APLIKACIJA ZA SAMOSTALNO INTERAKTIVNO RAZGLEDAVANJE

Marinović, Božo

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

BOŽO MARINOVIĆ

Z A V R Š N I R A D

APLIKACIJA ZA SAMOSTALNO INTERAKTIVNO RAZGLEDAVANJE

Split, rujan 2021.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

Predmet: Programiranje u C#

Z A V R Š N I R A D

Kandidat: Božo Marinović

Naslov rada: Aplikacija za samostalno interaktivno razgledavanje

Mentor: pred. Josip Vrlić, dipl. ing. rač.

Split, rujan 2021.

SADRŽAJ

SAŽETAK

Cilj ovog završnog rada je izrada web aplikacije koja će služiti za pametno razgledavanje grada korištenjem WhatsApp komunikacijske platforme. Ideja aplikacije je da korisnici putem internetske stranice pristupaju informacijama o pojedinim atrakcijama i kreiraju korisnički račun na temelju kojeg mogu sastaviti vlastiti turistički obilazak. Nakon kreiranja turističkog obilaska isti se započinje slanjem poruke putem WhatsAppa ili skeniranjem QR kôda. Isti obilazak može koristiti više osoba čiji se napredak može uspoređivati, a isti se može i dijeliti putem telefonskog broja registriranog korisnika. U Web aplikaciji definirano je više rola korisnika: administrator i korisnik. Administrator je zadužen za kreiranje novih atrakcija, uređivanje podataka o atrakciji i upravljanje korisničkim računima. Sama aplikacija je koncipirana od programskog aplikacijskog sučelja *(engl. application programming interface, API)*, klijentske aplikacije i konzolne aplikacije za slanje i primanje WhatsApp poruka. Aplikacija je izrađena korištenjem Infobip programskog aplikacijskog sučelja i softverskog okvira Blazor (koji inkomponira .NET Core programski okvir) koji omogućava kreiranje web aplikacija korištenjem programskih jezika C# i HTML.

Ključne riječi: web aplikacija, Infobip API, WhatsApp, C#, .NET Core, HTML, Blazor, pametni obilazak grada

SUMMARY

Application for independent interactive tours

The goal of this thesis is the development of web application for smart city tour using WhatsApp platform. The idea of this application is for users to access attraction data and create accounts that allows them to create tours based on their preferences. After the user creates his tour he can start it by sending WhatsApp message or by scanning the QR code. Tours can be shared via phone number which allows users that share the same tour to compare progress with each other. Application users can have either user or administrator role. The administrator is in charge of creating new attractions, attraction management and user accounts management. The application is conceived of application programming interface, client application and console application for WhatsApp communication. Application was made by using Infobip application programming interface and software framework Blazor (that includes .NET Core framework) that allows creation of web applications using programming languages C# and HTML.

Key words: web application, Infobip API, Whatsapp, C#, .NET Core, HTML, Blazor, smart city tour

1. UVOD

Turistički obilasci grada su danas gotovo nezamislivi bez turističkih vodiča i unaprijed definiranih obilazaka. Upravo odatle dolazi glavna motivacija za izradu aplikacije koja će korisnicima omogućiti izradu samostalnih personaliziranih turističkih obilazaka. Korisnicima se nakon kreiranja korisničkog računa pruža mogućnost izrade turističkog obilaska s atrakcijama koje želi posjetiti. Sam turistički obilazak se u potpunosti obavlja preko interaktivne i multimedijske komunikacije na WhatsApp platformi.

Korisnik može kreirati više turističkih obilazaka preko korisničkog sučelja, te u njih dodavati atrakcije koje želi posjetiti. Nakon što korisnik kreira turistički obilazak, može ga započeti skeniranjem QR kôda na web stranici ili slanjem poruke putem WhatsApp platforme na telefonski broj koji im je dostupan nakon registracije. WhatsApp komunikacija je interaktivna, to jest nakon što registrirani korisnik pošalje poruku, odgovor se automatski šalje s obzirom na sadržaj poruke. Korisniku se u prvom dijelu komunikacije šalju svi dostupni turistički obilasci te status obilazaka. U ovom dijelu korisnik odabire obilazak te ovisno o statusu turističkog obilaska može ga započeti, nastaviti ili početi iz početka.

Sam turistički obilazak je osmišljen tako da se korisniku šalju podaci o prvoj atrakciji iz obilaska koju nije posjetio. Korisnik slanjem WhatsApp poruka može pristupiti podacima o lokaciji kao što su Google Maps poveznica na lokaciju, povijesnim informacijama o lokaciji i slično. Nakon što korisnik posjeti atrakciju može pristupiti kvizu koji se sastoji od jednog pitanja. Ako uspješno odgovori na pitanje, korisniku se dodjeljuje bod i može nastaviti na sljedeću lokaciju. Korisnik nije obavezan odgovoriti na pitanje o lokaciji te ga može preskočiti, ali tada mu se ne dodjeljuje bod. Po završetku turističkog obilaska korisnik može započeti obilazak iznova kao i započeti ili nastaviti neki od drugih kreiranih obilazaka.

Kako je prethodno navedeno korisnik može dijeliti svoj turistički obilazak s drugim korisnicima aplikacije. Upravo zato je osmišljen i implementiran sustav bodovanja, jer korisnici mogu putem web stranice pristupiti podacima o ostvarenim bodovima svih korisnika koji dijele taj obilazak. Korisnici putem web stanice mogu uređivati obilaske dodavanjem i brisanjem atrakcija. Ako više korisnika dijeli turistički obilazak kod dodavanja i brisanja atrakcija svim

korisnicima se obilazak ažurira sukladno promjenama. Korisnik također može pobrisati turistički obilazak iz svojih obilazaka čime obilazak briše samo za sebe, ne i za druge korisnike.

Kod dijeljena turističkog obilaska korisnik unosi broj telefona od korisnika s kojim želi podijeliti obilazak. Ako korisnik s tim telefonskim brojem (ne)postoji korisnika se obavještava sukladno tomu. Korisnik kome se dijeli obilazak može ga prihvatiti ili odbiti putem web stranice. Na web stranici su također dostupni podaci o svakoj od atrakcija.

Uloga administratora stranice je upravljanje turističkim atrakcijama i korisnicima putem web korisničkog sučelja *(engl. user interface, UI).* Administrator putem sučelja može dodavati nove atrakcije, brisati postojeće kao i uređivati informacije o atrakcijama. Administrator također može uređivati podatke koji će biti poslani korisniku kod turističkog obilaska. Osim toga može pregledavati pitanja koja se postavljaju korisniku kod pojedine atrakcije. Za svaku atrakciju se može dodavati više pitanja, a za svako pitanje više točnih odgovora. Kod turističkog obilaska grada korisniku se nasumično šalje samo jedno od dostupnih pitanja. Tako kod ponovnih obilazaka ili kod istovremenog obilaska više korisnika postiže se novo i zanimljivije iskustvo.

U prvom dijelu rada opisane se korištene tehnologije neophodne za razvoj aplikacije. Ovdje je opisan njihov način korištenja u izradi aplikacije, karakteristike i praktična primjena. U drugom dijelu opisana je implementacija aplikacije, to jest sama struktura projekta, baze podataka i drugih tehnologija korištenih u izradi. U trećem dijelu nalazi se sam opis aplikacije. Ovaj dio služi kao svojevrstan korisnički priručnik, to jest opisan je kompletan proces koji korisnik mora proći kako bi uspješno koristio aplikaciju. U četvrtom dijelu opisan je proces instalacije aplikacije. Ovdje je detaljno opisano proces pokretanja same aplikacije. U petom i zadnjem dijelu rada donesen je zaključak u kojem su opisani doprinosi rada, praktična primjena kao i predložena unaprjeđenja s postavljenim ciljevima. Također u ovom dijelu se nalazi i popis korištene literature.

2. KORIŠTENE TEHNOLOGIJE

Za razvoj aplikacije korištene su tehnologije (.NET Core programski okvir *(engl. framework),* C#, Blazor, Infobip programsko aplikacijsko sučelje i Microsoft SQL) i programski alati (Visual Studio IDE, Git, Ngrok i Postman) koji će biti detaljno opisani u ovom dijelu.

2.1. C#

C# je objektno orijentirani programski jezik. C# omogućava razvijanje sigurnih aplikacija koje se pokreću u .NETu. C# potječe iz C obitelji programskih jezika te je kao takav poznat korisnicima C, C++, Java i JavaScript programskih jezika. C# ima mnoge značajke koje pomažu kod razvoja aplikacija. *Garbage collection* automatski oslobađa zauzetu memoriju koja je zauzeta od strane nekorištenih objekata. Upravljanje iznimkama *(engl. exception handling)* pruža strukturiran pristup kod otkrivanja grešaka i oporavka sustava. C# također pruža podršku za korištenje lambda izraza *(engl. lambda expressions)* i integriranih jezičnih upita *(engl. language-integrated querry, LINQ)*. Svi C# tipovi, uključujući i primitivne tipove, su naslijeđeni iz jednog korijenskog tipa *object*. Svi tipovi dijele set zajedničkih operacija. Nadalje, C# pruža podršku za referentne tipove kreirane od strane korisnika kao i za vrijednosne tipove podataka. C# klase podataka mogu poprimati parametre i metode drugih klasa što nazivamo polimorfizmom. Objektima izvedenih klasa može se upravljati kao i s objektima bazne klase. Bazni objekti mogu definirati virtualne metode koje izvedene klase mogu *overrideati,* to jest pružiti vlastitu implementaciju za te metode.[1]

2.2. ASP.NET Core

ASP.NET Core je programski okvir otvorenog kôda za kreiranje web aplikacija. ASP.NET u potpunosti ujedinjuje ASP.NET MVC i ASP.NET Web API programske okvire u jedan

programski modul. Korištenje .NETa za klijentsku stranu web aplikacije pruža mogućnosti kao što su:

- pisanje programskog kôda u C# umjesto u JavaScriptu
- kreiranje interaktivnih korisničkih sučelja korištenjem C#
- korištenje modernih posluživačkih platformi poput Dockera
- razvijanje korisničkih sučelja za dostupnost na različitim preglednicima, uključujući i mobilne preglednike

ASP.NET Core također pruža jednostavnu integraciju s programskim okvirima za klijentsku stranu aplikacije kao što su Blazor, Angular, React i Bootstrap. [2]

2.3. Visual Studio

Microsoft visual Studio je integrirano razvojno okruženje *(engl. integrated development environment, IDE)*. Koristi se za razvoj računalnih programa za windows operativni sustav, web stranica i aplikacija. Visual Studio sadrži uređivač kôda koji podržava IntelliSense (komponenta koja predlaže ostatak kôda) kao i usluge refaktoriranje kôda. Za Visual Studio postoji velik broj proširenja koja poboljšavaju osnovne funkcionalnosti na skoro svakom nivou.

Visual Studio podržava niz programskih jezika i omogućava korištenje usluga poput uređivača kôda i *debuggera*. Jezici koji dolaze u osnovnom Visual Studio paketu su C, C++, VB.NET, C#, I F#. Podrška za ostale jezike poput M, Python i Ruby dostupna je instalacijom zasebnih jezičnih servisa.

Microsoft besplatno pruža izdanja "Express" programa Visual Studio. Komercijalna izdanja programa Visual Studio kao i određena starija izdanja korisnicima su dostupna preko Microsoftovog programa DreamSpark. [3]

2.4. Blazor

Blazor je besplatan programski okvir otvorenog kôda koji programerima omogućuje izradu web aplikacija korištenjem C # i HTML programskih jezika.

Blazor poslužiteljske aplikacije su u pozadini pokrenute na ASP.NET jezgri. Udaljeni klijenti ponašaju se kao tanki klijenti *(engl. thin clients)* , što znači da se glavnina opterećenja obrade podataka odvija na poslužitelju. Blazor Server izdan je kao dio .NET jezgre 3.

Blazor WebAssembly je SPA (engl. single-page app) programski okvir za izradu interaktivnih klijentskih aplikacija u .NET programskom okviru. Veličina datoteke koja se preuzima veća je nego kod Blazor poslužiteljske stranice, ali se obrada podataka u cijelosti odvija na klijentskom hardveru. Prednost ove vrste aplikacije je brže vrijeme odziva u odnosu na Blazor Server aplikacije. [4]

Blazor programski okvir je korišten za pisanje cjelokupnog aplikacijskog kôda. Osim osnovnih blazor funkcionalnosti korišten je velik broj paketa koji proširuju set funkcionalnosti Blazora. Implementacija paketa i njihovo upravljanje obavljeno je korištenjem sustava NuGet.

NuGet je sustav upravljanja paketima koji je omogućava korisnicima dijeljenje kôda. Nuget je softver čija je klijentska aplikacija besplatna i otvorenog kôda. NuGet paket je ZIP datoteka koja ima ekstenziju *.nupacl* ili *.nupkg*. Od Visual Studio 2012., Visual Studio i Visual Studio za Mac računala mogu koristiti NuGet pakete. U nastavku će biti opisani najbitniji korišteni NuGet paketi i njihova implementacija. [5]

2.4.1. Entity Framework Core

Entity Framework Core je programski okvir otvorenog kôda koji se nadovezuje na popularni Entity programski okvir te služi kao tehnologija za pristup podacima. Pristup podacima obavlja se korištenjem modela. Model se sastoji od klase i kontekstnog objekta *(engl. context object)* koji predstavlja sesiju s bazom podataka. Kontekstni objekt omogućava spremanje podataka i

izvršavanje upita nad podacima. Glavna svrha korištenja ovog paketa je kreiranje modela koji se odražavaju na Microsoft SQL bazu podataka. Nakon kreiranja modela koriste se Entity Framework migracije koje kreiraju bazu podataka na temelju modela ili ažuriraju postojeće modele na temelju promjena u klasi entiteta. [6]

2.4.2. HttpClient

HttpClient se nalazi unutar *System.Net* NuGet paketa te pruža klasu za slanje HTTP zahtjeva i primanje HTTP odgovora. Instanca HttpClient klase ponaša se kao sesija za slanje HTTP zahtjeva. HttpClient instanca je skup postavki koje su primijenjene na sve zahtjeve instance. Svaka instanca HttpClienta izolira svoje zahtjeve od zahtjeva drugih HttpClient instanci. [7]

2.4.3. RestSharp

Glavni razlog korištenja RestSharp je kreiranje sinkronih i asinkronih poziva prema udaljenim HTTP resursima preko HTTP protokola. Kako i ime nalaže, glavni korisnici ovog paketa su razvojni programeri koji koriste *REST (engl. Representational state transfer)* programska aplikacijska sučelja. RestSharp može poslati zahtjev na bilo koje programsko aplikacijsko sučelje preko HTTP protokola ako postoji jedinstveni identifikator *resursa (engl. Uniform Resource Identifier, URI)* i parametri zahtjeva koji su u skladu s HTTP standardima. Jedan od glavnih izazova korištenja HTTP programskih aplikacijskih sučelja za .NET razvojne programere je korištenje zahtjeva i odgovora različitih vrsta i njihovo prevođenje u C# tipove podataka. RestSharp obavlja funkcije serijalizacije tijela zahtjeva u JSON ili XML format i deserijalizacije odgovora. [8]

2.4.4. Identity

ASP.NET Core Identity je programsko aplikacijsko sučelje koje pruža podršku za sustav registriranja korisnika. Identity također pruža mogućnost upravljanja korisničkim računima, lozinkama, podacima profila, rolama, potvrdama putem elektroničke pošte i slično. Korištenjem Identityja korisnici mogu kreirati korisniče račune pohranjene u sustavu pohrane po izboru. Također Identity pruža mogućnost korištenja nekih od vanjskih sustava za prijavu korisnika kao što su Facebook, Google, Microsoft i Twitter. Potpuni Identity programski kôd je dostupan na GitHubu. Nakon instalacije u projektu se generiraju stranice koje podržavaju Identity operacije. Identity je najčešće konfiguriran korištenjem SQL baze podataka za pohranu imena, lozinki i ostalih podataka o profilu. Osim SQL baze podataka mogu se koristiti i drugi sustavi za pohranu podataka kao što je Azure Table Storage. Nakon promjena nad entitetima ili kod kreiranja novih entiteta, da bi se promjene primijenile na bazu podataka potrebno je kreirati migraciju. Migracije se kreiraju i izvršavaju korištenjem naredbi u konzoli za upravljanje paketima *(engl. package manager console)*. Nakon uspješnog izvršavanja migracije baza podataka će se ažurirati sukladno izmjenama i biti spremna za korištenje. [9]

2.5. Ngrok

Ngrok pruža niz drugih poput pokretanja osobnog cloud servisa, prikazivanje web stranice na internetu izbjegavajući NAT protokole i vatrozide preko sigurnih tunela, kreiranje *webhook* usluge, testiranje lokalno povezanih mobilnih kao i pružanje usluge stabilnih adresa za spojene uređaje.

Ngrok automatski kreira javni jedinstveni HTTPS *uri* za web stranicu koja je pokrenuta lokalno. Ngrok se također brine za sigurnost podataka korištenjem protokola transportnog sloja što korisniku olakšava konfiguriranje sigurnosnih postavki. Autentikacija je ostvarena korištenjem autentikacijskih podataka kako bi se ograničio pristup korisničkim tunelima. Ngrok također podržava i druge autentikacijske metode kao što su Oauth. Ngrok je dostupan u više

regija i zona dostupnosti u cijelom svijetu, osiguravajući da su korisnikovi tuneli uvijek dostupni. Ngrok radi u svim uvjetima, čak i kada uređaj prelazi iz jedne mreže u drugu.

Korisnik može korištenjem internetskog sučelja pregledavati sve HTTP zahtjeve i odgovore preko zadanog tunela. Korištenjem ngrok *url* adrese korisnik može konfigurirati *webhook* uslugu. Korisnik također ima mogućnost pokretanja više simultanih tunela korištenjem jednog Ngrok klijenta. Korisnički računi mogu međusobno dijeliti pristup domenama i adresama, što omogućava razvojnim programerima suradnju na projektu, pri čemu svi koriste svoje autentikacijske podatke. Korištenjem TCP tunela bilo koji mrežni servis se može izložiti internetu, uključujući one koji je koriste HTTP protokole. Korištenjem Ngroka korisnik može upravljati pristupima bilo kojem TCP servisu.

Ngrok također pruža podršku za internetske utičnice *(engl. socket)*. Ngrok prosljeđuje konekcije ostvarene preko internetskih utičnica korištenjem HTTP tunela bez ikakvih promjena u podacima. Ngrok, osim za HTTP, pruža podršku i za tunele preko HTTPS protokola. Ova usluga je korisna za dijeljenje web servera, prikazivanje stranica u razvoju ili dijeljenje pristupa programskim aplikacijskim sučeljima.

Ngrok pruža internetsko korisničko sučelje gdje korisnik može pregledavati sve HTTP zahtjeve koji su poslani preko tunela. Nakon pokretanja Ngroka u pregledniku je dovoljno otvoriti adresu http://localhost:4040/ da bi se pregledali svi poslani zahtjevi. Nakon što se zahtjev pošalje na javni URL on je automatski vidljiv u korisničkom sučelju. Ovdje korisnik može vidjeti sve detalje zahtjeva uključujući vrijeme, trajanje, zaglavlje zahtjeva, parametre upita, sadržaj upita kao i odgovor na zahtjev.

Korištenje *webhookova* izdanih od strane vanjskih programskih aplikacijskih sučelja može usporiti razvojni ciklus aplikacije radi često složenih oblika zahtjeva, poput telefonskog poziva, da bi se generirao *webhook* zahtjev. Ngrok omogućava ponovno slanje odgovora na zahtjev jednim klikom što značajno ubrzava proces testiranja. [10]

2.6. Git

Git je softver za praćenje promjena u bilo kojem datotečnom sustavu, najčešće korišten za koordiniranje rada između programera koji zajedno rade na razvijanju softvera.

Kao i kod ostalih sustava distribuirane kontrole verzije *(engl. distributed version control)*, svaki Git direktorij na svakom računalu je repozitorij koji sadrži čitavu povijest izmjena nad repozitorijem. [11]

GitHub je pružatelj internetskih usluga poslužitelja *(engl. hosting provider)* za razvoj softvera i kontrolu verzije korištenjem Gita. GitHub pruža sustav kontrole verzija i funkcionalnosti upravljanja kôdom. Također GitHub pruža usluge kontrole pristupa, otkrivanja grešaka i dokumentacije za svaki projekt.

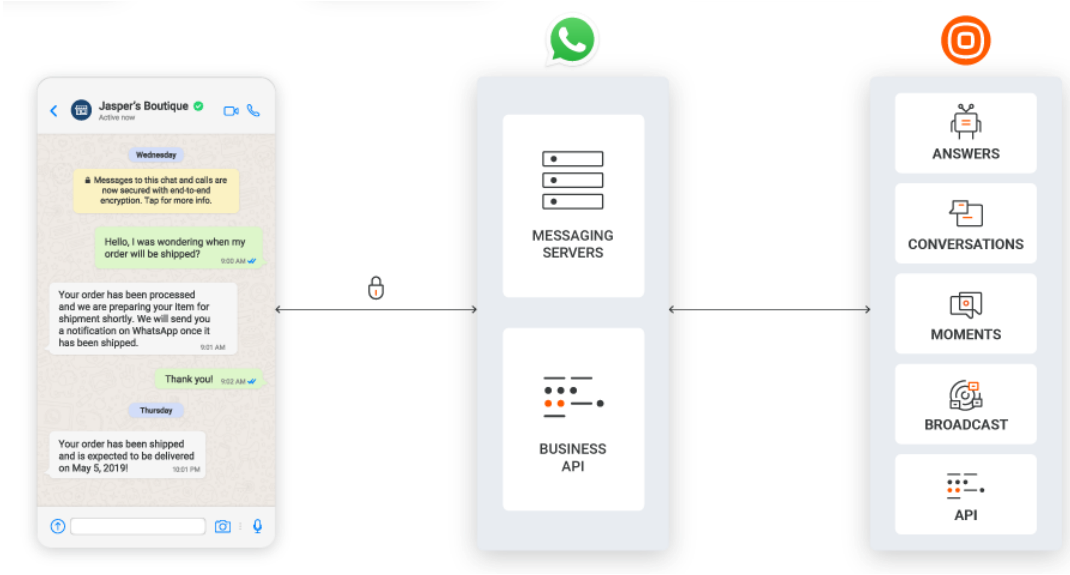
Najčešća uporaba GitHuba je kod projekata otvorenog kôda. Projektima na GitHubu se može pristupiti i upravljati korištenjem standardnog Git sučelja naredbene linije *(engl. command line interface)* u kojem su dostupne sve standardne Git naredbe. Osim toga korisnici mogu pregledavati sve javno dostupne repozitorije na stranici. Stranica pruža usluge slične društvenim mrežama poput novosti, pratitelja, dokumentacije i dijeljenih repozitorija.

Bilo tko može pregledavati i preuzimati javno dostupne repozitorije, ali samo registrirani korisnici mogu kontribuirati sadžajem u repozitorij. Korisnici nakon registracije mogu voditi diskusije, upravljati repozitorijima, raditi izmjene na repozitorijima drugih korisnika i recenzirati promjene napravljene na kôdu. Prethodno su samo javni repozitoriji bili besplatni ali od travnja 2020. GitHub je omogućio svim korisnicima beskonačan broj besplatnih privatnih repozitorija. Glavna svrha korištenja GitHuba je omogućiti korisnicima sustav kontrole verzije kôda i otkrivanja problema kod razvoja softvera.

Za kontrolu verzije Git dozvoljava zahtjeve za povlačenjem (engl. pull request) kako bi ostali korisnici mogli preuzeti izvorni kôd i raditi izmjene na njemu. Korisnici imaju mogućnost pregledavanja predloženih promjena i mogu vidjeti detaljno koje su promjene napravljene i verificirati ih. Povijest svih promjena se sprema i može se vidjeti u bilo kojem trenutku. [12]

2.7. Infobip API

WhatsApp je jednostavan, siguran i pouzdan kanal za srednje i velike projekte koji žele doći do svojih korisnika bilo gdje u svijetu. Očekivanja od projekata i korisnika su u moderno vrijeme evoluirala. Današnji korisnici očekuju imati mogućnost komuniciranja direktno s kreatorima projekta/aplikacije, kao i automatiziranu korisničku podršku u stvarnom vremenu. Na slici 1 prikazan je model komuniciranja WhatsApp aplikacije s Infobip programskim aplikacijskim sučeljem.



Slika 1. Infobip WhatsApp model komunikacije

Korištenjem Infobip programskog aplikacijskog sučelja može upravljati svim WhatsApp komunikacijama direktno kroz Infobipovu platformu. Korisničko sučelje pruža mogućnost definiranja *webhook* adrese na koju se prosljeđujuWhatsApp poruke u obliku zahtjeva. Nakon što programsko aplikacijsko sučelje zaprimi WhatsApp poruku, ona se prosljeđuje na *webhook* adresu u obliku POST zahtjeva.

WhatsApp programsko aplikacijsko sučelje pruža niz mogućnosti za WhatsApp aplikacijske klijente poput Android, iOS, Web i drugih platformi. Sama aplikacija se nalazi na serveru, a klijent može koristiti programsko aplikacijsko sučelje za slanje i primanje poruka. [13]

Infobip programsko aplikacijsko sučelje pruža usluge prikupljanja i verifikacije prikupljenih medijskih datoteka kroz razgovor s korisnicima, uključujući sike, video zapise, datoteke i druge oblike podataka.

Programsko aplikacijsko sučelje također nudi i mogućnost brzog dijeljenja kontakata preko WhatsApp aplikacije. Korisnici također mogu podijeliti svoju trenutnu lokaciju ako su aplikaciji potrebi takvi podaci; primjer: predviđeno vrijeme isporuke poruke. [14]

2.8. Microsoft SQL Server

Microsoft SQL Server je sustav za upravljanje bazom podataka koje je razvio Microsoft. Glavna zadaća Microsoft SQL Servera je spremanje i dohvaćanje podataka prema zahtjevima drugih softverskih aplikacija koje mogu biti pokrenute na istom računalu ili na drugom računalu na mreži. Microsoft pruža nekoliko različitih verzija Microsoft SQL Servera namijenjenih različitoj skupini korisnika kao i različitim opsezima posla.

SQL Server Management Studio je grafičko korisničko sučelje za konfiguriranje, upravljanje i administriranje svih komponenti uključenih u Microsoft SQL Server. Ovaj alat uključuje uređivače skripti *(engl. script editors)* i grafičke alate za rad sa serverom. SQL Server Management Studio je zamijenio Enterprise Manager kao primarno sučelje za upravljanje Microsoft SQL serverom od 2005. Godine.

Glavna značajka SQL Server Management Studija je pretraživač objekata *(engl. object explorer),* koji omogućava korisniku da pretražuje, odabire, i izvršava radnje nad objektima pohranjenim na serveru. SQL Server Management Studio se može koristiti za kreiranje nove baze podataka kao i za ažuriranje postojećih shema baze podataka dodavanjem ili modificiranjem tablica. Uključuje i prozore za upit *(engl. query windows)* koji sadrže grafičko korisničko sučelje za pisanje i pokretanje upita. [15]

2.9. Bootstrap

Bootstrap je besplatan programski okvir otvorenog kôda usmjeren responzivnom front-end razvoju web stranica. Sadrži CSS i JavaScript predloške za tipografiju, obrasce, botune, navigaciju i ostale komponente sučelja.

Bootstrap je HTML, CSS i JavaScript biblioteka za koju je glavni motiv korištenja pojednostavljenje razvoja klijentske strane web stranica. Glavni cilj dodavanja Bootstrapa u web projekt je korištenje njegovih opcija za boje, veličine, font i raspored elemenata na stranici. Nakon dodavanja u projekt, Bootstrap pruža osnovne stilove za sve HTML elemente. Rezultat je uniforman izgled teksta, tablica i elementa obrazaca u svim internetskim preglednicima. Dodatno, programeri mogu koristiti CSS klase definirane u Bootstrapu za daljnje prilagođavanje izgleda njihovom sadržaju. Bootstrap također dolazi s nekoliko JavaScript komponenti u formi priključaka *(engl. plugins)*. Oni pružaju dodatne elemente za korisničko sučelje poput razgovornih okvira, oblačića za savjete i karusel. Svaka Bootstrap komponenta se sastoji od HTML strukture, CSS deklaracije i u nekim slučajevima JavaScript kôda. Komponente također proširuju funkcionalnost postojećih elemenata sučelja, uključujući automatsko dovršavanje teksta kod polja za unosi slično.

Najistaknutije komponente za Bootstrap su komponente za raspored, jer se one apliciraju na čitavu web stranicu. Osnovni element rasporeda se naziva kontejner *(engl. container)*, a svi ostali elementi su smješteni unutar njega. Programeri mogu birati između kontejnera određene veličine ili kontejnera prilagodive veličine. Nakon što je kontejner postavljen, ostale komponente su implementirane kroz sustav redova i stupaca unutar njega. [16]

2.10. Postman

Postman je klijent programskog aplikacijskog sučelja koji olakšava process kreiranja, dijeljenja, testiranja i dokumentiranja programskih aplikacijskih sučelja. Postman korisniku dozvoljava kreiranje i spremanje jednostavnih i složenih HTTP/s zahtjeva. Postman je korišten

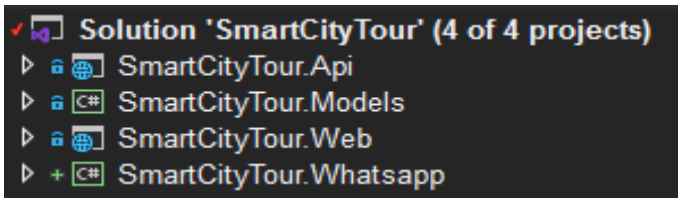
kroz čitav proces izrade ovog rada radi testiranja komunikacije programskog aplikacijskog sučelja s bazom podataka i verifikacijom podataka koji su dobavljeni kod raznih oblika HTTP zahtjeva. [17]

3. IMPLEMENTACIJA APLIKACIJE

U ovom poglavlju je detaljan opis strukture projekta, baze podataka i konfiguracije ostalih programskih alata koji su korišteni za razvoj aplikacije.

3.1. Struktura projekta

Sam projekt je strukturiran od 4 dijela Programskog aplikacijskog sučelja SmartCityTour.Api, klijentske aplikacije SmartCityTour.Web, konzolne aplikacije preko koje se odvija sva WhatsApp komunikacija SmartCityTour.WhatsApp i biblioteke klasa SmartCityTour.Models gdje su pohranjeni modeli potrebni za funkcioniranje aplikacije. Na slici 2 prikazana je struktura projekta unutar Visual Studia. U nastavku će biti opisan svaki od dijelova aplikacije uključujući alate korištene izvan Visual Studia.

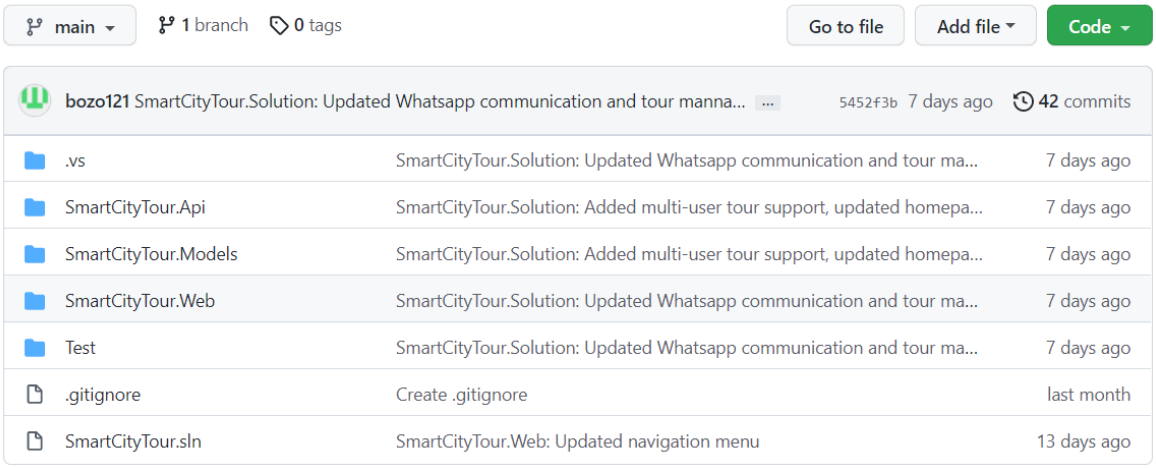


Slika 2. Struktura projekta SmartCityTour

3.2. Git

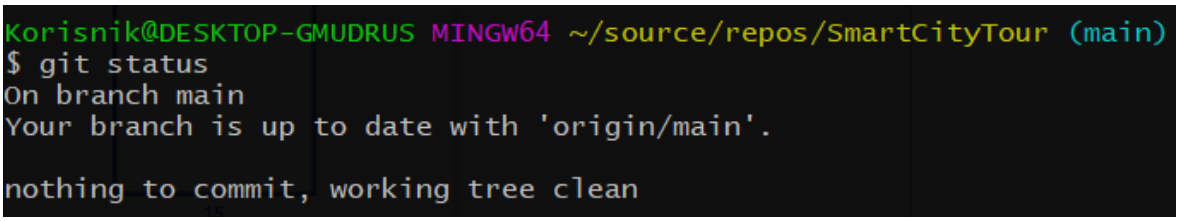
Tijekom čitavog razvojnog procesa korišten je Git sustav kontrole verzije. Git je korišten preko Git Bash konzole a svaka izmjena je povezana s GitHub repozitorijem te se automatski ažurira. Tako se postigla javna dostupnost kôda, te mogućnost povratka na posljednju stabilnu

verziju aplikacije ako dođe problema kod razvoja softvera. Na slici 3 je prikaz repozitorija u GitHub korisničkom sučelju.



Slika 3. Prikaz projekta u GitHub repozitoriju

Kod svake nove veće promjene u pravilu se kreira *Git commit* koji sadrži sve datoteke uključene u promjenu. Proces kreiranja *commita* u pravilu ide redoslijedom dodavanje datoteka, kreiranje *commita* sa smislenom porukom, te se na kraju radi *push* koji lokalne promjene primjenjuje na udaljeni Git repozitorij. Status lokalnog repozitorija u odnosu na originalni repozitorij u svakom trenutku se može provjeriti naredbom "git status" prikazanoj na slici 4.



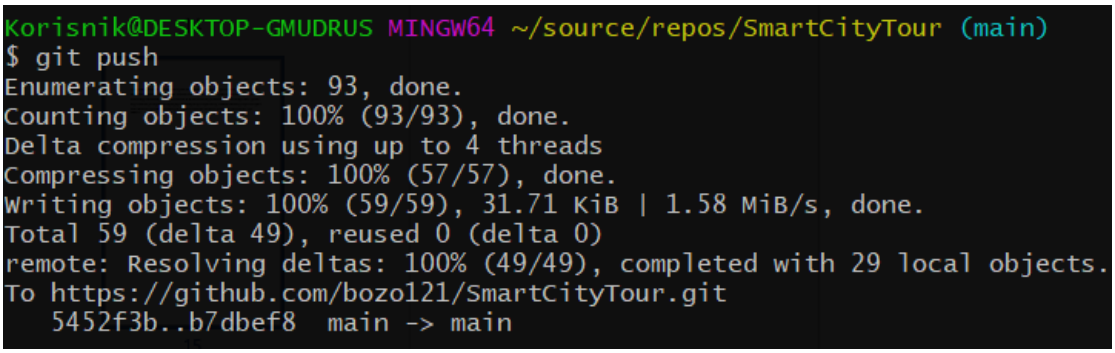
Slika 4. Primjer korištenja "git status" naredbe

Vrlo je bitno da se kod kreiranja commita doda smislena poruka kako bi budući razvojni programeri koji će raditi na aplikaciji lakše pratili tijek razvoja i bolje razumjeli sadržaj svakog pojedinačnog *commita*. Primjer kreiranja *commita* je prikazan u ispisu 1.

```
$ git commit -m "SmartCityTour.Web: Updated web services to web clients"
```

Ispis 1. Primjer kreiranja git commita

Nakon kreiranja *commita* "git push" naredbom ažuriramo udaljeni Git repozitorij u skladu s promjenama napravljenim lokalno. Primjer korištenja "git push" naredbe prikazan je na slici 5.



Slika 5. Primjer korištenja "git push" naredbe

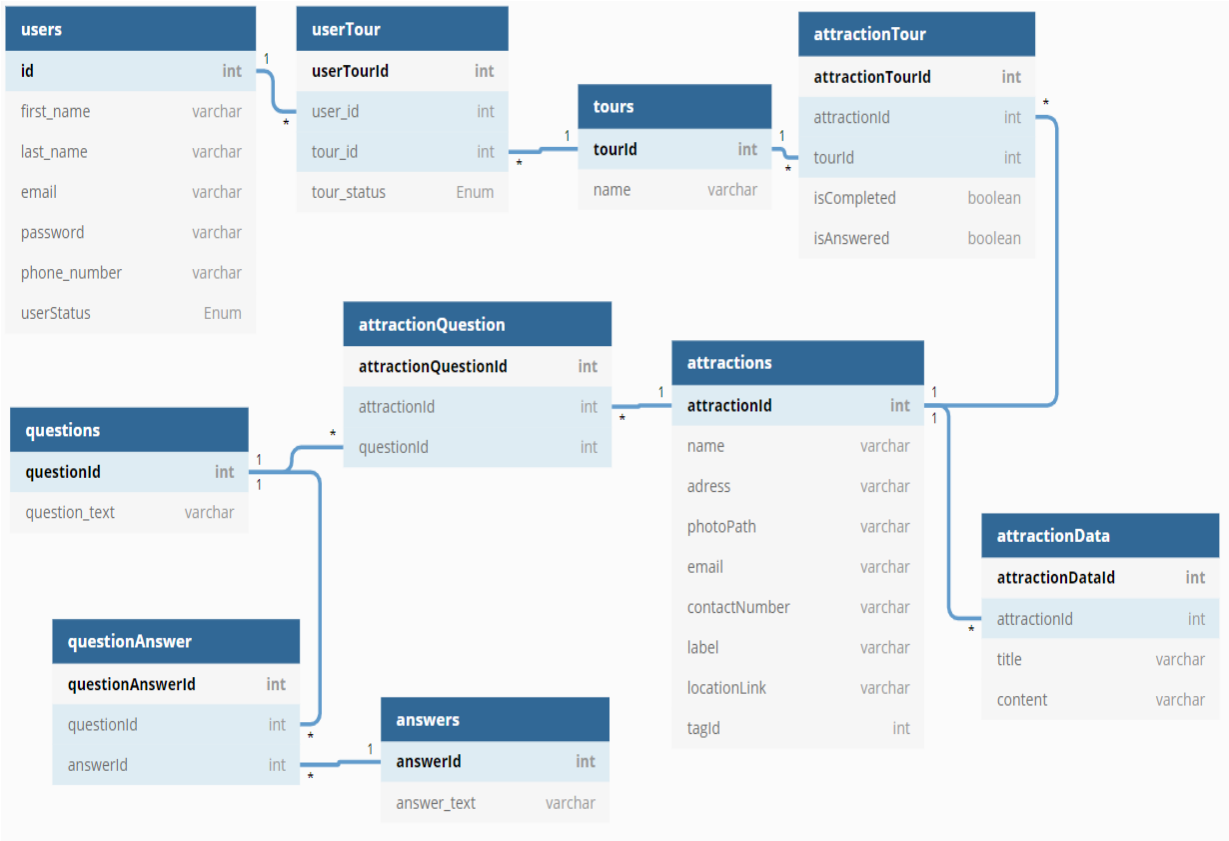
Čitava povijest svih commitova može se vidjeti korištenjem "git log" naredbe. U dobivenom prikazu može se kronološki vidjeti lista svih commitova, autor, datum kada je *commit pushan*, poruka *commita* i jedinstveni identifikator *commita*. Korisnik se korištenjem identifikatora može vratiti na bilo koju prošlu verziju aplikacije. Na slici 6 prikazana je jedan od *commitova* iz liste dobivene korištenjem "git log" naredbe.

```
commit 0b816bad1fd60e797607183cd9a48dc2bc401d21
Author: bozo121 <bozo121@live.com>
Date:   Fri Aug 13 12:27:51 2021 +0200

    SmartCityTour.Web: Added placeholders for registration page
```

Slika 6. Primjer ispisa "git log" naredbe

3.3. Microsoft SQL baza podataka

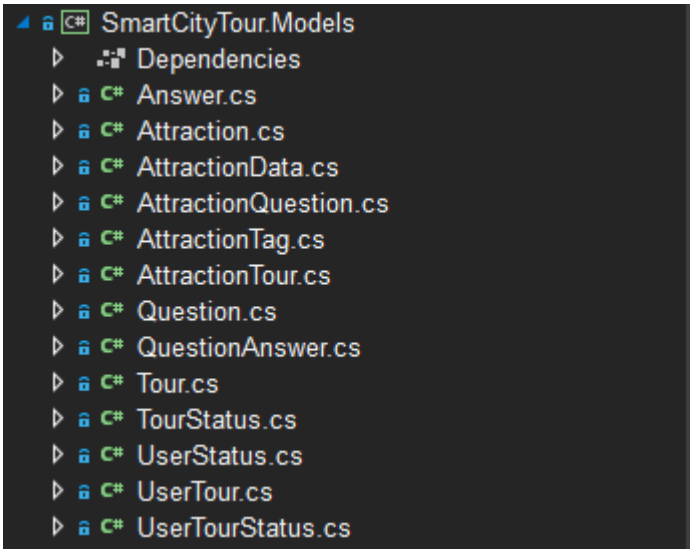
Za bazu podataka korištena je Microsoft SQL baza podataka. Na slici 7 prikazana je shema baze podataka. Sve operacije nad bazom podataka rađene su korištenjem migracija tako da će detaljniji opis komunikacije s bazom podataka biti opisan u dijelu programskog aplikacijskog sučelja SmartCityTour.Api.



Slika 7. Shema baze podataka korištene u aplikaciji

3.4. SmartCityTour.Models

SmartCityTour.Models je biblioteka klasa u kojoj su pohranjene sve klase korištene u aplikaciji. Na slici 8 prikazana je struktura SmartCityTour.Models biblioteke, a u ispisu 2 je primjer jedne od definiranih klasa unutar biblioteke.



Slika 8. Shema baze podataka korištene u aplikaciji

```
public class AttractionData
{
    public int AttractionDataId { get; set; }
    public int AttractionId { get; set; }
    public string DataTitle { get; set; }
    public string DataContent { get; set; }
}
```

Ispis 2. Primjer jedne klase podataka u biblioteci

U klasi su definirana svojstva klase *(engl. property)*, koja se odražavaju na stupce unutar baze podataka. Na slici 9 nalazi se prikaz tablice unutar baze podataka kreirane na temelju zadanog entiteta. Nad svojstvima klase možemo definirati anotacije poput *"Required"* i *"MinLenght"* što ograničava oblik podataka koji se može unijeti u pripadajuću tablicu unutar baze podataka. ASP.NET Core nam također omogućava da unutar klase definiramo navigacijska svojstva *(engl. navigation properties)* koja će nam olakšati pristup podacima kod entiteta koji se nalaze u relaciji s drugim entitetima. Navigacijska svojstva nisu pohranjena unutar baze podataka. Dovoljno je samo pohraniti *id* drugog entiteta u relaciji. U ispisu 3 prikazan je pristup podacima korištenjem navigacijskog svojstva.

	AttractionDataId	AttractionId	DataTitle	DataContent
1	4	1	General	The Church of St Donatus is a church located in Za...
2	5	2	General	Although partially overrun by buildings, Zadar's foru...
3	6	2	A planned city - urbs quadrata	In order to make the city look truly Roman, imperial s...
4	7	2	Today's layout	In the period of late antiquity, the foundations of Chri...
5	8	2	Our hint...	On the west side of the Capitolio you may notice a pl...
6	9	5	General	Another project by the award-wining architect Nikola...
7	10	21	History	The street older than the city, once the Via Magna, ...

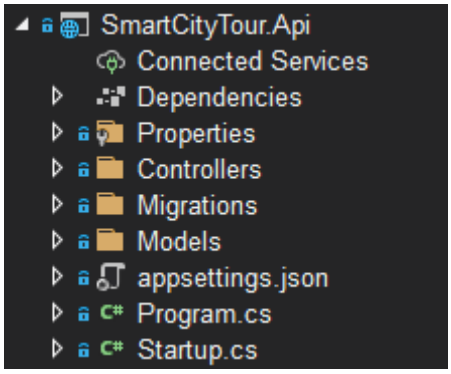
Slika 9. Primjer spremanja klase unutar baze podataka

```
AttractionTour attractionTour = new AttractionTour();
attractionTour.Attraction.Label = "some text";
```

Ispis 3. Primjer spremanja klase unutar baze podataka

3.5. SmartCityTour.Api

SmartCityTour.Api je programsko aplikacijsko sučelje čija je glavna namjena komuniciranje s bazom podataka. SmartCityTour.Api obavlja sve operacije dohvaćanja, ažuriranja i brisanja podataka iz baze podataka. Sama struktura projekta prikazana je na slici 10.



Slika 10. Struktura projekta SmartCityTour.Api

Unutar *Models* direktorija nalazi se AppDbContext klasa i kontroleri. AppDbContext klasa služi kao klasa za operacije nad bazom podataka. Prvi korak postavljanja SmartCityTour.Api projekta je postavljanje kontekstne klase za bazu podataka. To se radi tako da unutar appsettings.json datoteke definiramo *connection string* što je prikazano u ispisu 4. On nam služi za ostvarivanje konekcije s bazom podataka kod instanciranja kontekstnog objekta baze podataka što je prikazano u ispisu 5.

```
"ConnectionStrings": {
"DBConnection":"Data Source=DESKTOP-GMUDRUS\\SQLEXPRESS;Initial
Catalog=SmartCityTour;Integrated Security=True"
```

Ispis 4. Primjer konfiguriranja *connection stringa*

```
services.AddDbContext<AppDbContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString
("DBConnection")));
```

Ispis 5. Primjer instanciranja kontekstne klase za bazu podataka

Entity Framework Core omogućava korištenje definiranih servisa kroz cijelu aplikaciju. Kontekstna klasa za bazu podataka se definira u *startup.cs* i dodaje se u servise (engl. services) pri čemu je potreban *connection string* i klasa koja je naslijeđena iz DbContext klase unutar Entity Framework Core. U našem slučaju ta klasa je AppDbContext. Na ispisu 6 prikazana je AppDbContext klasa.

```
public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> options)
        : base(options)
        {
        }
        public DbSet<Attraction> Attractions { get; set; }
        public DbSet<AttractionTag> AttractionTags { get; set; }

       protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            //Seed AttractionTags Table
            modelBuilder.Entity<AttractionTag>().HasData(
               new AttractionTag
               {
                   AttractionTagId = 1,
                   AttractionTagName = "Attraction",
                   PictureUrl = "",
               });
        }

      }
```

Ispis 6. Prikaz AppDbContext klase

Unutar AppDbContext klase potrebno je definirati sve klase koje želimo pohraniti kao entitete u bazu podataka. Unutar klase također možemo napraviti *override OnModelCreating* metode što nam omogućava populiranje baze podataka sa željenim podacima kod kreiranja entiteta. Nakon što smo definirali sve klase podataka koje želimo spremiti unutar baze podataka, ili ako smo ažurirali postojeće klase potrebno je kreirati Entity Framework Core migraciju putem konzole za upravljanje paketima. Nakon što je migracija kreirana da bi se promjene odrazile na bazu podataka potrebno je izvršiti update-database naredbu. Sve napravljene migracije spremaju se unutar direktorija Migrations. Primjer kreiranja migracije prikazan je u ispisu 7. Slično kao i kod kreiranja commitova, kod kreiranja migracije također je potrebno dati smisleno ime migraciji.

```
PM> add-migration AddedAttractionsEntity
PM> Update-Database
```

Ispis 7. Primjer kreiranja migracije

Unutar *Models* direktorija definirani su i repozitoriji. Uloga repozitorija je obavljanje operacija nad bazom podataka korištenjem implementacije AppDbContext klase. Primjer jedne klase repozitorija prikazana je u ispisu 8.

```
public class AttractionRepository : IAttractionRepository
    {
        private readonly AppDbContext appDbContext;

        public AttractionRepository(AppDbContext appDbContext)
        {
            this.appDbContext = appDbContext;
        }
        public async Task<Attraction> AddAttraction(Attraction
attraction)
        {
            var result = await
appDbContext.Attractions.AddAsync(attraction);
            await appDbContext.SaveChangesAsync();
            return result.Entity;
        }
```

Ispis 8. Primjer klase repozitorija

AppDbContext klasa uzima implementaciju iz *startup* klase. U repozitoriju su najčešće definirane CRUD (engl. create, read, update, delete) operacije nad entitetima, ali mogu se definirati i ostale operacije po želji korisnika. U ispisu 7 nalazi se primjer metode za dodavanje atrakcije u pripadajuću tablicu u bazi podataka. Pošto su operacije nad bazom podataka asinkrone, to jest potrebno je vrijeme za njihovo izvršavanje, potrebno je čekati da se takva operacija izvrši što je u kôdu označeno s *await*. Slično kao i kod kontekstne klase za bazu podataka, repozitorijima također dajemo implementaciju unutar startup.cs klase što je prikazano u ispisu 9.

```
services.AddScoped<IAttractionRepository, AttractionRepository>();
services.AddScoped<IAttractionTagRepository, AttractionTagRepository>();
```

Ispis 9. Primjer instanciranja repozitorija

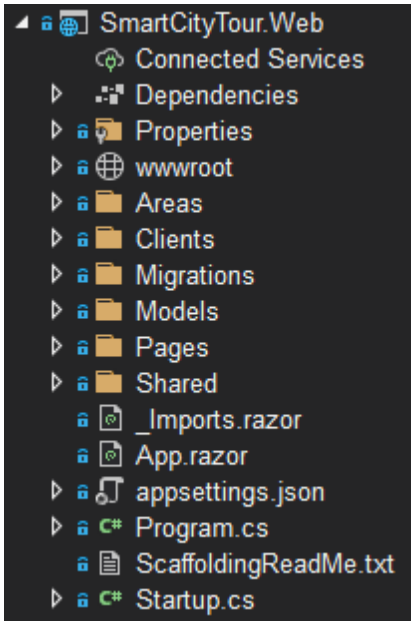
Unutar *Controllers* direktorija nalaze se kontroleri. Uloga kontrolera je definirati koje se operacije nad podacima unutar baze podataka izvršavaju kod pojedinog oblika HTTP zahtjeva. Primjer klase kontrolera nalazi se u ispisu 10. Kontroleri obavljaju operacije nad bazom podataka korištenjem prethodno opisanih repozitorija. Anotacijom klase *Route* definiramo *uri* za ovaj kontroler koji je sačinjen i od imena kontrolera pa je njegov oblik *"api/answers"*. U primjeru s ispisa 10 definirana je jedna metoda za dohvaćanje podataka iz baze *GetAnswers*. Metoda će se izvršiti kada se pošalje HTTP GET zahtjev na prethodno definirani *endpoint*. Ako je dohvaćanje podataka bilo uspješno podaci se vraćaju u *JSON* formatu, a ako se pojavi greška kod dohvaćanja podataka generira se iznimka. Klijentska strana aplikacije pristupa podacima isključivo generiranjem HTTP zahtjeva.

```
    [Route("api/[controller]")]
    [ApiController]
    public class AnswersController : ControllerBase
    {
        private readonly IAnswerRepository answerRepository;

        public AnswersController(IAnswerRepository answerRepository)
        {
            this.answerRepository = answerRepository;
        }

        [HttpGet]
        public async Task<ActionResult> GetAnswers()
        {
            try
            {
                return Ok(await answerRepository.GetAnswers());
            }
            catch (Exception)
            {
                return
StatusCode(StatusCodes.Status500InternalServerError,
                    "Error retrieving data from the database");
            }
        }
```

Ispis 10. Prikaz klase kontrolera

3.6. SmartCityTour.Web

SmartCityTour.Web je klijentska aplikacija u kojoj je kreirana web stranica i korisničko sučelje. Struktura aplikacije prikazana je na slici 11.



Slika 11. Prikaz strukture projekta SmartCityTour.Web

Kako je ranije spomenuto klijentska aplikacija koristi HTTP zahtjeve kako bi preko programskog aplikacijskog sučelja SmartCityTour.Api pristupala podacima iz baze podataka. Unutar *Clients* direktorija definirane su klijentske klase koje generiraju HTTP zahtjeve prema programskim aplikacijskom sučelju. Klijentske klase također je potrebno definirati unutar servisa u startup.cs klasi kako bi bili dostupni kroz čitavu aplikaciju. Na ispisu 11 nalazi se primjer klijentske aplikacije, a na ispisu 12 primjer dodavanja klijentske aplikacije u servise.

```
public class AnswerClient : IAnswerClient
    {
        private readonly HttpClient httpClient;

        public AnswerClient(HttpClient httpClient)
        {
            this.httpClient = httpClient;
        }
        public async Task<Answer> AddAnswer(Answer newAnswer)
        {
            return await httpClient.PostJsonAsync<Answer>("api/answers",
newAnswer);
        }
    }
```

Ispis 11. Prikaz klase klijenta

```
services.AddHttpClient<IAnswerClient, AnswerClient>(client =>
            {
                client.BaseAddress = new Uri("https://localhost:44379/");
            });
```

Ispis 12. Prikaz dodavanja klijentske klase u servise

Kod dodavanja klijentske klase u servise pružamo i *BaseAdress uri* za *HttpClient* kojeg ćemo koristiti za slanje HTTP zahtjeva. U ispisu 11 definirana je metoda za dodavanje novog *Answer* objekta u bazu podataka. Kod pozivanja ove metode dodaje se *Answer* objekt u tijelo zahtjeva u *JSON* formatu, te se šalje POST zahtjev na adresu https://localhost:44379/api/answers.

Blazor web stranice su *Razor* komponente koje se sastoje od HTML dijela, koji je ustvari korisničko sučelje, i C# dijela u kojem se nalazi kôd stranice. U procesu izrade aplikacije korišten je *Base* pristup komponentama, to jest HTML dio i C# dio podijeljeni su u dvije različite datoteke (primjer: *ListAricles.razor* za HTML dio kôda i *ListAticlesBase.cs* za C# dio kôd) . Primjer jedne Base klase nalazi se u ispisu 13.

```
public class AttractionsListBase : ComponentBase
    {
        [Inject]
        public IAttractionClient AttractionClient { get; set; }

        public IEnumerable<Attraction> Attractions { get; set; }

        protected override async Task OnInitializedAsync()
        {
            Attractions = (await
AttractionClient.GetAttractions()).ToList();
        }
    }
```

Ispis 13. Prikaz dodavanja klijentske klase u servise

Pošto smo ranije definirali klijentske klase i dodali ih u servise, ASP.NET Core nam dozvoljava korištenje [Inject] anotacije kako bismo mogli koristiti klijentsku klasu unutar *Base* klase. Također u ispisu 13 možemo vidjeti *overrideanje* metode *OnInitialisedAsync*. Ova metoda se poziva kod inicijaliziranja stranice, najčešće kod otvaranja ili osvježavanja stranice. Unutar *OnInitialisedAsync* metode populiramo listu atrakcija koristeći klijentsku klasu, te nakon toga podatke iz liste možemo koristiti na stranici.

U *Razor* komponenti potrebno je definirati Base klasu koja se koristi korištenjem *@inherits* direktive i url na kojem će stranica biti dostupna korištenjem *@page* direktive kao što je prikazano u ispisu 14. Kako bi koristili elemente Base klase u Razor komponenti dovoljno je koristiti simbol @ kao što je prikazano u ispisu 15.

```
@page "/attractionslist"
@inherits
```

Ispis 14. Primjer korištenja @page i @inherits directive

```
<img style="height:250px; width:400px;" src="@attraction.PhotoPath" />
```

Ispis 15. Primjer korištenja elementa iz Base klase

Sustav autorizacije i autentikacije uspostavljen je korištenjem *Identity* programskog okvira. *Identity* programski okvir implementira se u aplikaciju *scaffoldanjem* čime se u projekt dodaju sve potrebne stranice i servisi za njegovo funkcioniranje. Kod postavljanja kontekstne klase za bazu podataka u *startup.cs* klasi potrebno je definirati klasu korisnika koja se koristi kao što je prikazano u ispisu 16. Identity programski okvir pruža već zadanu klasu *IdentityUser*, no zbog limitiranog seta atributa u projektu je ta klasa proširena ka što je prikazano u ispisu 17. Za pristup korisničkim podacima nećemo koristiti klijentske klase kao za ostale entitete nego ćemo im pristupati koristeći *UserMannager* klasu i *IHttpContextAccessor* sučelje koji se nalaze unutar ASP.NET Core programskog okvira kao što je prikazano na ispisu 18. Ako želimo ograničiti pristup korisnicima na stranici koji nemaju korisnički račun ASP.NET Core nam pruža mogućnost korištenja anotacije <*AuthorizeView*> čiji su elementi vidljivi samo prijavljenim korisnicima.

```
services.AddDefaultIdentity<ApplicationUser>(options =>
options.SignIn.RequireConfirmedAccount = true)
```

Ispis 16. Prikaz definiranja korisnika u startup.cs

```
public class ApplicationUser : IdentityUser
    {
        public String firstName { get; set; }
        public String lastName { get; set; }
        public UserStatus userStatus { get; set;
}
```

Ispis 17. Prikaz proširenja *IdentityUser* klase

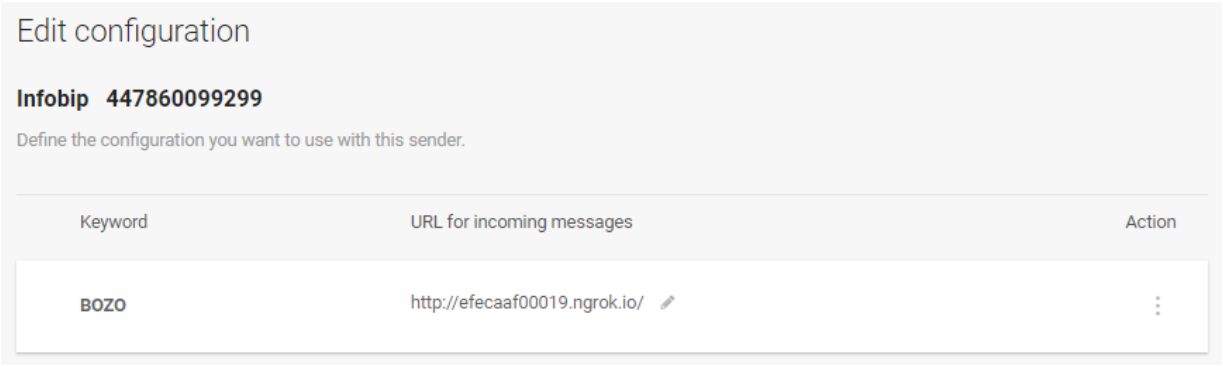
```
user = await userManager.GetUserAsync(HttpContextAccessor.HttpContext.User);
```

Ispis 18. Primjer dohvaćanja prijavljenog korisnika

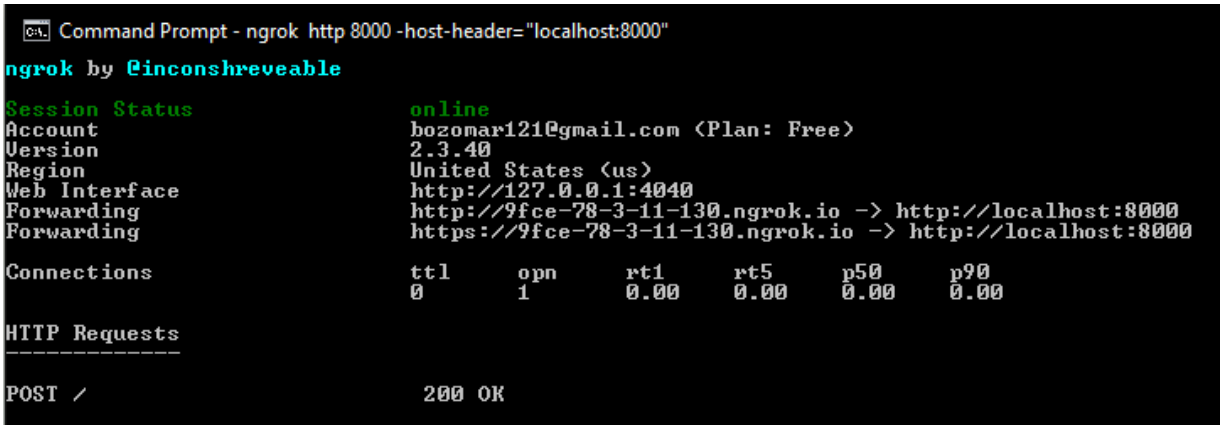
3.7. SmartCityTour.Whatsapp

SmartCityTour.WhatsApp je konzolna aplikacija preko koje se odvija WhatsApp komunikacija. Međutim, za WhatsApp komunikaciju ne koristi se samo konzolna aplikacija nego i Infobip programsko aplikacijsko sučelje i Ngrok. Prvi korak u ostvarivanju komunikacije je konfiguriranje Infobip programskog aplikacijskog sučelja.

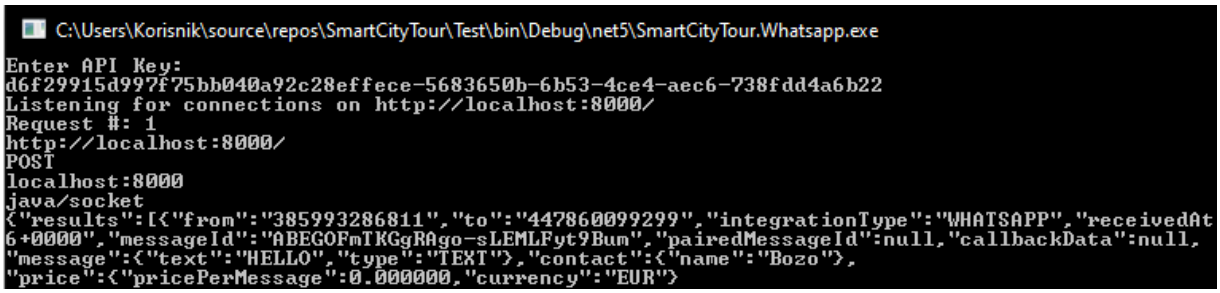
Infobip programsko aplikacijsko sučelje nam nakon registracije nudi javni telefonski broj preko kojeg se može vršiti komunikacija. Komunikacija preko dobivenog telefonskog broja je uvijek aktivna te prima poruke, no kako bi te poruke mogli primati lokalno i na temelju njih slati odgovor potrebno je definirati *webhook* uslugu. Kada se pošalje WhatsApp poruka na dobiveni broj, Infobip programsko aplikacijsko sučelje prosljeđuje poruku na javno dostupnu *webhook* adresu u obliku zahtjeva. *Webhook* adresa za pojedini telefonski broj definira se u Infobip korisničkom sučelju prikazanom na slici 12. Za *webhook* servis korišten je Ngrok. Ngrok nam omogućava prosljeđivanje zahtjeva s javno dostupne *webhook* adrese na lokalni http port (u našem slučaju *port* 8000). Primjer prosljeđivanja prometa koristeći *Ngrok* prikazan je na slici 13. Naša konzolna aplikacija otvara HTTP server na istom *portu* te osluškuje sav nadolazeći promet kao što je prikazano na slici 14. Konzolna aplikacija provjerava postoji li korisnik s telefonskim brojem s kojeg je poruka poslana, te ako postoji obrađuje sadržaj poruke i šalje mu prikladan odgovor.



Slika 12. Primjer konfiguracije *webhook* adrese za zadani broj

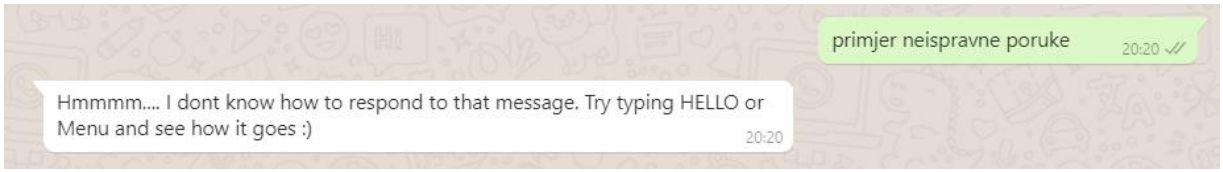


Slika 13. Primjer Ngrok prosljeđivanja

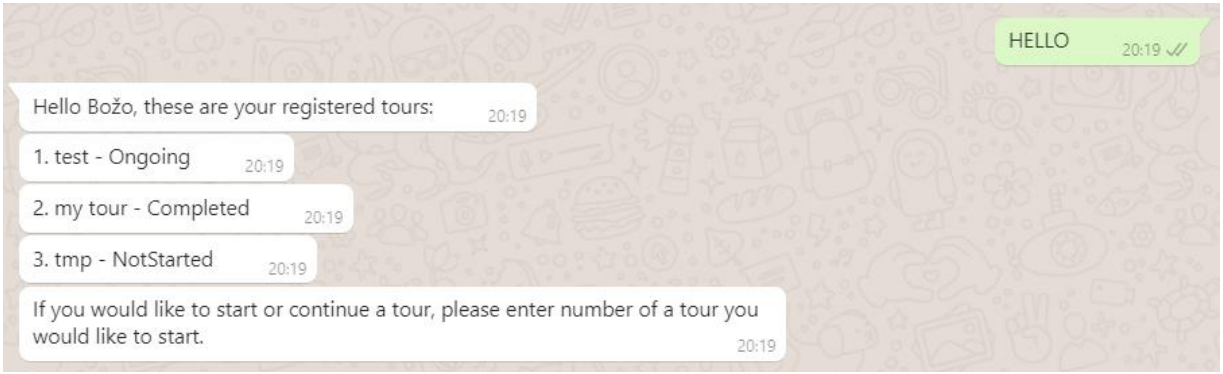


Slika 14. Primjer HTTP servera na portu 8000

SmartCityTour.Whatsapp aplikacija pristupa podacima iz baze podataka slično kao i klijentska aplikacija SmartCityTour.Web. Kod dolazne poruke konzolna aplikacija provjerava postoji li registrirani korisnik u bazi podataka s tim telefonskim brojem (telefonski broj registriranog korisnika mora biti jedinstven). Aplikacija može prepoznavati samo određen set poruka na temelju kojih obavlja neke od operacija nad korisnikovim turističkim obilascima. Cilj same interakcije je da bude što jednostavnija i intuitivnija, pa se korisnika kod unosa poruke koja nije prepoznata obavještava i savjetuje o drugim mogućnostima unosa kao što je prikazano na slici 15. Ako korisnik unese poruku koju aplikacija prepoznaje, korisniku se šalje predefinirana poruka te ga se navodi kroz daljnju komunikaciju kao što je prikazano na slici 16.



Slika 15. Primjer odgovora na neispravnu poruku



Slika 16. Primjer odgovora na ispravnu poruku

Razgovor korisnika s aplikacijom se može podijeliti u dvije faze. Prva faza sastoji se od odabira obilaska gdje su korisniku dostupne svi njegovi obilasci i funkcionalnosti poput pokretanja obilaska, resetiranja obilaska ili nastavka trenutnog obilaska. Druga faza komunikacije je sam obilazak u kojem je korisniku dostupan drugi set poruka (naredbi) s kojima komunicira s aplikacijom. Kad je obilazak u tijeku aplikacija ne prepoznaje naredbe faze odabira obilaska i obrnuto. Svaki put kad korisnik točno odgovori na pitanje iz kviza ažurira se redak *IsAnswered* unutar AttractionTour entiteta u *true*. Tako je sustav bodovanja obilazaka. Također isti entitet sadrži svojstvo *IsCompleted* koje se ažurira u *true* kada korisnik završi obilazak atrakcije (bilo s odgovorom na kviz ili preskakanjem kviza porukom "SKIP"). Tako se može pratiti napredak korisnika kod obilazaka. Nakon što korisnik završi obilazak svih atrakcija iz obilaska ponovno se vraća u prvu fazu komunikacije.

Nakon što aplikacija zaprimi poruku, i korisnik je autenticiran, aplikacija šalje odgovor preko *RestCilent* klase. Infobip programsko aplikacijsko sučelje pruža predefinirani kôd za slanje WhatsApp poruka. Primjer kôda za slanje poruka prikazan je u ispisu 19. Za autentikaciju

se koristi ključ programskog aplikacijskog sučelja *(engl. API key)* koji se dodaje u zaglavlje kod slanja zahtjeva. Kada se poruka primi na lokalni HTTP server ona se obrađuje. Aplikacija može prepoznavati samo određen set poruka na temelju kojih obavlja neke od operacija nad korisnikovim turističkim obilascima. Cilj same interakcije je da bude što jednostavnija i intuitivnija, pa se korisnika kod unosa poruke koja nije prepoznata obavještava i savjetuje o drugim mogućnostima unosa kao što je prikazano na slici 15.

```
public void sendMessage(string message)
{
    var client = new
RestClient("https://gy199w.api.infobip.com/whatsapp/1/message/text");
    client.Timeout = -1;
    var request = new RestRequest(Method.POST);
    request.AddHeader("Authorization", "App " + ApiKey);
    request.AddHeader("Content-Type", "application/json");
    request.AddHeader("Accept", "application/json");
    request.AddParameter("application/json", "
      {\"from\":\"447860099299\",\"to\":" + phoneNumber + ",\"messageId\":
      \"a28dd97c-1ffb-4fcf-99f1-0b557ed381da\",\"content\":{\"text\":\"" +
      message + "\",\"previewUrl\":true},\"callbackData\":\"Callback data\"}",
       ParameterType.RequestBody);
    IRestResponse response = client.Execute(request);
    Console.WriteLine(response.Content);
    Console.WriteLine(response.StatusDescription);
}
```

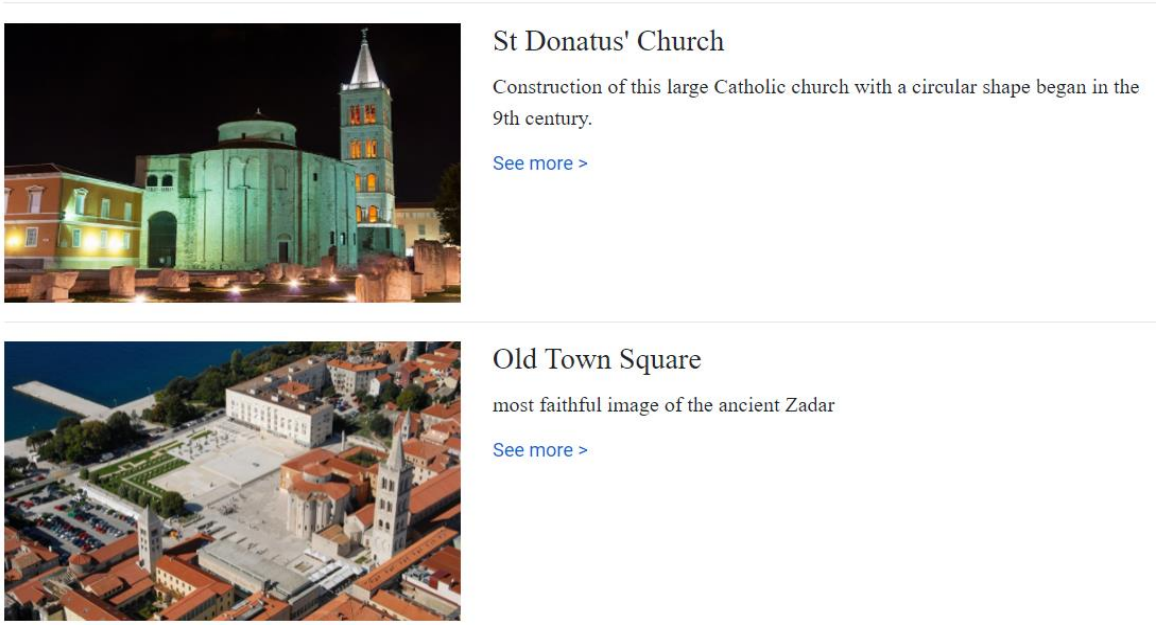
Ispis 19. Primjer metode za slanje poruke

4. OPIS APLIKACIJE

4.1. Klijentska strana aplikacije

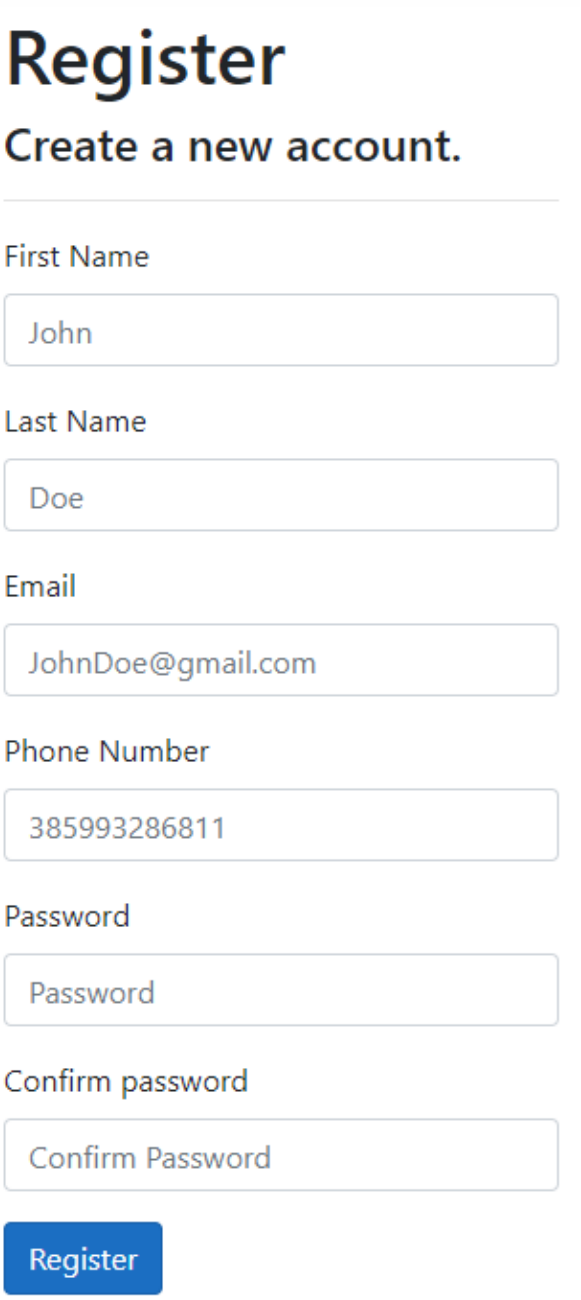
U ovom poglavlju bit će detaljno opisan način korištenja aplikacije. Korisnicima koji nemaju registrirani korisnički račun vidljivo je nekoliko stranica: home, attractions, museums, login i register. Svim stranicama može se pristupiti preko navigacijskog izbornika koji se nalazi na vrhu stranice. *Home* stranica služi kao početna stranica na kojoj se nalaze neke osnovne informacije o gradu Zadru uz pregled atrakcija i muzeja unutar karusela.

Attractions i *Museums* stranica su sličnog dizajna, samo jedna pruža informacije o atrakcijama a druga o muzejima. Svaka atrakcija prikazana je u listi kao što je prikazano na slici 17.



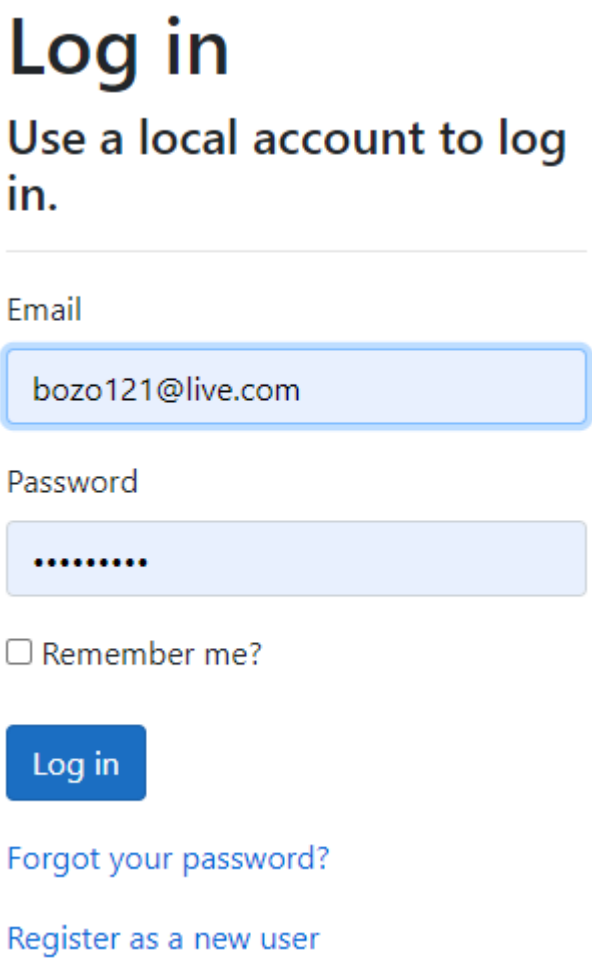
Slika 17. Prikaz atrakcija na stranici

Klikom na *See more* poveznicu otvara se stranica sa svim detaljima o atrakciji/muzeju. Sve informacije koje su vidljive definira administrator stranice. Korisnicima koji nisu registrirani dostupna je i stranica za registraciju prikazana na slici 18.



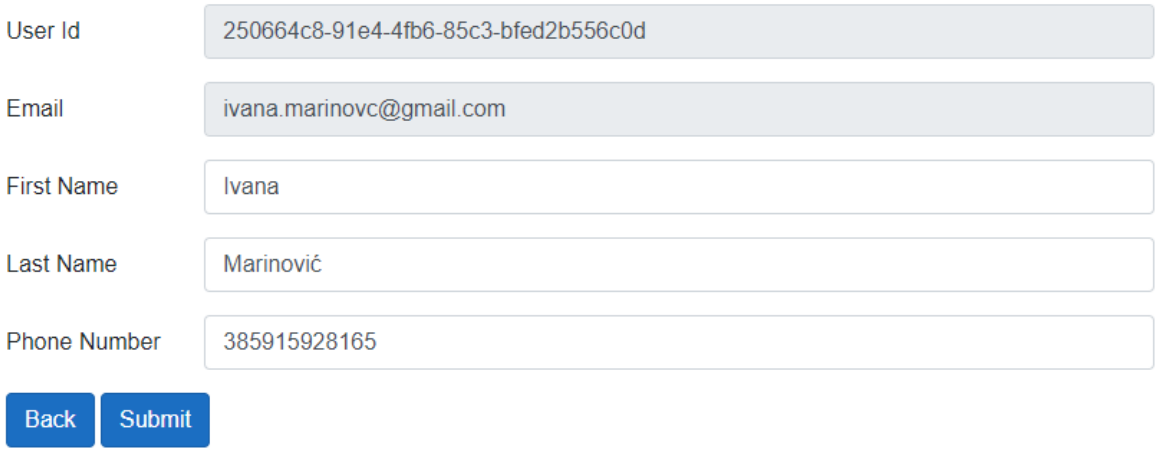
Slika 18. Forma za registraciju korisnika

Korisnik se nakon registracije može prijaviti na stranicu korištenjem registriranog email računa i lozinke kao što je prikazano na slici 19.



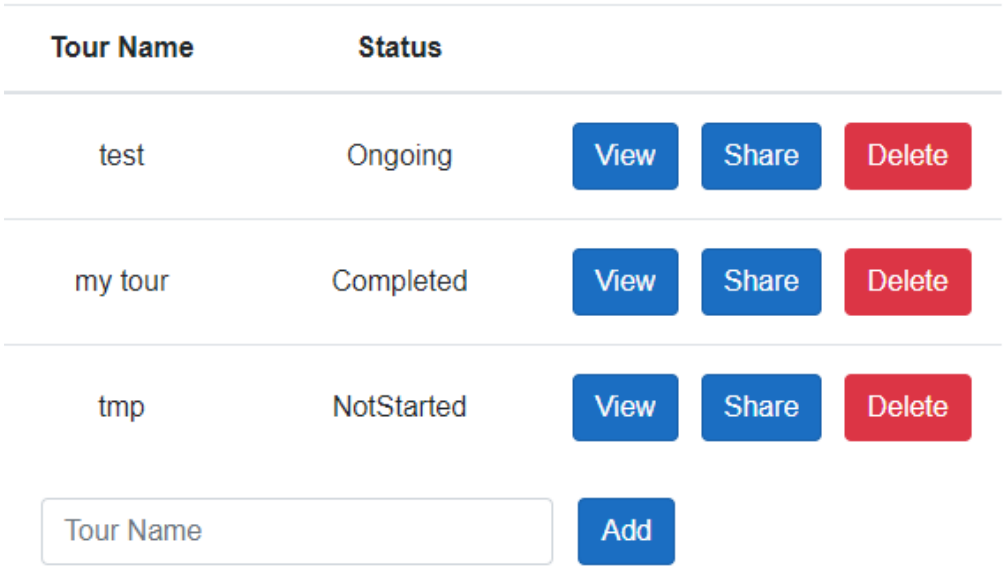
Slika 19. Forma za prijavu korisnika

Korisniku nakon prijave može pristupiti dodatnim stranicama *My Tours* za prikaz turističkih obilazaka i stranici za uređivanje korisničkog profila. Stranici za uređivanje korisničkih podataka može se pristupiti preko navigacijskog menija klikom na korisničko ime. Prikaz forme za uređivanje korisničkih podataka prikazan je na slici 20.



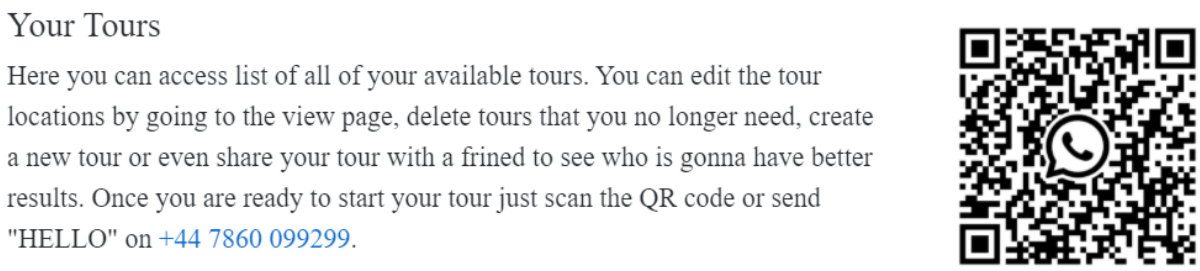
Slika 20. Forma uređivanje korisničkih podataka

Na stranici *MyTours* korisniku su dostupni svi njegovi turistički obilasci kao što je prikazano na slici 21. U listi su prikazani svi korisnikovi obilasci, status svakog obilaska, kao i botuni za pregled obilaska *(View)*, dijeljenje obilaska *(Share)* i brisanje obilaska *(Delete)*. Korisnik također može dodati novi obilazak unošenjem imena obilaska i pritiskom botuna *Add*.



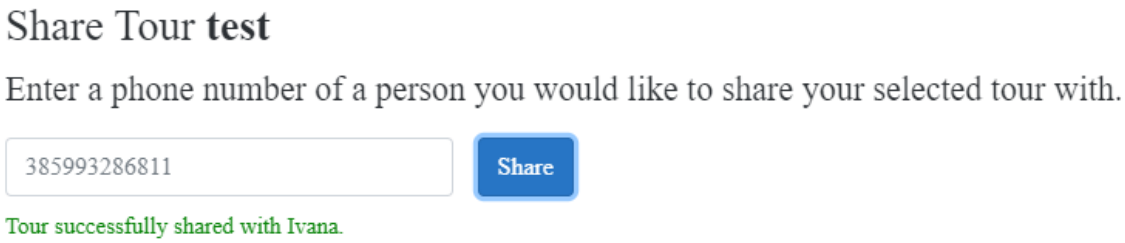
Slika 21. Prikaz liste korisnikovih obilazaka

Na stranici je dostupan i QR kôd koji skeniranjem otvarana WhatsApp razgovor na korisnikovom mobilnom uređaju s registriranim brojem za turistički obilazak. QR kôd prikazan je na slici 22. Korisniku je također na stranici dostupan i broj za WhatsApp komunikaciju kojeg može samostalno unijeti započeti razgovor.

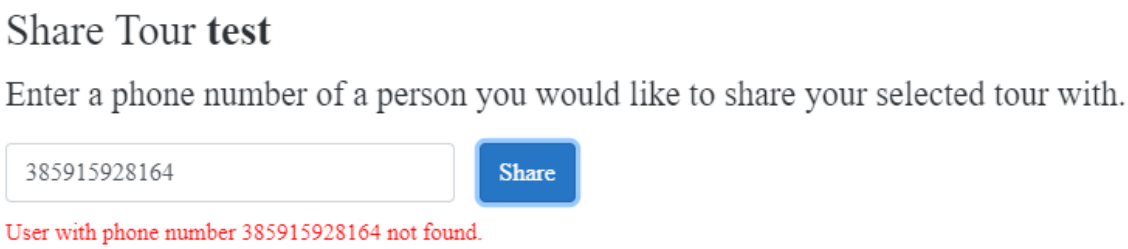


Slika 22. Prikaz QR kôda i broja za WhatsApp komunikaciju

Klikom na *Share* botun korisnik se usmjerava na stranicu za dijeljenje obilaska. Na stranici za dijeljenje korisnik unosi broj korisnika s kojim želi podijeliti obilazak. Ako postoji korisnik s tim registriranim brojem, tom korisniku se dijeli taj obilazak i korisnik se obavještava sukladno kao što je prikazano na slici 23. Ako korisnik ne postoji obilazak se ne dijeli te se korisnik također obavještava kao što je prikazano na slici 24.



Slika 23. Primjer uspješno podijeljenog obilaska



Slika 24. Primjer neuspješno podijeljenog obilaska

Klikom na View botun korisnik može vidjeti sve atrakcije u odabranom obilasku te listu dostupnih atrakcija koje može dodati u obilazak što je prikazano na slici 25. Ako obilazak dijeli više korisnika, obilazak se ažurira za sve korisnike jednako. Korisnik također može vidjeti listu ostvarenih bodova svih korisnika koji dijele obilazak što je prikazano na slici 26.

Name	Description	
St Donatus' Church	Construction of this large Catholic church with a circular shape began in the 9th century.	×
Old Town Square	most faithful image of the ancient Zadar	×
Cathedral of St. Anastasia	The Cathedral of St. Anastasia is the Roman Catholic cathedral of Zadar, Croatia, seat of the Archdiocese of Zadar, and the largest church in all of Dalmatia.	×
Queen Jelena Madijevka Park	1800s public park built on top of Zadar's old fortifications offering shaded promenades & a cafe.	×

Name	Description	
Sea Organ	Large experimental musical instrument mounted beneath a staircase & played by the waves on the sea.	+
Kalelarga	The street older than the city, once the Via Magna runs along the south edge of the Narodni trg ("People's Square").	+
Arsenal	One of the centers of the public, culture and entertainment, but also a place where a visitor can get all the necessary information about Zadar.	+

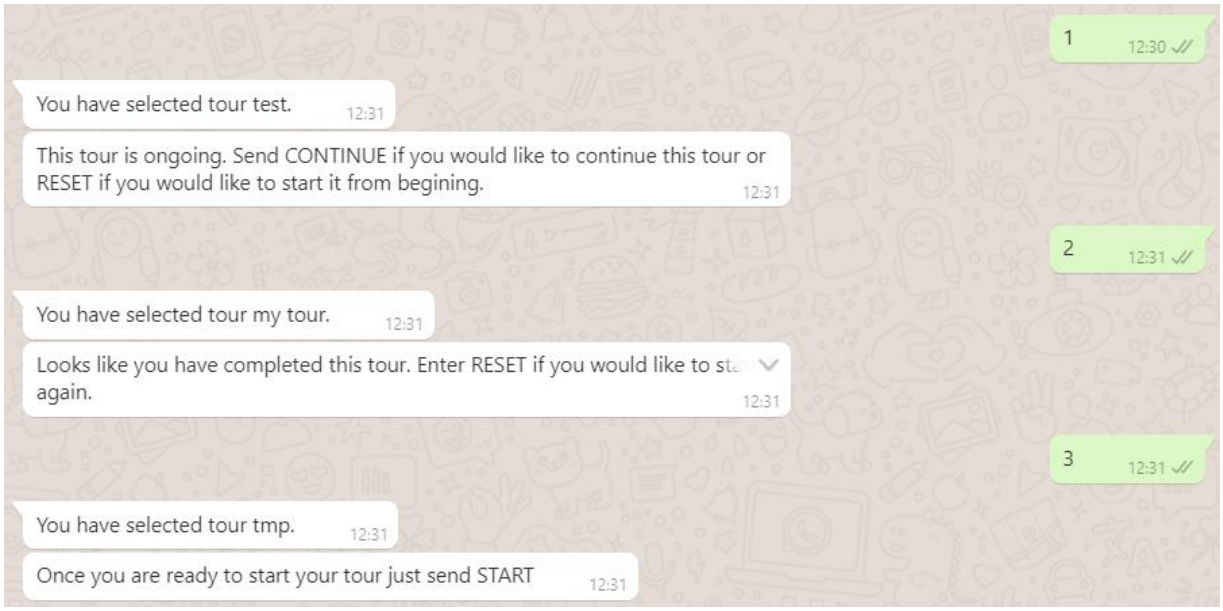
Slika 24. Prikaz za uređivanje odabranog obilaska

Leaderboard

Name	Answered Questions
Ivana	4
Božo	2

Slika 25. Prikaz tablice s ostvarenim bodovima korisnika

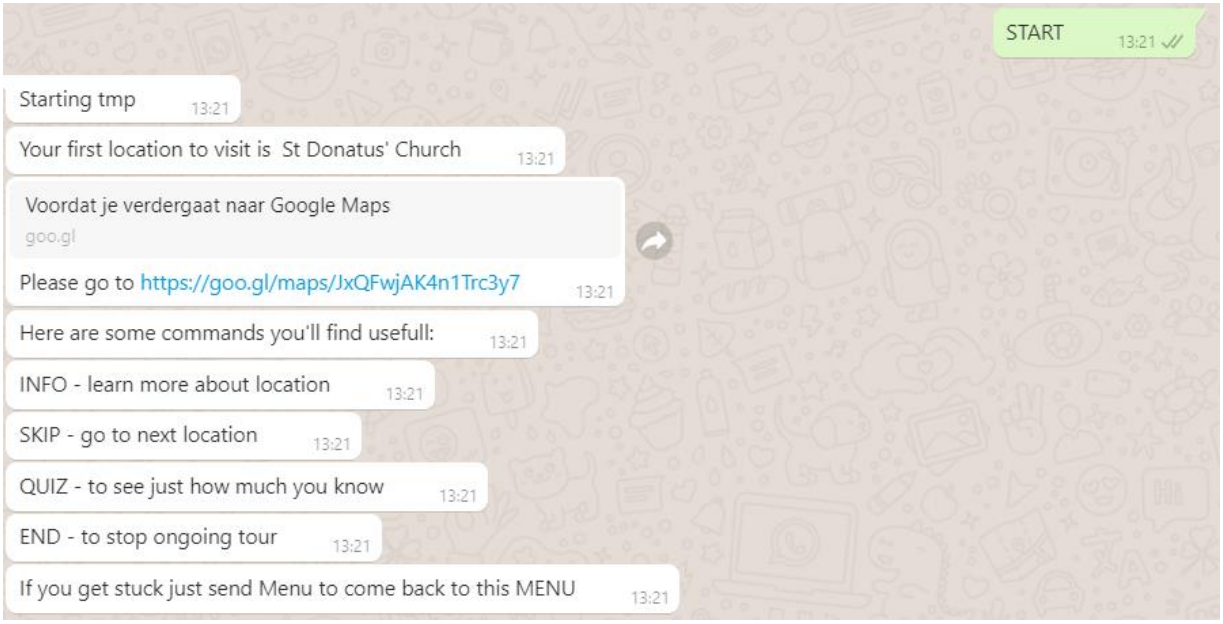
Korisnik kreirane turističke obilaske može koristiti putem WhatsApp platforme. Korisnik započinje komunikaciju slanjem "HELLO" ili "MENU" poruke, nakon čega dobije ispis svih njegovih turističkih obilazaka sa statusom kao što je prethodno prikazano na prikazano na slici 16. Korisnik tada odabire obilazak koji želi započeti, te ovisno o statusu obilaska dobiva prikladan odgovor kao što je prikazano na slici 27.



Slika 27. Prikaz odabira obilaska iz liste

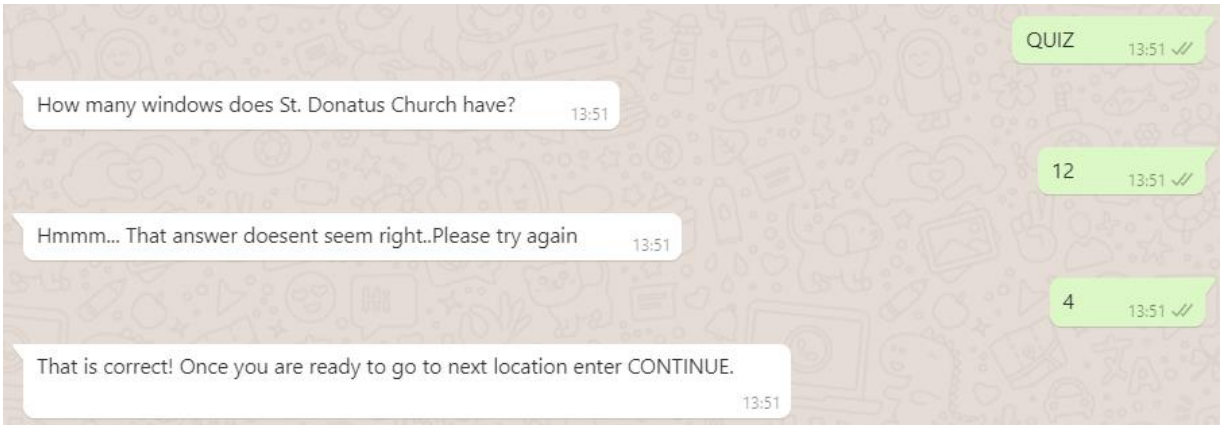
Nakon što korisnik odabere željeni obilazak može ga započeti unosom poruka prikazanih na slici 27. Obilazak koji još nije započet može se započeti slanjem *"START"* poruke. Obilazak

koji je započet a nije završen može se nastaviti slanjem *"CONTINUE"* poruke ili započeti iznova slanjem *"RESET"* poruke. Završeni obilazak može se započeti iznova slanjem "RESET" poruke. Nakon što korisnik započne obilazak šalju mu se podaci o prvoj neposjećenoj atrakciji. Također korisniku se kod slanja atrakcije šalje i popis poruka koje može koristiti tijekom turističkog obilaska. Prikaz pokretanja turističkog obilaska nalazi se na slici 28.



Slika 28. Prikaz pokretanja obilaska

Korisnik u bilo kojem trenutku može pristupiti izborniku s dostupnim porukama slanjem "MENU" poruke. Korisnik također može pauzirati trenutni obilazak unošenjem "END" poruke. Status obilaska tada prelazi u "Ongoing" I korisnik ga može nastaviti u bilo kojem trenutku kako je ranije opisano. Korisnik slanjem "INFO" poruke dobiva informacije o trenutnoj atrakciji. Korisnik kod razgledavanja može pristupiti kvizu, pri čemu mu se postavlja pitanje za trenutnu atrakciju. Nakon što korisnik pruži točan odgovor može nastaviti na sljedeću atrakciju te mu se dodjeljuje bod. Primjer kviza prikazan je na slici 29. Ako korisnik ne želi odgovoriti na kviz može ga preskočiti porukom "SKIP" i nastaviti na sljedeću atrakciju, no pri tome mu se ne dodjeljuje bod.

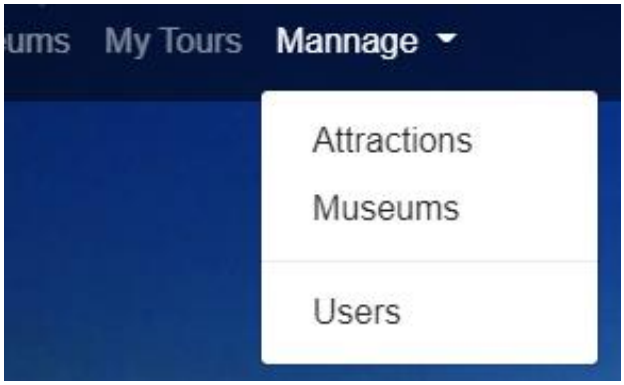


Slika 29. Prikaz kviza

Korisnik na opisani način prolazi sve atrakcije unutar obilaska. Nakon što prođe zadnju atrakciju u obilasku, korisnika se obavještava o završetku. Korisnik tada ponovno može pristupiti početnom izborniku i započeti neki od dostupnih obilazaka, ili ako je završio s korištenjem aplikacije jednostavno prekinuti komunikaciju.

4.2. Administrator aplikacije

Administratoru aplikacije dostupne su sve funkcionalnosti kao i korisniku aplikacije uz proširenu mogućnost dodavanja, brisanja i uređivanja pojedinih atrakcija, te mogućnost upravljanja korisničkim računima. Korisnik također može uređivati pitanja i točne odgovore koji se koriste kod obilazaka za pojedinu atrakciju. Administratoru aplikacije je nakon prijave dostupan padajući meni preko kojeg može pristupiti korisničkom sučelju za uređivanje podataka kao što je prikazano na slici 30.

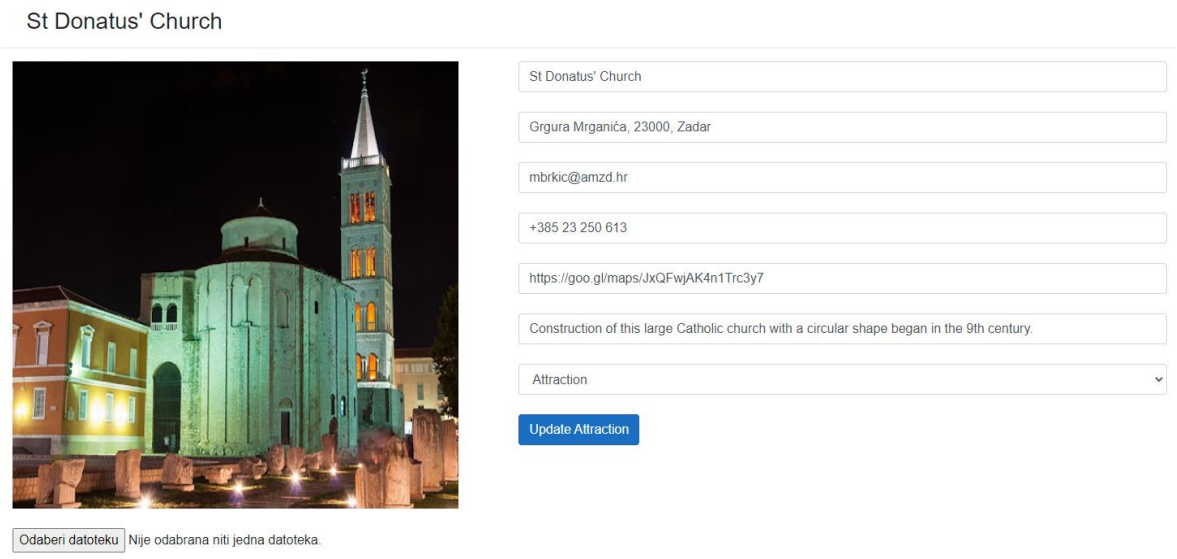


Slika 30. Prikaz padajućeg izbornika

Administrator se klikom na “Attractions” i “Museums” vodi na stranicu gdje je prikazana lista sa svim atrakcijama i muzejima. Prikaz jednog elementa liste nalazi se na slici 31.

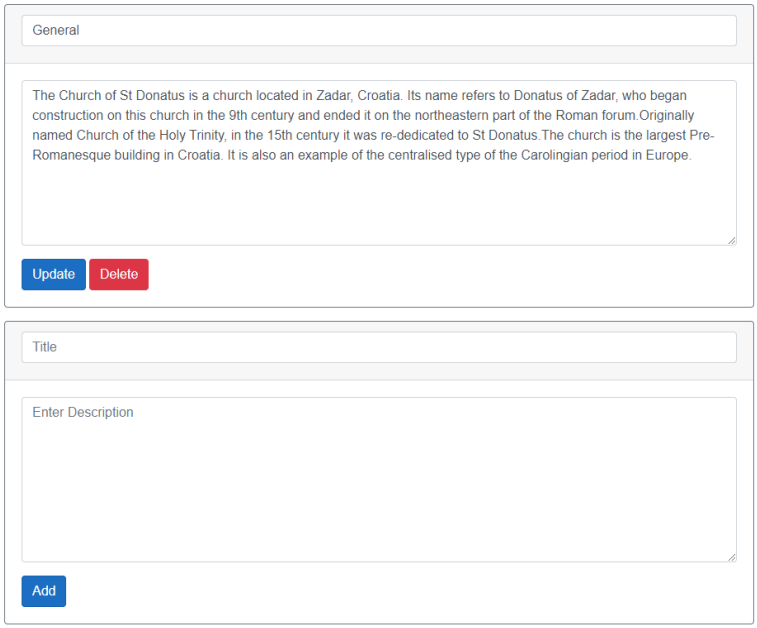
Slika 31. Prikaz elementa iz liste atrakcija

Administrator se klikom na “View”botun usmjerava na stranicu s dostupnim informacijama o atrakciji koja je vidljiva i korisniku aplikacije. Klikom na “Edit” botun korisnik se usmjerava na stranicu za uređivanje podataka o atrakciji. Klikom na “Questions” usmjerava se na stranicu za uređivanje pitanja i odgovora o pojedinoj atrakciji, te klikom na Delete briše sve podatke o atrakciji. Stranica za uređivanje podataka o atrakciji prikazana je na slici 32.

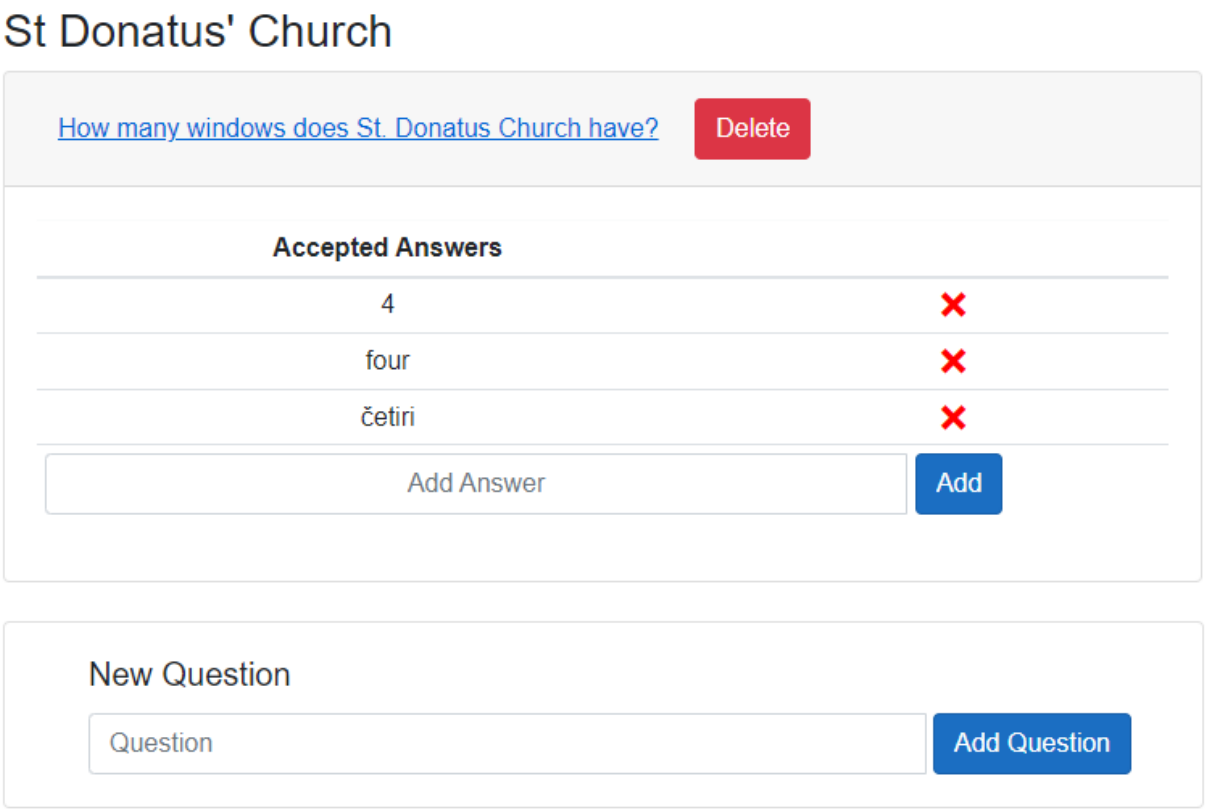


Slika 32. Forma za uređivanje podataka o atrakciji

Također, na stranici za uređivanje podataka o atrakciji korisnik može uređivati i dodavati informacije dostupne na stranici i kod WhatsApp obilaska što je prikazano na slici 33. Korisnik na stranici za pitanja i odgovore može dodavati nova pitanja i odgovore koji se prihvaćaju kao točni što je prikazano na slici 34.

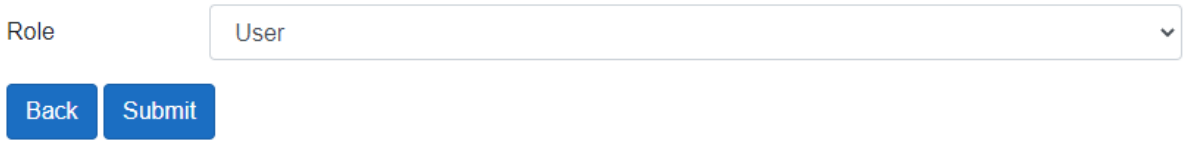


Slika 33. Forma za uređivanje i dodavanje informacija o atrakciji



Slika 34. Forma za dodavanje pitanja i odgovora

Administrator može vidjeti listi svih korisnika na stranici “Users” gdje može pristupiti svim korisnicima i upravljati njihovim podacima. Forma za uređivanje korisničkih podataka ista je kao i kod korisnika aplikacije, samo što admin može dodijeliti drugom korisniku ulogu administratora. Primjer uređivanja role korisnika prikazan je na slici 35.



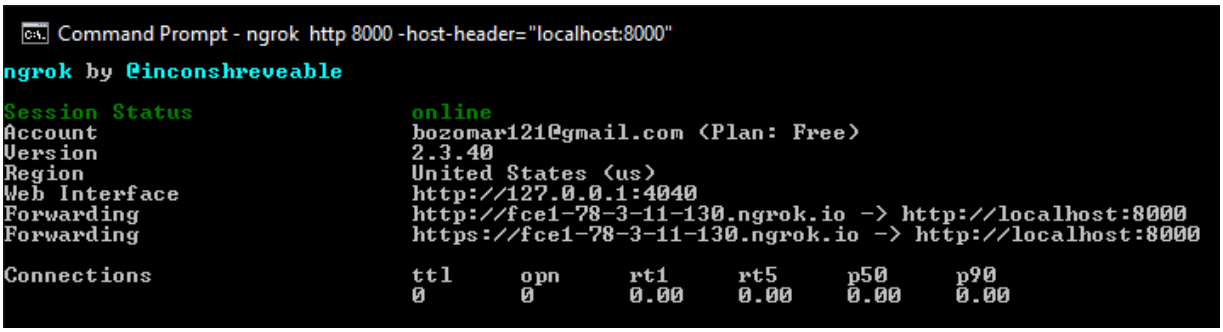
Slika 35. Forma uređivanje role korisnika

5. INSTALACIJA APLIKACIJE

Da bi se aplikacija uspješno pokrenula potrebno je napraviti nekoliko koraka koji će biti opisani u ovom poglavlju. Prvi korak je pokretanje Ngrok programskog alata koji generira javnu *webhook* adresu. Ngrok se pokreće unutar naredbenog retka izvršavanjem naredbe prikazane u ispisu 20. Nakon pokretanja Ngroka dostupna je jedna HTTP i jedna HTTPS *webhook* adresa kao što je prikazano na slici 36. Za funkcioniranje aplikacije možemo koristiti bilo koju od navedenih.

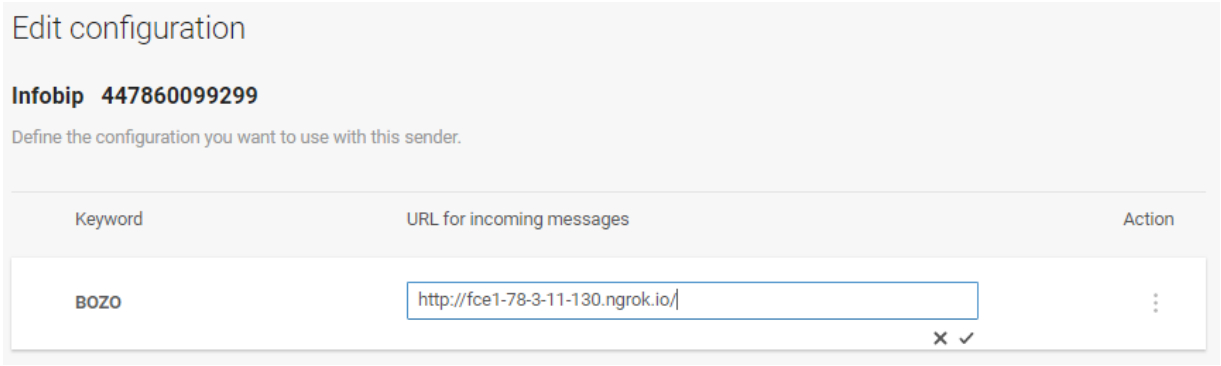
```
C:\Users\Korisnik\Desktop>ngrok http 8000 -host-header="localhost:8000"
```

Ispis 20. Primjer pokretanja Ngroka



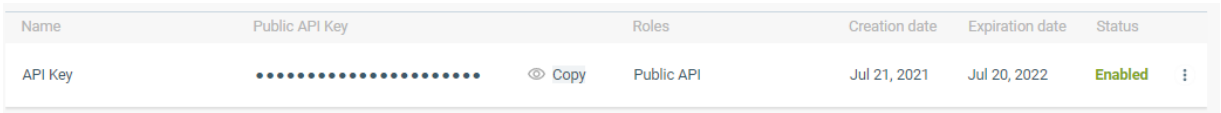
Slika 36. Primjer pokretanja Ngroka

Sljedeći korak je konfiguriranje *webhook* adrese za dodijeljeni broj korištenjem Infobip korisničkog sučelja. Za adresu koristimo *webhook* adresu generiranu od strane Ngroka kao što je prikazano na slici 37.

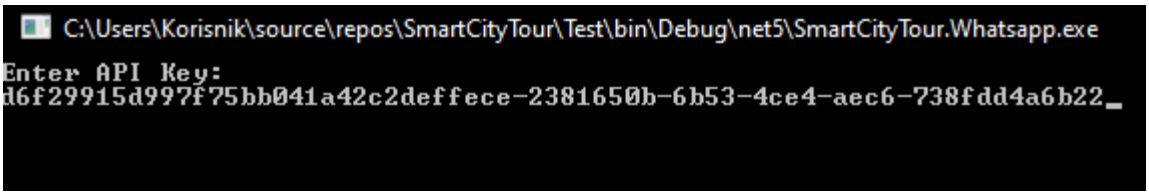


Slika 37. Primjer definiranja webhook adrese u Infobip korisničkom sučelju

Infobip korisničko sučelje koristimo i za generiranje ključa programskog aplikacijskog sučelja (engl. API key) kao što je prikazano na slici 38. Ključ programskog aplikacijskog sučelja koristimo kod slanja poruka iz SmartCityTour.WhatsApp projekta dodajući ga u zaglavlje zahtjeva u svrhu autentikacije. Ključ programskog aplikacijskog sučelja koristi se kod pokretanja konzolne aplikacije projekta SmartCityTour.WhatsApp kao što je prikazano na slici 39. Nakon napravljenih opisanih koraka aplikacija je spremna za korištenje.



Slika 38. Primjer dohvaćanja ključa programskog aplikacijskog sučelja



Slika 39. Unos ključa programskog aplikacijskog sučelja kod pokretanja aplikacije

6. ZAKLJUČAK

U ovom radu razvijena je potpuno funkcionalna aplikacija za pametno razgledavanje grada koja omogućava korisnicima kreiranje personaliziranih turističkih obilazaka. Turistički obilasci odvijaju se korištenjem WhatsApp platforme preko koje korisnici kroz interaktivnu komunikaciju obilaze turističke atrakcije. Možemo zaključiti da je cilj ovog rada ostvaren jer se korisnicima pruža mogućnost kreiranja turističkih obilazaka po želji, kao i mogućnost razgledavanja grada koristeći mobilni ili neki drugi uređaj s podrškom za WhatsApp.

Korisnik također može dijeliti svoje obilaske s prijateljima, obitelji ili drugim korisnicima aplikacije. Tako je postignuto zanimljivije iskustvo turističkog obilaska jer se korisnicima putem korisničkog sučelja omogućava uvid u broj ostvarenih bodova svih korisnika koji dijele obilazak.

Potencijalne nadogradnje sustava mogu biti u vidu sustava naplate i podrške za više jezika što bi aplikaciju približilo većem broju korisnika.

LITERATURA

[1] Microsoft Corporation “A tour of C# language”

https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

(posjećeno 28. kolovoza 2021.).

[2] Microsoft Corporation “Introduction to .NET”

https://docs.microsoft.com/en-us/dotnet/core/introduction (posjećeno 28. kolovoza 2021.).

[3] Microsoft Corporation “Welcome to the Visual Studio IDE”

https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019

(posjećeno 28. kolovoza 2021.).

[4] Microsoft Corporation, “Introduction to ASP.NET Core Blazor”

https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-5.0

(posjećeno 28. kolovoza 2021.).

[5] Microsoft Corporation, “An introduction to NuGet” https://docs.microsoft.com/en-us/nuget/what-is-nuget (posjećeno kolovoza 2021.).

[6] Microsoft Corporation “Entity Framework Core” https://docs.microsoft.com/hr-hr/ef/core/

(posjećeno 28. kolovoza 2021.).

[7] Microsoft Corporation “HttpClient Class”

https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=net-5.0
(posjećeno 28. kolovoza 2021.).

[8] Microsoft Corporation “RestSharp Introduction” https://restsharp.dev/getting-started/ (posjećeno 28. kolovoza 2021.).

[9] Microsoft Corporation “Introduction to Identity on ASP.NET Core”

https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio

(posjećeno 29. kolovoza 2021.).

[10] Ngrok “Documentation” https://ngrok.com/docs (posjećeno 29. kolovoza 2021.).

[11] Git “Documentation” https://git-scm.com/docs/git (posjećeno 29. kolovoza 2021.).

[12] Github “Git and Github essentials for Docs”

https://docs.microsoft.com/en-us/contribute/git-github-fundamentals

(posjećeno 29. kolovoza 2021.).

[13] Infobip “WhatsApp” https://www.infobip.com/docs/whatsapp (posjećeno 29. kolovoza 2021.).

[14] Infobip Company “Conversations API”

https://www.infobip.com/docs/api#customer-engagement/conversations-api

(posjećeno 29. kolovoza 2021.).

[15] Microsoft Corporation “Editions and supported features of SQL Server 2019”

https://docs.microsoft.com/en-us/sql/sql-server/editions-and-components-of-sql-server-version-15?view=sql-server-ver15 (posjećeno 29. kolovoza 2021.).

[16] Bootstrap “Getting Started” https://getbootstrap.com/docs/3.4/getting-started/

(posjećeno 29. kolovoza 2021.).

[17] Postman “How to use Postman for API Testing Automation”

https://www.blazemeter.com/blog/how-use-postman-manage-and-execute-your-apis (posjećeno 29. kolovoza 2021.).