	Koristi se za prikaz isključivo jednog znaka (simbola)
Bool	Može sadržavati smo dvije vrijednosti – istina (*true*) ili laž (*false*)
Optional	Može sadržavati omotanu vrijednost ili vrijednost može biti odsutna
Enumeracija	Tip podataka za grupu povezanih vrijednosti

Osim toga, *Swift* je aktivno podržavan od strane *Applea* i zajednice programera. Neprestano se ažurira i unaprjeđuje kako bi se osigurala bolja funkcionalnost i poboljšalo korisničko iskustvo. Također, *Swift* je postao sve popularniji izbor za razvoj aplikacija za *Apple* platforme, a postoji i mogućnost korištenja *Swifta* u kombinaciji s postojećim *Objective-C* kodom, omogućavajući postupno prelazak na novi jezik.

5.2. SwiftUI

Godine 2019., Apple je predstavio revolucionarni radni okvir pod imenom *SwiftUI*, označavajući korak naprijed u izradi korisničkih sučelja na svojim platformama poput iOS-a, macOS-a, tvOS-a i watchOS-a. Ovaj inovativni alat omogućuje programerima kreiranje bogatih korisničkih sučelja korištenjem jednostavnog skupa alata i API-ja, unificirajući razvoj na različitim platformama. Centralna karakteristika *SwiftUI-ja* je deklarativni pristup izgradnji sučelja, gdje se

izgled i ponašanje sučelja opisuju na način koji je čitljiv i intuitivan za razumijevanje [19].

SwiftUI se temelji na konceptu "pogleda" (engl. *view*). Pogled je osnovna komponenta koja čini korisničko sučelje i koja se može konfigurirati putem modifikatora [19]. Kada se stanje pogleda promijeni, novi izgled se automatski generira i prikazuje na zaslonu. Ova deklarativna paradigma olakšava razvoj, čitanje i održavanje koda. Jedna od ključnih prednosti *SwiftUI*-ja je njegova integracija s jezikom Swift. Primjena *SwiftUI*-ja može se vidjeti u mnogim aplikacijama, uključujući popularnu Apple-ovu aplikaciju *Weather*. Koristeći ovaj okvir, programeri deklariraju kako žele da njihova korisnička sučelja izgledaju, a *SwiftUI* automatski generira ispravna sučelja na temelju trenutnog stanja aplikacije.

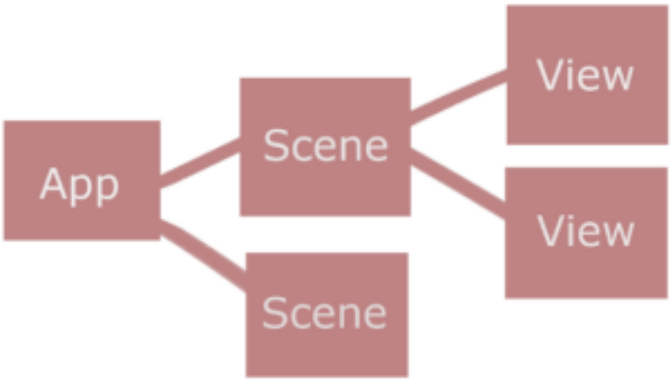
Bitno je napomenuti da *SwiftUI* omogućuje jednostavno rukovanje animacijama, pružajući dinamičan izgled aplikacija. Ipak, važno je imati na umu da *SwiftUI* zahtijeva novije verzije operativnih sustava na uređajima na kojima će se aplikacije izvoditi. Iako je dostupan od iOS 13 verzije, bilo je potrebno neko vrijeme za zreliju komercijalnu primjenu. Sve u svemu, *SwiftUI* predstavlja velik korak prema naprijed u razvoju korisničkih sučelja za Apple platforme [20].

5.3. Struktura *SwiftUI* aplikacije

Svaka *SwiftUI* aplikacija počinje stvaranjem strukture koja nasljeđuje *App* protokol, a ova struktura se označava ključnom riječi *@main* kako bi se pridružila osnovnoj implementaciji glavne funkcije. *App protokol* predstavlja okvir za definiranje strukture i ponašanja aplikacije. Središnja točka svake aplikacije je njeno tijelo, odnosno atribut *body*. Ovdje se definira sadržaj aplikacije, što uključuje različite scene, prikaze i njihove kompozicije [23].

Svaka scena u aplikaciji je sastavljena od različitih prikaza koje korisnik vidi na zaslonu (Slika 4). Prikazi su osnovne komponente sučelja koje se mogu kombinirati kako bi se postigao željeni izgled i funkcionalnost. Ovaj pristup

omogućuje programerima da deklarativno opišu strukturu aplikacije, bez potrebe za ručnom manipulacijom s prikazima na niskom nivou. Ključna prednost ovog pristupa je što omogućuje programerima da se usredotoče na opisivanje željenog izgleda i ponašanja aplikacije, a da se detalji o tome kako će se to postići prepuštaju *SwiftUI* okviru. Kroz deklarativni pristup i komponiranje prikaza, *SwiftUI* omogućuje izgradnju dinamičnih i privlačnih korisničkih sučelja.



Slika 4. Struktura *SwiftUI* aplikacije [20]

Jedan od ključnih aspekata koje donosi *SwiftUI* je njegova orijentacija prema strukturama umjesto klasa, što pozitivno utječe na performanse izvođenja aplikacije. U *Swift* programskom jeziku, strukture su tipovi vrijednosti (engl. *value type*), što znači da se tretiraju kao bilo koja druga vrijednost i mogu se bezbrižno prenositi s jednog mjesta na drugo, bez potrebe za brigu o upravljanju memorijom ili održavanju pokazivača.

Svaki prikaz (engl. *view*) u *SwiftUI* mora naslijediti *View* protokol, što implicira da mora implementirati atribut *body* koji vraća točno jedan tip prikaza. Ovaj atribut definira sadržaj prikaza i njegovo ponašanje [23]. Korisničko sučelje se konstruira komponiranjem različitih prikaza kako bi se prikazalo trenutno stanje aplikacije. Jedna od korisnih praksi u razvoju korisničkog sučelja je razdvajanje različitih dijelova sučelja u manje prikaze. Ovo pruža veću jasnoću koda, omogućava višestruko korištenje istih prikaza na različitim mjestima unutar aplikacije te

smanjuje kompleksnost koda i mogućnost grešaka. Ovakvom strukturom, razvoj postaje efikasniji i održavanje aplikacije postaje manje sklono problemima.

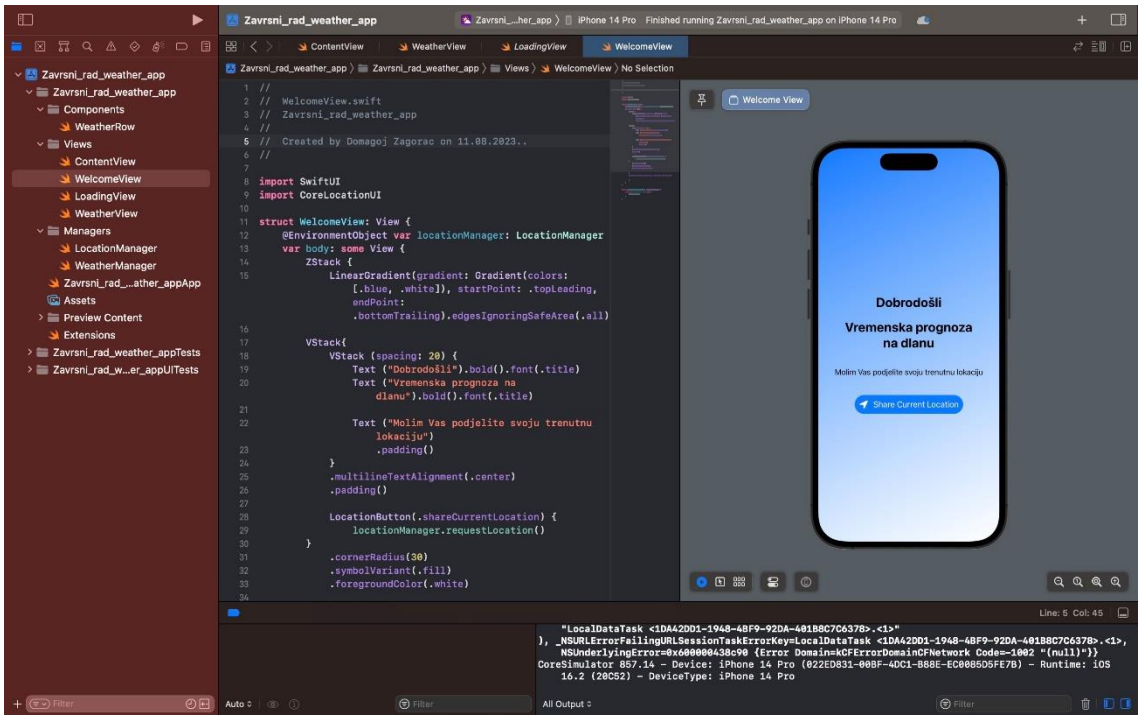
SwiftUI također omogućuje konfiguriranje prikaza putem modifikatora. Modifikatori su metode koje se primjenjuju na prikazima kako bi se promijenio njihov izgled ili ponašanje. Oni se mogu ulančavati jedan iza drugoga, stvarajući tako lanac modifikacija koje definiraju konačan izgled prikaza. *SwiftUI* pristup strukturama, deklarativnom definiranju prikaza te mogućnosti komponiranja i konfiguriranja prikaza putem modifikatora značajno doprinosi efikasnosti i čitljivosti koda, što rezultira optimalnim i intuitivnim korisničkim sučeljima na *Apple* platformama.

SwiftUI također koristi omotače svojstava (engl. *property wrappers*) za upravljanje stanjem. Ovi omotači, kao što su @Binding, @State, @StateObject, @ObservedObject, @Environment i drugi, omogućuju programerima da elegantno rješavaju složene scenarije upravljanja stanjem, umjesto korištenja tradicionalnih obrazaca poput *delegate* [24].

6. VREMENSKA PROGNOZA NA DLANU: *SWIFT* APLIKACIJA SA *OPENWEATHERMAP* API

6.1. Postavljanje projekta

Prvi korak je priprema. U ovoj fazi priprema se osnova za razvoj Vremenske aplikacije pomoću *OpenWeatherMap* API-ja. Ona uključuje stvaranje novog projekta u *Xcode* razvojnom okruženju, definiranje ključnih parametara projekta te osiguravanje važećeg API ključa s platforme *OpenWeatherMap*. U prvom koraku otvoren je *Xcode* i odabrana je opcija za stvaranje novog iOS projekta (Slika 5). Definiran je naziv projekta koji će biti vidljiv korisnicima i na App Store platformi. Prilikom izrade projekta, odabrani su naziv projekta koji će služiti kao identifikacija. Također, *Swift* je izabran kao programski jezik je brz, učinkovit i jednostavno se integrira s iOS-om.



Slika 5. Prikaz projekta u *Xcode* razvojnom okruženju

Kao ciljna platforma odabran je iPhone uređaj. Da bi aplikacija mogla pristupiti vremenskim podacima putem *OpenWeatherMap* API-ja, važno je posjedovati API ključ. Registracija na *OpenWeatherMap* platformi je preduvjet za generiranje ovog ključa. Generirani API ključ omogućava aplikaciji komunikaciju s poslužiteljima *OpenWeatherMap*. Pravilna izvedba ovog koraka postavlja temelje za razvoj aplikacije. Pravilno definiranje projekta i osiguranje API ključa su ključni za uspješno korištenje podataka o vremenskim uvjetima iz *OpenWeatherMap* API-ja u *iOS* aplikaciji.

Nakon završetka ovog prvog koraka, može se prijeći na sljedeće korake kako bi se postupno razvijale funkcionalnosti aplikacije i omogućilo korisnicima pristupanje vremenskim podacima putem *OpenWeatherMap* API-ja. U drugom koraku ulazi se u korisničko sučelje vremenske aplikacije. Ovdje su dodani potrebni elementi za unos podataka, postavljeni odgovarajući dizajnerski zahtjevi te izvršeno povezivanje elemenata s odgovarajućim akcijskim metodama u *ViewControlleru*.

6.2. API komunikacija i obrada JSON odgovora

Unutar korisničkog sučelja, postavlja se *UITextField* kako bi se korisnicima omogućio unos naziva grada za koji se traže vremenski podaci. Dodaje se *DropDownMenu* sa opisom "Odaberi grad" i izborom dostupnih gradova. Koordinate za dostupne gradove spremljene su memoriji.

Odabirom grada pokreće se odgovarajuća metoda za traženje vremenskih podataka. Ovaj korak omogućava korisnicima da koriste aplikaciju putem korisničkog sučelja i time dobe pripadajuće vremenske informacije.

U trećem koraku, izvršava se slanje zahtjeva API-ju *OpenWeatherMap* prilikom odabira grada iz *DropDownMenu* (Slika 6). Korišten je *URLSession* za slanje *GET* zahtjeva. Ova interakcija omogućava komunikaciju aplikacije sa serverom kako bi se dobili traženi vremenski podaci. Slanjem zahtjeva GET okida se *endpoint* na strani *OpenWeatherMap*. Na toj strani ulazi se u *backend* koji sa

svojom definiranom logikom povlači podatke iz baze podataka, te ih takve prosljeđuje kao odgovor. Kao parametre, važno je poslati geografsku širinu i dužinu, API ključ i mjerne jedinice u kojima želimo odgovor.

```swift
//
//  WeatherManager.swift
//  Zavrsni_rad_weather_app
//
//  Created by Domagoj Zagorac on 06.09.2023..
//

import Foundation
import CoreLocation

class WeatherManager {
    func getCurrentWeather(lattitude: CLLocationDegrees, longitude: CLLocationDegrees) async throws -> ResponseBody {
        guard let url = URL(string:
            "api.openweathermap.org/data/2
            .5/weather?lat=\(lattitude)&lon=\(longitude)&appid=\("cc5f952b219eb5b9869796a923680839")&units=metric")
        else { fatalError("Nepostojeci URL")}

        let urlRequest = URLRequest (url: url)

      let (data, response) = try await URLSession.shared.data(for: urlRequest)

        guard (response as? HTTPURLResponse)?.statusCode == 200 else { fatalError("Error prilikom dobavljanja podataka o vremenu")}

        let decodedData = try JSONDecoder().decode(ResponseBody.self, from: data)

        return decodedData
    }
}
```

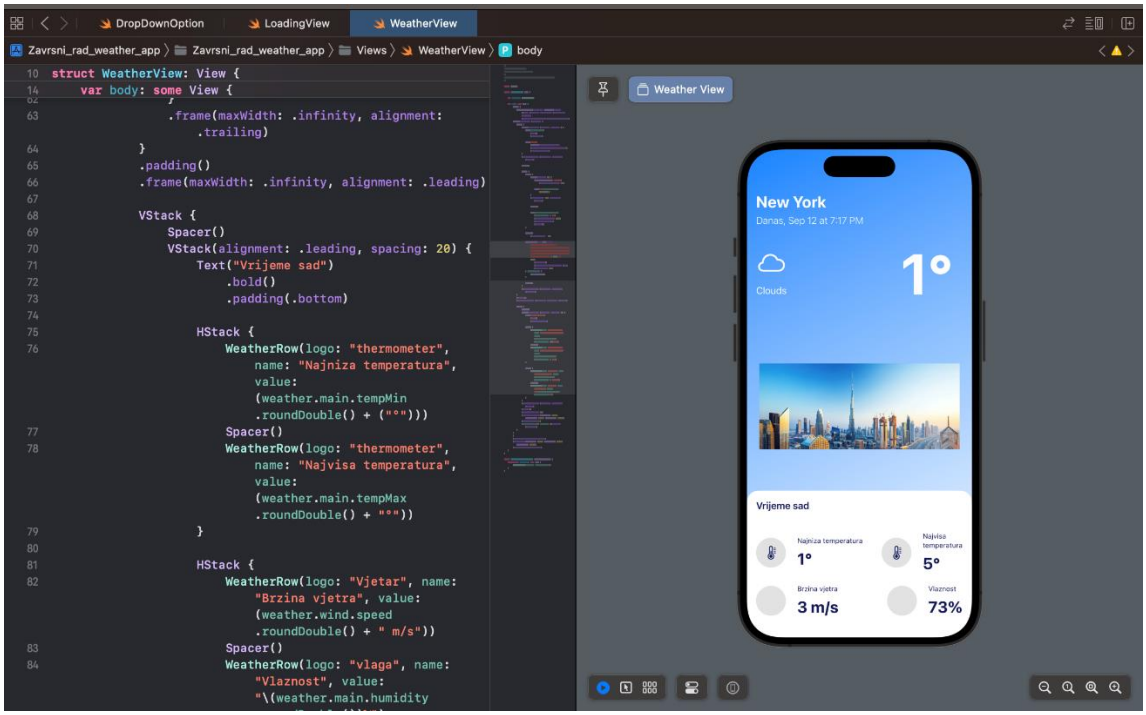
Slika 6. Slanje zahtjeva na *OpenWeatherMap* API

Dodavanjem API ključa kao parametra URL-u, *OpenWeatherMap backend* odrađuje zahtjev sukladno pravima pristupa podaka.

U četvrtom koraku, obrađuje se API. Ovdje se primljeni JSON odgovor pretvara u objekte s kojima se može dalje manipulirati, te se time pronalaze bitne informacije o vremenskim uvjetima. Koristi se *JSONSerialization*, koji omogućuje parsiranje JSON odgovora u iskoristive objekte s vremenskim podacima. To može uključivati informacije o temperaturi, vlažnosti, brzini vjetra i drugima. Nakon obrade, ovi podaci mogu se koristiti za prikaz korisnicima i daljnju analizu vremenskih uvjeta.

6.3. Grafičko korisničko sučelje

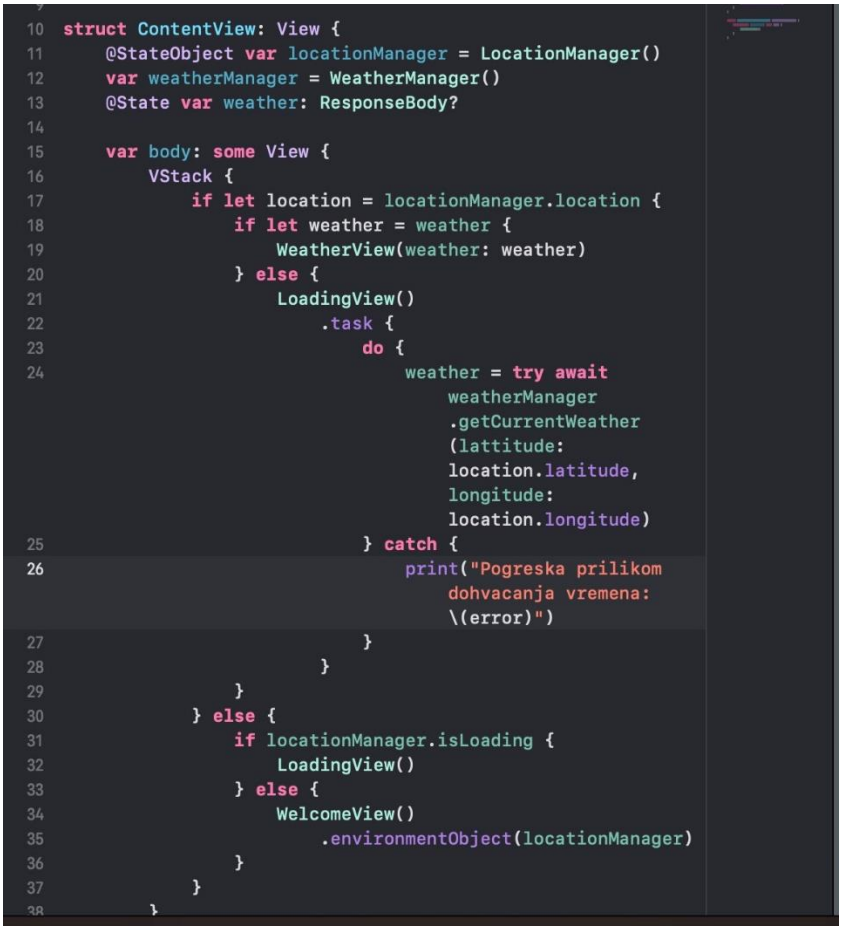
U petom koraku, realizira se grafički prikaz dobivenih informacija o vremenskim uvjetima. Za ovo se kreiraju odgovarajući pogledi unutar korisničkog sučelja, kao što su *ContentView* ili *WeatherView*, kako bi se vizualno predstavile informacije korisnicima. Unutar korisničkog sučelja, stvaraju se odgovarajući elementi kao što su *VStack*-i (vertikalni pregled elemenata) i *HStack* (horizontalni pregled elemenata) za prikaz vremenskih informacija. Ovo je važno kako bi se postigao pregledan i intuitivan prikaz informacija. Nakon što su podaci o vremenskim uvjetima obrađeni, ti podaci se umeću u odgovarajuće medijske elemente korisničkog sučelja. Na primjer, temperaturni podaci i ikona vremenskih uvjeta prikazuje se unutar *WeatherRow*. Ovaj korak omogućava da korisnici vizualno percipiraju prikupljene vremenske podatke.



Slika 7. *View* s prikazom informacija o vremenskim uvjetima

U šestom koraku, posvećuje se pažnja rukovanju pogreškama koje mogu nastati tijekom procesa slanja zahtjeva i obrade odgovora. Svrha ovog koraka je osigurati da se korisnici informiraju o eventualnim problemima i greškama kako bi se pružilo bolje korisničko iskustvo (Slika). Ovo omogućava detekciju i kontrolu nad mogućim greškama tijekom tih koraka. U slučaju problema kao što su slaba internetska veza ili neuspješna analiza odgovora, korisnicima se prikazuju obavijesti. Prikazivanje informativnih obavijesti korisnicima, prilikom grešaka, pomaže u održavanju pozitivnog korisničkog iskustva i olakšava rješavanje potencijalnih problema.

U sedmom koraku, provodi se testiranje vremenske aplikacije u različitim okruženjima, gradovima i situacijama kako bi se osigurao točan prikaz vremenskih podataka i funkcionalnost aplikacije. Aplikacija se testira uz pomoć različitih gradova i scenarija kako bi se provjerila njezina funkcionalnost i reakcija na različite vremenske uvjete.



```
struct ContentView: View {
    @StateObject var locationManager = LocationManager()
    var weatherManager = WeatherManager()
    @State var weather: ResponseBody?

    var body: some View {
        VStack {
            if let location = locationManager.location {
                if let weather = weather {
                    WeatherView(weather: weather)
                } else {
                    LoadingView()
                        .task {
                            do {
                                weather = try await
                                    weatherManager
                                    .getCurrentWeather
                                    (lattitude:
                                    location.latitude,
                                    longitude:
                                    location.longitude)
                            } catch {
                                print("Pogreska prilikom
                                    dohvacanja vremena:
                                    \(error)")
                            }
                        }
                }
            } else {
                if locationManager.isLoading {
                    LoadingView()
                } else {
                    WelcomeView()
                        .environmentObject(locationManager)
                }
            }
        }
```

Slika 8. Sustav rukovanja pogreška

Ovo testiranje osigurava da aplikacija radi ispravno i konzistentno bez obzira na varijacije. Glavni cilj testiranja je osigurati da aplikacija točno prikazuje vremenske podatke za odabrane gradove. Ovo uključuje provjeru jesu li prikazane temperature, ikone vremenskih uvjeta, brzina vjetra i ostale informacije točne i u skladu s očekivanjima. Tijekom testiranja, provjerava se ispravnost svih funkcionalnosti aplikacije, kao što su unos grada, slanje zahtjeva API-ju, obrada odgovora, prikaz podataka i rukovanje pogreškama. Cilj je identificirati potencijalne probleme i osigurati da aplikacija radi u idealnim uvjetima bez ikakvih problema.

6.4. Moguće dodatne funkcionalnosti

U osmom koraku, razmatra se implementacija dodatnih funkcionalnosti u vremenskoj aplikaciji koje obogaćuju korisničko iskustvo. Ove dodatne funkcionalnosti nisu obavezne, ali mogu doprinijeti interaktivnosti aplikacije. Jedna od takvih korisnicima omogućava dodavanje omiljenih gradova na listu. To olakšava brz pristup vremenskim podacima za više gradova bez potrebe za ponovnim unosom. Uvođenje animacija ili dodatnih informacija, kao što su detalji o UV indeksu, predviđanja vjetra ili mogućnosti padalina, još više obogaćuje korisničko iskustvo. Animacije mogu donijeti dinamičnost aplikaciji, dok dodatne informacije pružaju korisnicima dublji uvid u vremenske uvjete.

U devetom koraku, razmatra se opcionalno poboljšanje izgleda korisničkog sučelja vremenske aplikacije. Ovo poboljšanje uključuje primjenu automatskog izgleda (engl. *autolayout*) za osiguranje konzistentnog prikaza na različitim uređajima te korištenje odgovarajućih grafičkih resursa kako bi se estetski podigao izgled aplikacije. Upotrebom automatskog izgleda, osigurava se da se elementi korisničkog sučelja pravilno pozicioniraju i skaliraju na različitim veličinama i orijentacijama uređaja. Za poboljšanje vizualnog izgleda aplikacije, koriste se odgovarajući grafički resursi kao što su ikone, pozadine ili stilizirani elementi.

U desetom koraku, vremenska aplikacija je dovedena do završne faze prije distribucije. Ovaj korak uključuje posljednje pripreme, testiranje i postavljanje aplikacije na App Store radi dostupnosti korisnicima. Ikona aplikacije, naziv aplikacije, opis i druge relevantne informacije su postavljeni prema smjernicama App Store-a. Ovi detalji su ključni za privlačenje korisnika i stvaranje prepoznatljivosti aplikacije.

Zadnji korak uključuje kreiranje *developer* računa na *Apple Developer Portalu*, slanje aplikacije na pregled, odobrenje i konačno postavljanje na App Store kako bi korisnici mogli preuzeti aplikaciju.

7. ZAKLJUČAK

U ovom radu detaljno smo istražili ključne koncepte i tehnologije povezane s izradom aplikacije za vremensku prognozu temeljene na API-ju. Kroz razmatranje sučelja za programiranje aplikacija (API), naglasili smo važnost API-ja u osiguravanju ažuriranih vremenskih informacija za aplikacije putem formata razmjene podataka kao što je JSON. Grafičko korisničko sučelje (GUI) također je bilo ključno za intuitivan i jednostavan pristup podacima o vremenu.

Razmotrili smo protokol dobivanja informacija putem API-ja, uključujući HTTP zahtjeve i upravljanje pristupom podacima putem API ključeva. Integracija vremenskih API-ja omogućila je dobivanje relevantnih vremenskih podataka za našu aplikaciju. Povezali smo te podatke s interaktivnim grafičkim korisničkim sučeljem (GUI), omogućujući korisnicima jednostavan i privlačan način za pregled informacija o vremenu.

Izrada mobilne aplikacije korištenjem *Swift* programskog jezika i *SwiftUI* tehnologije omogućila nam je brzu i efikasnu razvojnu okolinu. Ubrzali smo proces izrade korisničkog sučelja kroz korištenje elemenata korisničkog sučelja koje nudi *SwiftUI*. Kao praktičan primjer, implementirali smo aplikaciju za vremensku prognozu temeljenu na *OpenWeatherMap* API-ju koristeći *Swift* programski jezik. Ova aplikacija omogućila je korisnicima da na brz i jednostavan način pristupe ažuriranim vremenskim informacijama na svom mobilnom uređaju.

Rad pruža uvid u proces izrade aplikacije temeljene na API-ju s fokusom na informacije o vremenu. Kombinacija API-ja, grafičkog korisničkog sučelja i modernih tehnologija poput *Swift* programskog jezika i *SwiftUI* tehnologije omogućuje razvoj aplikacija koje pružaju korisnicima korisno i privlačno iskustvo. Ovaj rad može poslužiti kao osnovni vodič za razvojne inženjere i programere koji žele istražiti ovu temu dalje i stvoriti slične aplikacije za svoje korisnike.

8. LITERATURA

[1] Crockford, D.: The application/json Media Type for JavaScript Object Notation (JSON) (2006.), https://tools.ietf.org/html/rfc4627, pristupljeno 15.8.2023.
[2] Ecma International.: Standard ECMA-404: The JSON Data Interchange Syntax (2nd ed.) (2017.), https://www.ecmainternational.org/publications/standards/Ecma-404.htm, pristupljeno 15.8.2023.
[3] Fielding, R., Gettys, J., Mogul, J., Berners-Lee, T., et al.: Hypertext Transfer Protocol: HTTP/1.1. (1999.), https://tools.ietf.org/html/rfc2616, pristupljeno 16.8.2023.
[4] Fielding, R., & Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content (2014.), https://tools.ietf.org/html/rfc7231, pristupljeno 14.8.2023.
[5] RapidAPI.: API glossary: API Key – What is an API Key (2015.), https://rapidapi.com/blog/api-glossary/api-key/, pristupljeno 17.8.2023.
[6] RapidAPI: Types of APIs (2020.), https://rapidapi.com/blog/types-of-apis/, pristupljeno 17.8.2023.
[7] SWIFT.: Application Programming Interfaces: Delivering a global platform for the financial services API economy (2018.), https://www.swift.com/sites/default/files/documents/swift_standards_whitepaper_api.pdf, pristupljeno 17.8.2023.
[8] Losari, A.: Corona Virus Stats & Advices App with SwiftUI (2012.), https://github.com/alfianlosari/CoronaVirusTrackerSwiftUI, pristupljeno 17.8.2023.
[9] Apple: SwiftUI tutorials: Introducing SwiftUI (2010.), https://developer.apple.com/tutorials/swiftui, pristupljeno 17.8.2023.
[10] Cocoacasts: What is SwiftUI? (2014.), https://cocoacasts.com/swiftui-fundamentals-what-isswiftui#:~:text=SwiftUI%20is%20Apples%20brand%20new,is%20a%20cross%2Dplatform%20framework, pristupljeno 17.8.2023.

[11] Apple: App Structure and Behavior (2010.), https://developer.apple.com/documentation/swiftui/app-structure-and-behavior, pristupljeno 17.8.2023.
[12] Apple: SwiftUI View (2010.), https://developer.apple.com/documentation/swiftui/view/, pristupljeno 17.8.2023.
[13] Breur, R.: The SwiftUI render loop (2022.), https://rensbr.eu/blog/swiftui-renderloop/#:~:text=Just%20like%20UIKit%20%2C%20SwiftUI%20is,render%22loop%20of%20an%20application, pristupljeno 17.8.2023.
[14] Balkaya, C.: Animations in SwiftUI: Get to Know Transactions (2021.), https://betterprogramming.pub/animation-in-swiftui-get-to-know-transactions7cd57cfb299f, pristupljeno 18.8.2023.
[15] The Weather Channel: The Weather Channel Surpasses 100 Million App Downloads Across Smartphones and Tablets (2013.), http://www.theweathercompany.com/newsroom/2014/08/19/weather-channel-surpasses-100-million-app-downloads-across-smartphones-and, pristupljeno 18.8.2023.
[16] Šarić, M.: Izrada korisničkog sučelja, Seminarski rad, Sveučilište Josipa Jurja Strossmayera u Osijeku, Osijek, (2018.)
[17] Tomiša, M.: Određivanje kvalitativnih kriterija dizajna grafičkoga proizvoda u procesu grafičke komunikacije, Doktorski rad, Sveučilište u Zagrebu, Grafički fakultet, Zagreb, (2012.)
[18] Kliček, M.: Dizajniranje korisničkog sučelja mobilne aplikacije Sveučilišta Sjever, Završni rad, Sveučilište Sjever, Varaždin, (2016.)
[19] Tomiša, M., Milković, M.: Grafički dizajn i komunikacija, Veleučilište u Varaždinu, Varaždin, (2011.)
[20] SWIFT: Application Programming Interfaces: Delivering a global platform for the financial services API economy, (2018.), https://www.swift.com/sites/default/files/documents/swift_standards_whitepaper_api.pdf, pristupljeno 18.8.2023
[21] Apple: About the Weather app and icons on your iPhone and iPod touch, (2017.), https://support.apple.com/en-us/HT207492, pristupljeno 18.8.2023

[22] Bomhold, C. R.: Educational Use of Smart Phone Technology: A Survey of Mobile Phone Application Use by Undergraduate University Students, Program: electronic library and information systems, 47 (2013.), 5, 424-436, doi:10.1108/PROG-01-2013-0003
[23] Corso, R. A.: Local Television News is the Place for Weather Forecasts for a Plurality of Americans, The Harris Poll, Harris Interactive, (2007.), http://www.harrisinteractive.com/vault/Harris-Interactive-Poll-Research-Weather-2007-11.pdf, pristupljeno 18.8.2023
[24] National Weather Service: Weather.gov on Your Mobile Phone, (2016.), http://www.nws.noaa.gov/com/weatherreadynation/mobilephone.html, pristupljeno 19.8.2023

9. PRILOZI

9.1. Popis slika

POPIS SLIKA:

Stranica

9.2. Popis tablica

POPIS TABLICA:

Stranica