Geo označena pripovjedačka karta

Janković, Ivan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

GEO OZNAČENA PRIPOVJEDAČKA KARTA

Diplomski rad

Ivan Janković

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Ivan Janković
Studij, smjer:	Sveučilišni diplomski studij Računarstvo
Mat. br. pristupnika, god.	D1288R, 07.10.2022.
JMBAG:	0165078488
Mentor:	izv. prof. dr. sc. Ivica Lukić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Mirko Köhler
Član Povjerenstva 1:	izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 2:	Miljenko Švarcmajer, univ. mag. ing. comp.
Naslov diplomskog rada:	Geo označena pripovjedačka karta
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Razviti web aplikaciju upotrebom C# ASP.NET i React tehnologija, te PostgreSQL bazom podataka. Aplikacija omogućuje korisnicima označavanje lokacija na karti, dijeljenje osobnih priča te pruža informacije o aktualnim događajima diljem svijeta. Kroz interaktivnu kartografiju, korisnici istražuju prošle priče i sudjeluju u društvenim interakcijama, potičući dijalog i razmjenu informacija unutar zajednice. Tema rezervirana za: Ivan Janković

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	20.08.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	18.9.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	23.09.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 23.09.2024.

Ime i prezime Pristupnika:	Ivan Janković
Studij:	Sveučilišni diplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D1288R, 07.10.2022.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Geo označena pripovjedačka karta**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivica Lukić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

Tehnologija u modernom svijetu preuzela je ključnu ulogu u oblikovanju svakodnevnog života te načina na koji ljudi doživljavaju i istražuju svijet. Njenim postepenim razvojem pristup informacijama postaje puno jednostavniji i pristupačniji te samim time pridonosi poboljšavanju i unaprjeđenju industrije zemalja. Među tim industrijama nalazi se i turizam koji primjenom označne internetske karte omogućuje korisnicima interaktivan i informativan pristup informacijama.

Karte su oduvijek bile temelj putovanja i istraživanja. S pomoću njih ljudi imaju vizualni pregled svijeta u kojem žive, pomoć pri putovanjima kao i pri istraživanju novih svjetskih mjesta. U kombinaciji s tehnologijom krenule su se stvarati kartografske aplikacije koje omogućuju puno bolji pregled svijeta od klasičnih karti. Ove moderne karte su dinamične i interaktivne te uključuju slojeve informacija koje nadilaze jednostavnu geografiju.

Turizam je jedna od jačih globalnih industrija koja korištenjem digitalnih alata putniku osigurava povećani ugođaj, obogaćuje avanture i poboljšava samo iskustvo putovanja. Porastom i razvojem turizma putnicima, odnosno turistima kartografske aplikacije postaju sve važniji alat, nudeći mogućnost interaktivnog i detaljnog istraživanja destinacija i mogućih događanja.

Aplikacija koja se opisuje u ovom diplomskom radu je primjer jedne takve kartografske aplikacije. *Geo označena pripovjedačka karta* korisnicima nudi lakši pristup informacijama i bolji vizualni pregled lokacije za razne događaje, turistička mjesta te osobne priče i slike korisnika.

1.1. Zadatak diplomskog rada

Razviti web aplikaciju upotrebom C# ASP.NET i React tehnologija, te PostgreSQL bazom podataka. Aplikacija omogućuje korisnicima označavanje lokacija na karti, dijeljenje osobnih priča te pruža informacije o aktualnim događajima diljem svijeta. Kroz interaktivnu kartografiju, korisnici istražuju prošle priče i sudjeluju u društvenim interakcijama, potičući dijalog i razmjenu informacija unutar zajednice.

2. PREGLED PODRUČJA TEME

Moderno digitalno doba stvorilo je sve predispozicije za kreaciju velikog obujma kartografskih aplikacija za različite potrebe, također, i aplikacija specifičnih za turizam i razna događanja. Na primjer, aplikacije poput *Google Maps* i *Sygic Maps* pružaju korisnicima jedinstveno sučelje u kojem mogu istraživati različite turističke lokacije te dohvaćati detaljne informacije vezane za te lokacije. Nadalje, u području svjetskih događanja, odnosno koncerata i festivala postoji platforma *SongKick* koja omogućuje praćenje budućih izvedbi te nudi detaljne informacije o istim.

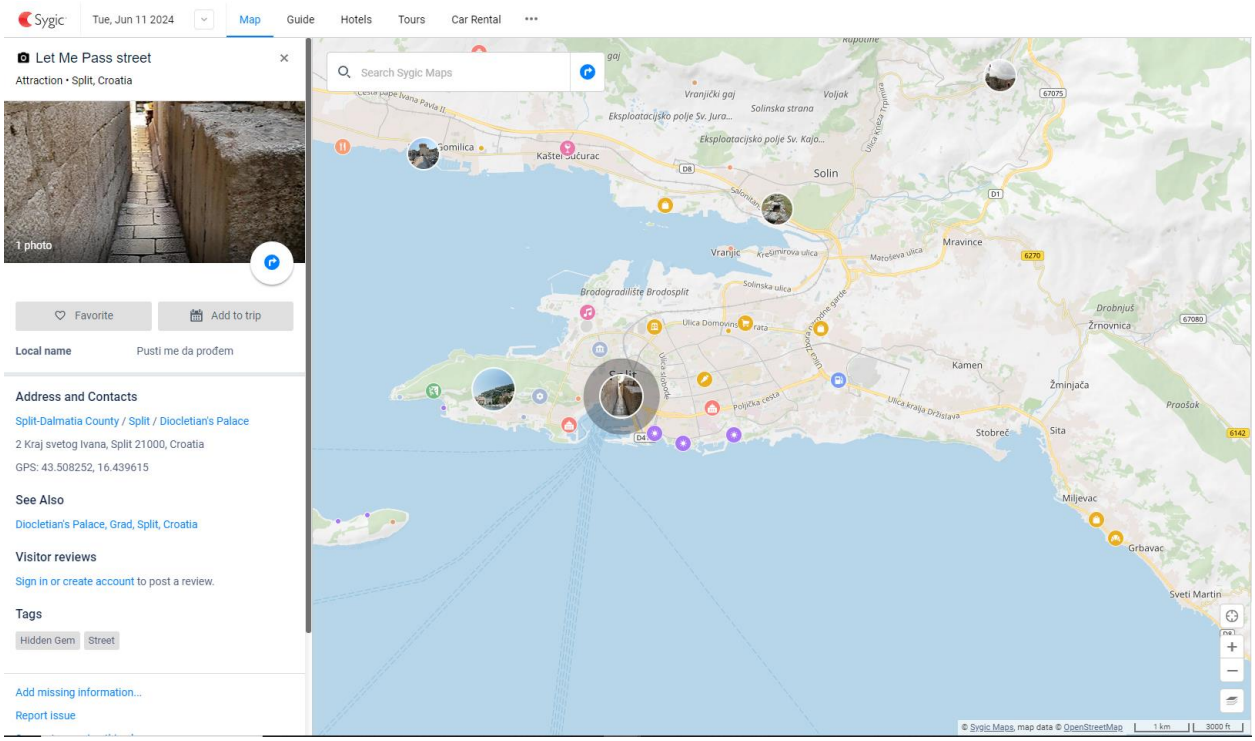
2.1. Google Maps

Google Maps (slika 2.1.) je web i mobilna aplikacija koju je napravio Google. Nudi detaljne informacije o geografskim područjima i lokacijama širom svijeta. Jedna je od najpoznatijih i najpreciznijih kartografskih aplikacija koje postoje na tržištu. Primarne značajke koje *Google Maps* nudi su detaljnu kartu svijet s geografskim informacijama, navigaciju, detaljne informacije poduzeća i turističkih mjesta. Također, pruža satelitski i ulični (*eng. Street View)* prikaz svijeta te samim time olakšava navigaciju kroz određena područja [1].

Slika 2.1 *Google Maps*

2.2. Sygic Maps

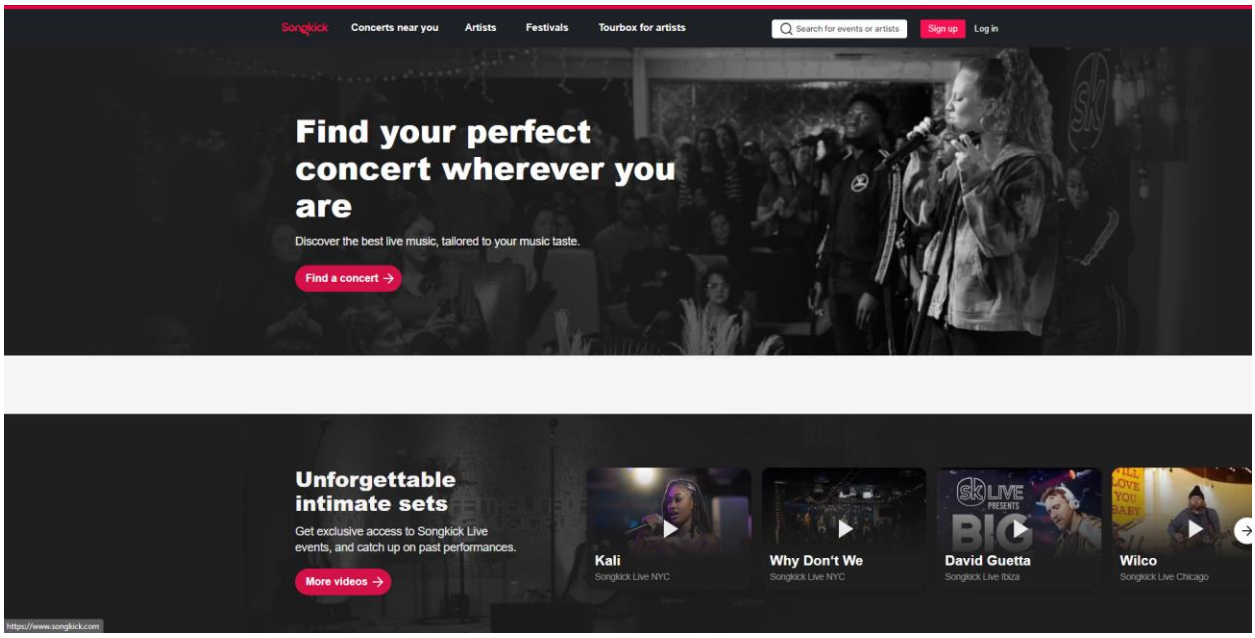
Kartografsko sučelje koje pruža prikaz najboljih atrakcija, hotela, restorana i trgovina širom svijeta. Namijenjeno je za turiste, odnosno za planiranje putovanja te istraživanje novih i atraktivnih lokacija. *Sygic* (slika 2.2.) je također poznat po svome GPS navigacijskom softveru koji osim informacija o navigaciji pruža informacije o prometu u stvarnom vremenu, kao što su gužve, ograničenja brzine te odabir odgovarajuće vozne prometne trake u kojoj vozač treba biti [2].



Slika 2.2 *Sygic Maps*

2.3. SongKick

Web sučelje koje korisnicima pruža uvid u buduće događaje, koncerte i festivale. Omogućuje obožavateljima da prate svoje najdraže izvođače kroz njihove turneje te im pružaju uslugu kupnje ulaznica. Točnije, ulaznice se ne prodaju direktno na *SongKick* aplikaciji (slika 2.3) već vodi na mjesta koja su ovlaštena za njihovu prodaju. Također, korisnicima se pruža mogućnost postavljanja recenzija o koncertima kako bi podijelili svoja iskustva sa zajednicom [3].



Slika 2.3 *SongKick Web Aplikacija*

3. KORIŠTENE TEHNOLOGIJE

Razvoj *Geo označene pripovjedačke karte* sastoji se od tri ključna dijela, a to su baza podataka, poslužiteljski dio (eng. back-end) i korisnički dio (eng. front-end). Baza podatak je relacijskog tipa i korišten je PostgreSQL. Glavna značajka relacijske baze podataka je sposobnost organiziranja podataka u tablice koje su povezane relacijama korištenjem jedinstvenih identifikatora. Poslužiteljski dio aplikacije sastavljen je u Microsoft ASP.NET okruženju koristeći c# programski jezik te koristi više-slojnu (eng. Multi-Layer) arhitekturu za rukovanje podacima. Takav tip modela arhitekture je jedan od najkorištenijih modela za softversku arhitekturu. Cijeli poslužiteljski dio rastavljen je na prezentacijski sloj, sloj poslovne logike te sloj za pristup podacima. ReactJS okruženje (eng. Framework) koristi se u korisničkom dijelu aplikacije.

3.1. PostgreSQL

PostgreSQL (slika 3.1) je objektno-relacijska baza podataka otvorenog koda koja koristi i proširuje SQL jezik. Bazirana je na POSTGRES verziji 4.21 koja je razvijena na odjelu za računalne znanosti Sveučilišta Kalifornije u Berkeleyju. Radi na svim glavnim operacijskim sustavima. ACID (eng. Atomicity, Consistency, Isolation, Durability) je usklađen od 2001. godine te posjeduje PostGIS geoprostorno proširenje za bazu podataka. PostGIS daje mogućnost PostgreSQL-u da sprema, indeksira i izvršava upite s geoprostornim podacima. Podržava najpopularnije programske jezike kao što su Python, Java, C#, C/C++, Ruby i mnoge druge. Moderne funkcionalnosti poput kompleksnih SQL upita, stranih ključeva, okidača (eng. trigger), transakcija i još puno njih su podržane od strane PostgreSQL [4].

Slika 3.1 *PostgreSQL logo*

3.2. ASP.NET Framework

.NET je Microsoftova razvojna platforma sastavljena od alata, programskih jezika i biblioteka za izgrađivanje različitih tipova aplikacija. Dvije glavne komponente .NET Frameworka su CLR (eng. Common Language Runtime) i klasna biblioteka (eng. Class Library). CLR je izvršno okruženje za .NET aplikacije te pruža usluge programerima kao što su upravljanje memorijom, sigurnost, izvršavanje koda prevođenjem međujezika u strojni, upravlja izuzetcima te pruža interoperabilnost, odnosno omogućava komunikaciju između .NET kod i koda napisanog u drugim jezicima. Klasna biblioteka pruža set API-ova za zajedničku funkcionalnost [5].

ASP.NET (slika 3.2) je podsustav .NET Frameworka i nasljednik klasične ASP (eng. Active Server Page). Izgrađen je na CLR-u koji omogućuje programerima da izvršavaju kod koristeći .NET jezik (C#, VB itd.). Posebno je dizajniran da radi s HTTP protokolima te da omogućuje programerima stvaranje dinamičnih web stranica, odnosno aplikacija. Nadodaje na .NET Framework i autentifikacijski sustav koji uključuje biblioteke, bazu podataka i šablonske stranice za rukovanjem prijavama [6].

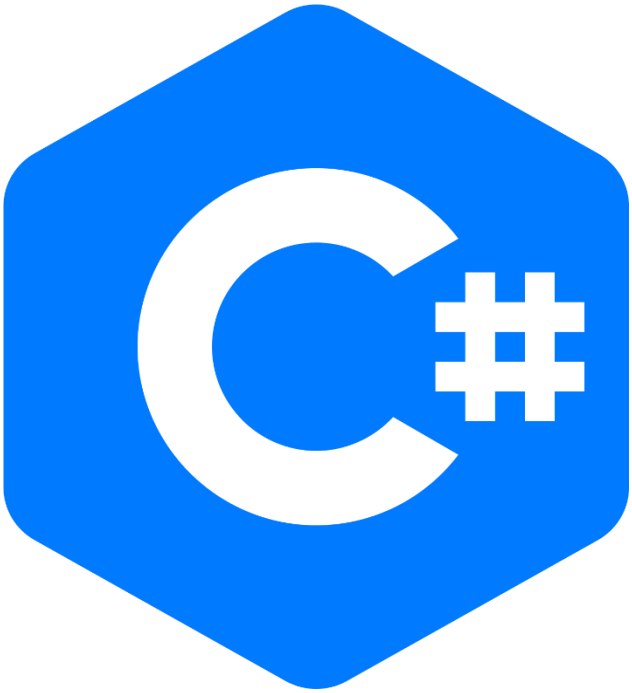
Slika 3.2 *ASP.NET logo*

3.3. C#

Objektno orijentirano programiranje (OOP) je programski model koji sastavlja softverski dizajn oko podataka, točnije objekata, a ne oko funkcija. Objekt predstavlja neku stvar iz stvarnog života (slika 3.3) te je upravo iz toga razloga programerima puno lakše sastavljati i manipulirati logiku koja je vezana za tu stvar u programskom smislu. Struktura ili građevne jedinice objektno orijentiranog programiranja uključuje klase, objekte, metode i atribute, dok su principi objektno orijentiranog programiranja enkapsulacija, apstrakcija, nasljeđivanje i polimorfizam [7].

Slika 3.3 *Primjer strukture objektno orijentiranog programiranja*

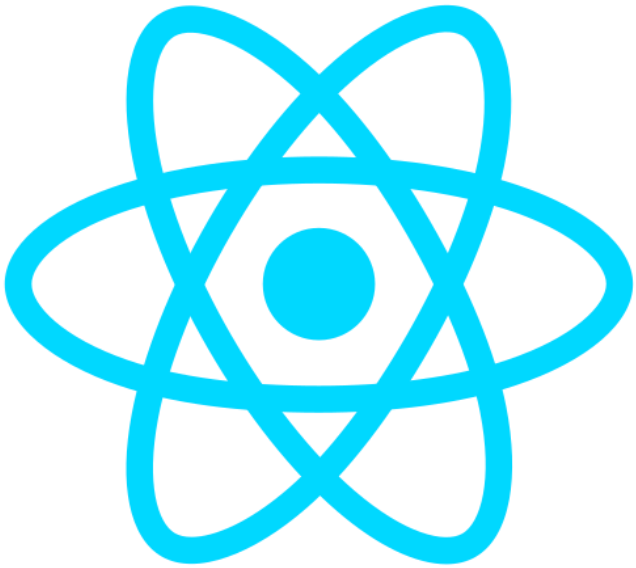
C# (slika 3.4) je opće namjenski, siguran po tipu (eng. type-safe), objektno orijentiran jezik. Uravnotežuje jednostavnost, izražajnost i produktivnost. Glavni arhitekt jezika od njegove prve verzije je Andres Hejlsberg. Platformski je neutralan programski jezik. C# je bogata implementacija objektno orijentirane paradigme, koja uključuje enkapsulaciju, nasljeđivanje i polimorfizam. Važnost enkapsulacije je u tome da kreira granice oko objekta kako bi se odvojilo njegovo vanjsko, javno ponašanje od njegove unutarnje, odnosno privatne implementacije. Značajke koje su karakteristične za C# iz objektno orijentirane perspektive su jedinstven tip sustav, definiciju klasa i sučelja te implementaciju svojstava, metoda i događaja [8].



Slika 3.4 *C# logo*

3.4. ReactJS

ReactJS je JavaScript biblioteka otvorenog koda za izgradnju korisničkih sučelja zasnovanim na komponentama. Glavna karakteristika mu je da se sastavljene komponente mogu ponovno upotrebljavati, što pomaže programerima da učinkovito upravljaju i organiziraju svoj kod. Naglasak postavlja na kreiranje vizualno interaktivnog i visoko-učinkovitog korisničkog sučelja. Kako bi se to ostvarilo koristi se virtualni DOM (eng. Document Object Model) koji je izrazito učinkovitiji od običnog, gledajući na brzinu. Učitavanjem stranice na internetu kreira se hijerarhijska struktura stabla koja predstavlja sadržaj same stranice. S programske strane, to je HTML dokument sa svojim elementima, a sve zajedno to predstavlja DOM. Glavni problem korištenja izvornog DOM-a je to što svaki puta kada programer promjeni nešto na nekom HTML elementu, cijeli DOM se rekreira od nule. Ovakvo ponašanje u velikoj mjeri utječe na brzinu i na učinkovitost aplikacije. Kao rješenje za ovaj problem ReactJS je osmislio virtualni DOM. Memorijsku reprezentaciju, odnosno kopiju stvarnog DOM-a, no ta kopija nema moć da direktno promjeni raspored stranice. Kada se nešto novo doda u aplikaciju ili staro izmjeni kreira se virtualni DOM predstavljen kao stablo. Svaki HTML element je čvor toga stabla te u slučaju promjene kreira se ponovno virtualni DOM koji se zatim uspoređuje sa starim te se bilježe promjene. Nakon toga procesa pronalazi se najbolji način za primjenu tih promjena na stvarni DOM te se umjesto ponovnog učitavanja cijelog DOM-a renderiraju samo oni elementi koji su ažurirani [9].



Slika 3.5 *ReactJS logo*

4. IZRADA APLIKACIJE

Aplikacija je podijeljena na tri glavna dijela. Prvi dio je sastavljanje baze podataka, odnosno relacijske, PostgreSQL baze podataka. Zatim drugi dio je sastavljanje poslužiteljskog dijela aplikacije u kojem je uspostavljena cijela njena logika kao što je autorizacija, komunikacija između baze podataka i korisničkog dijela aplikacije, manipulacija podacima i tako dalje. Korisničko sučelje je treći dio u kojem se zapravo pruža korisniku vizualizacija svih podataka, interakcija te kontrola nad podacima ovisno o razini pristupa kojeg korisnik ima.

4.1. Baza Podataka

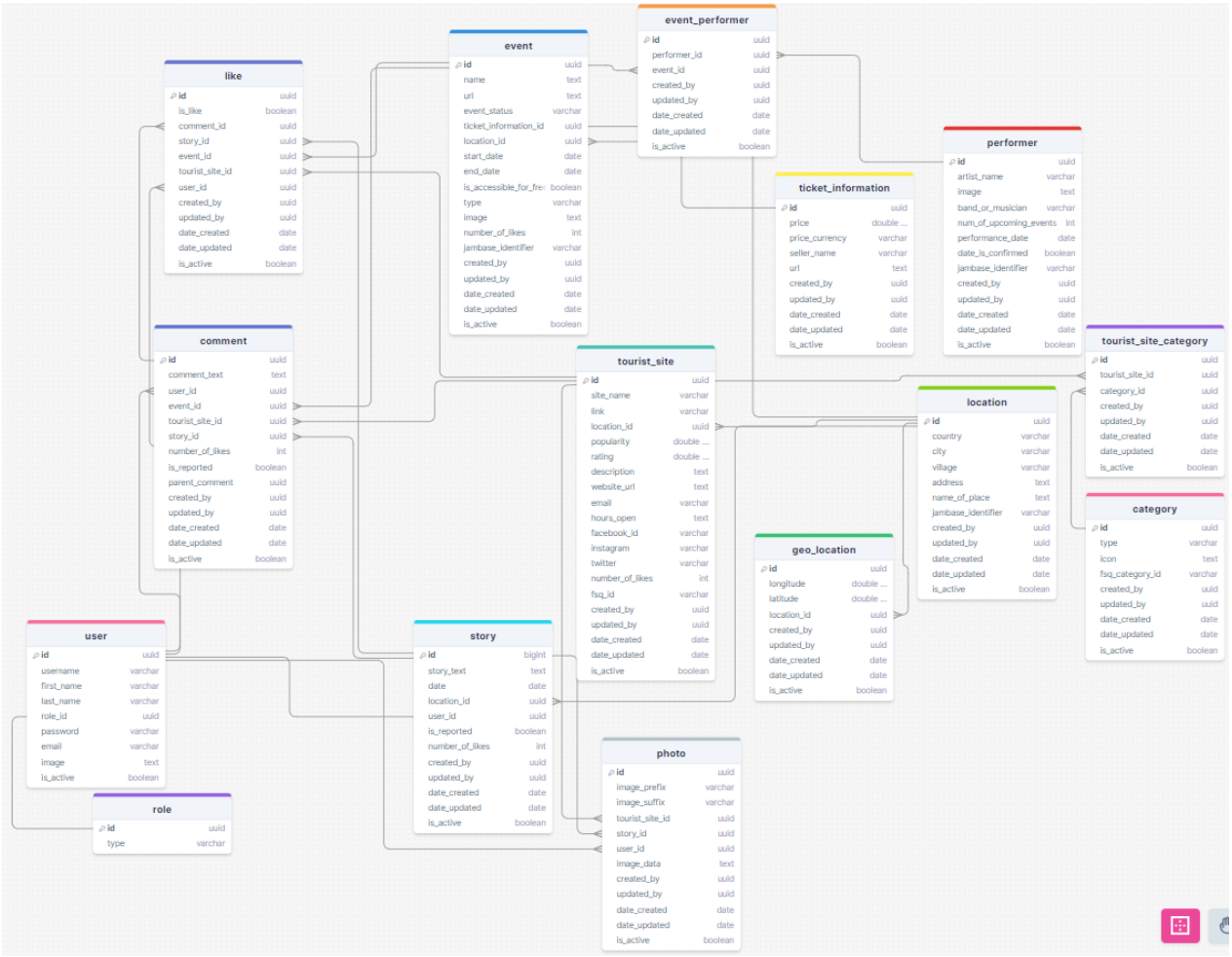
Baza podataka je relacijskog tipa, drugim riječima to je skup podataka koji su međusobno povezani nekakvim relacijama. Podaci se spremaju u tablice koje se sastoje od stupaca i redova što stvara bolju vizualizaciju svih podataka i veza (tablica 4.1) . Relacije se sastavljaju na logički način kao što bi se u stvarnom životu povezale dvije stvari koje međusobno jedna na drugu utječu preko nekog zajedničkog atributa. Svaki stupac unutar neke tablice pohranjuje određeni tip podataka, kao na primjer cijeli broj (eng. integer), tekst (eng. text) i slično. Svi redovi unutar tog stupca moraju biti istog podatkovnog tipa. Unutar tablice mora se nalaziti stupac koji pohranjuje jedinstvene identifikatore redova, koji zapravo predstavljaju primarni ključ (eng. Primary Key) te tablice. Veza između dvije tablice se uspostavlja upravo s pomoću stranog ključa (eng. Foreign Key) koji referencira primarni ključ neke druge tablice. Primarni ključ unutar tablice je podatak koji uvijek treba imati vrijednost te ne smije nikada biti prazan (eng. NULL), dok strani ključ može biti prazan ako relacija nije potrebna.

	Id [PK] uuid	Type Character varying (255)
1	0a4431b2-209b-46e9-bcfa-9d2e17b6e583	User
2	0d071ef9-46cf-4436-87d5-d2b7e140d8e6	SysAdmin
3	e8008559-d501-455b-b452-5e4b8cb2284c	Admin

Tablica 4.1 *Uloge (eng. Role) unutar baze podataka*

Relacijska baza podataka, njezine sve tablice te relacije između tih tablica, se prikazuje s pomoću relacijskog dijagrama, odnosno s pomoću ERD (eng. Entity Relationship Diagram) (slika 4.1). Veza koja se postavlja između tablica može biti jedan-na-jedan (eng. one-to-one), jedan-na-više (eng. one-to-many) i više-na-više (eng. many-to-many) te ona ovisi o načinu na kojeg jedna tablica utječe na drugu. Tako u ERD-u aplikacije može se uočiti kako tablica događaj (eng. event) i tablica

informacije o ulaznicama (eng. ticket_information) imaju jedan-na-jedan vezu zbog toga što jedan događaj može imati samo jednu informaciju o ulaznicama, a jedna informacija o ulaznicama može pripadati samo jednom događaju. Više-na-više veza se razbija tako da se postavi jedna tablica između te dvije tablice. Na primjer događaj i izvođač imaju više-na-više vezu jer jedan događaj može imati više izvođača, a jedan izvođač može pripadati u više događaja te iz toga razloga se između te dvije tablice postavila treća tablica izvođači događaja (eng. event_performer).



Slika 4.1 *ERD baze podataka.*

4.2. Poslužiteljski dio

Poslužiteljska strana aplikacije sastavljena je s pomoću višeslojne softverske arhitekture (eng. Multi layered software architecture) te se komunikacija između slojeva ostvaruje s pomoću ubrizgavanja ovisnosti (eng. dependency injection). Unutar slojeva uspostavljene su funkcionalnosti poput popunjavanja događaja i turističkih mjesta u bazu podataka koristeći vanjski API (eng. Application Programming Interface). Uspostavljen je okidač s pomoću Quartz biblioteke za raspoređivanje zadataka koja se koristi za zakazivanje i upravljanje periodičnim

zadacima. Okidač je uspostavljen zbog događaja, tako da se svaki dan u ponoć ažurira baza podataka za događaje i sve ostale tablice koje su povezane s njom. Nadalje uspostavljena je i autorizacija te razine pristupa određenim podacima i funkcionalnostima.

4.2.1. Višeslojna arhitektura

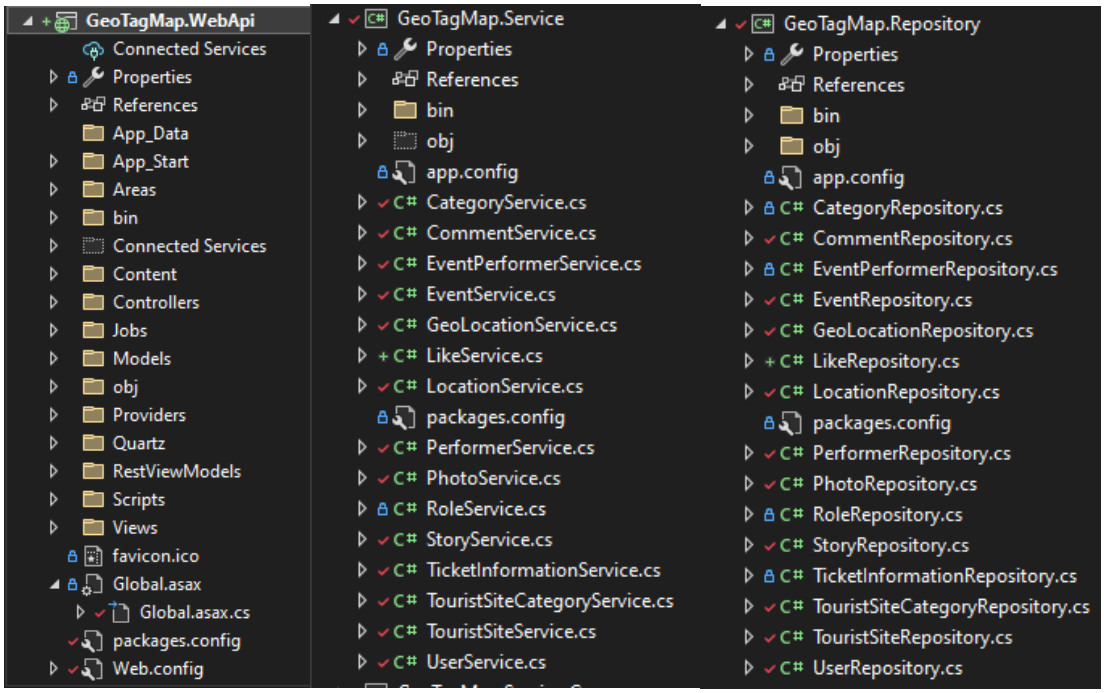
Taj tip arhitekture je jedan od najpoznatijih i najpopularnijih u današnjem vremenu. Također, ponekad taj tip arhitekture se naziva i n-sloj (eng. n-tier). Sastoji se od različitih slojeva s tim da svaki sloj ima specifičnu zadaću. Jedna od većih prednosti n-slojne arhitekture je u tome što prepravljanje različitih slojeva je uvelike olakšano jer je svaki sloj zaseban. Najjednostavniji tip je troslojna arhitektura, od koje sve ostale počinju [10].

Prezentacijski sloj je na najvišoj razini slojeva te on pruža prezentacijske usluge podataka. Odgovornost mu je interakcija s korisnikom te prikaz i prosljeđivanje podataka. Ovome sloju može se pristupiti putem bilo kojeg klijentskog uređaja poput laptopa, tableta, mobitela i slično. Prima od strane korisnika podatke, zatim ih prosljeđuje poslovnom sloju na obradu, te na kraju prikazuje obrađene podatke korisniku.

Sloj poslovne logike je srednji sloj arhitekture te se u njemu obrađuju podaci. Sadržava poslovna pravila i logiku prema kojima se podaci ažuriraju, modificiraju i pohranjuju. Odgovoran je za operacije kao što su validacija podataka, različite kalkulacije, transakcije i upravljanje radnim procesima.

Najniži sloj arhitekture je sloj za pristup podacima unutar baze podataka, odnosno DAL (eng. Dana Access Layer). Njegova glavna zadaća je komunikacija s bazom podataka, točnije pristup podacima unutar nje. Koristi se za spremanje, dohvaćanje, ažuriranje i brisanje podataka. Pojednostavljuje interakciju s bazom podataka, što omogućuje poslovnom sloju da ne mora znati kako se podaci koje on obrađuje spremaju i dohvaćaju te mu omogućuje da se fokusira na samu obradu podataka.

Višeslojna arhitektura unutar *GeoTagMap* aplikacije postavljena je na sljedeći način: prezentacijski sloj čini WebAPI, poslovnu logiku čini servis te sloj za pristup podacima čini repozitorij (slika 4.2).



Slika 4.2 *GeoTagMap WebApi, Servis i Repozitorij struktura*

Osim prezentacijskog sloja, poslovnog sloja i sloja za pristup podacima postoji, također, i sloj za modele. Modeli predstavljaju tablice unutar baze podataka na poslužiteljskoj strani aplikacije.

4.2.2. Ubrizgavanje ovisnosti

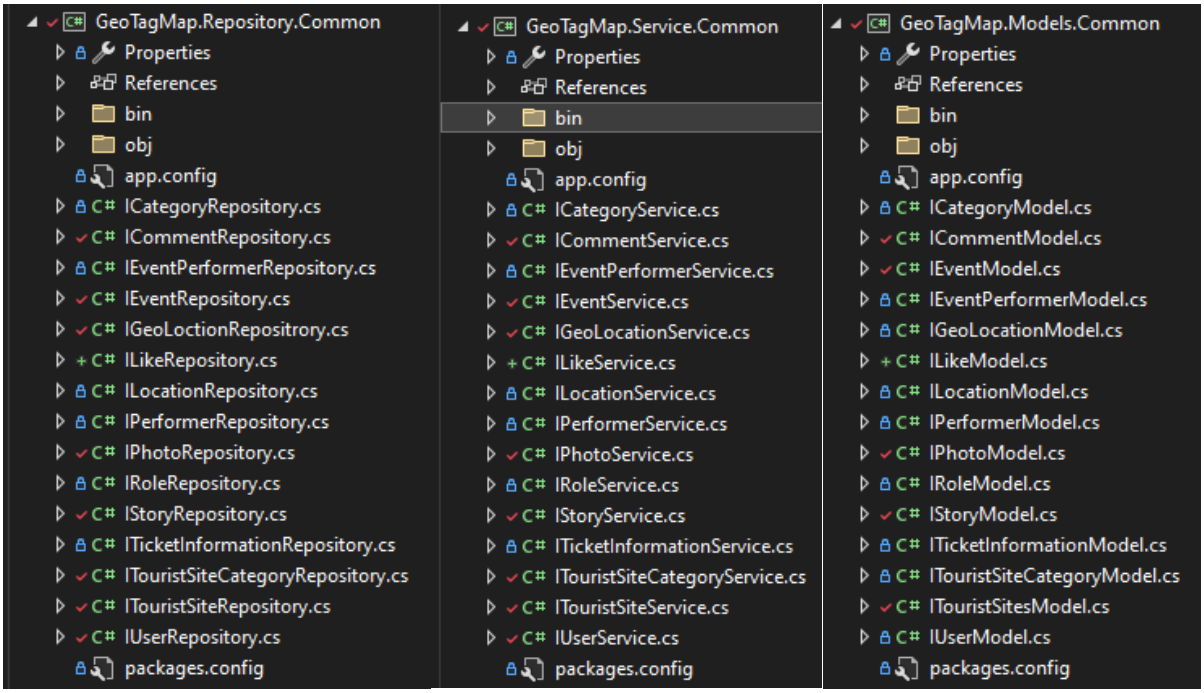
Klase više razine ne bi trebale ovisiti o klasama niže razine, već obje trebaju ovisiti o apstrakcijama. Tehnika ubrizgavanja ovisnosti oslobađa klasu od njezinih ovisnosti. To se postiže uvođenjem sučelja (eng. interface) kako bi se ona razbila. Time se omogućuje ponovno korištenje koda te smanjenje štete koja se može dogoditi na klasama više razine u slučaju promjene neke od klasa niže razine. Cilj ubrizgavanja ovisnosti je ukloniti ovisnost odvajanjem dijela za kreiranje objekta i dijela za upotrebu objekta.

Postoje četiri osnovne uloge koje klase trebaju ispuniti kako bi se pravilno uspostavilo ubrizgavanje ovisnosti.

- Usluga – klasa niže razine koja ima funkcionalnosti koje klasa više razine želi koristiti.
- Klijent – klasa više razine koja želi iskoristiti funkcionalnosti klase niže razine.
- Sučelje – implementira ga klasa niže razine, a koristi ga klasa više razine.
- Injektor – služi za stvaranje instance klase niže razine i ubrizgava ga u klijenta.

Prvi korak postizanja tehnike ubrizgavanje ovisnosti unutar aplikacije se ostvaruje tako da svaki sloj koji je naveden, osim prezentacijskog sloja, također ima sučelje kojeg taj sloj mora

implementirati (slika 4.3). Drugim riječima klasa unutar *Repository* sloja, *CategoryRepository* mora implementirati sučelje unutar *Repository.Common* sloja, *ICategoryRepository*.



Slika 4.3 *GeoTagMap struktura Repozitorij, Servis i Model sučelja*

Zatim se uspostavlja autofac biblioteka koja omogućuje učinkovito upravljanje ovisnostima unutar aplikacije. Njega uspostavljamo unutar *Application_Start* klase s dvije metode prva kreira i konfigurira autofac kontejner, dok druga linija se koristi za upravljanje ovisnostima (slika 4.4).

```
var container = ContainerConfig.ConfigureContainer();

GlobalConfiguration.Configuration.DependencyResolver = new AutofacWebApiDependencyResolver(container);
```

Slika 4.4 *Autofac konfiguracija*

Nakon toga uspostavlja se klasa, odnosno kontejner *ContainerConfig* koja se poziva u *Application_Start* klasi. Objekt *ContainerBuilder* koristi se za registraciju svih ovisnosti. Registrira se s pomoću *builder.RegisterType<T>()* metode (slika 4.5). Tako se registriraju unutar njega sve klase i sučelja svih repozitorija, servisa i kontrolera. Kada bi neka od klasa zatražila instancu neke klase na drugom sloju, kao na primjer ako bi se iz kontrolera zatražila *IUserService* instanca, registracija funkcionira tako da će ona, odnosno kontejner vratiti *UserService* instancu.

```
1 reference
public static class ContainerConfig
{
    1 reference
    public static IContainer ConfigureContainer()
    {
        var builder = new ContainerBuilder();

        builder.RegisterType<UserController>();
        builder.RegisterType<UserRepository>().As<IUserRepository>();
        builder.RegisterType<UserService>().As<IUserService>();

        builder.RegisterType<RoleController>();
        builder.RegisterType<RoleService>().As<IRoleService>();
        builder.RegisterType<RoleRepository>().As<IRoleRepository>();

        builder.RegisterType<EventController>();
        builder.RegisterType<EventService>().As<IEventService>();
        builder.RegisterType<EventRepository>().As<IEventRepository>();
```

Slika 4.5 *ContainerConfig klasa*

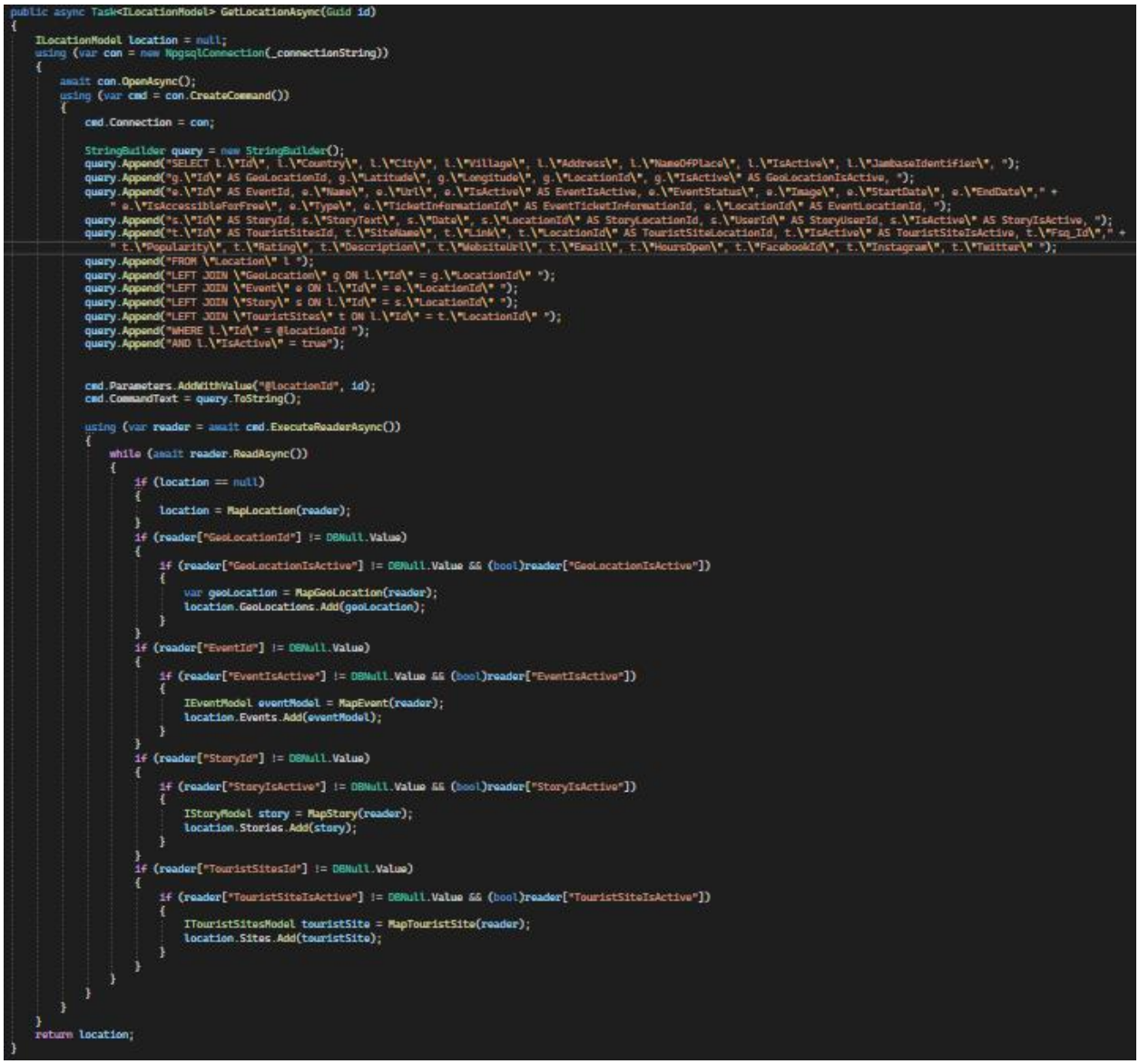
4.2.3. Repozitorij

Repository je sloj koji ima direktnu komunikaciju s bazom podataka putem SQL naredbi. Svaka klasa unutar repozitorija mora implementirati odgovarajuće sučelje unutar *Repository.Common* sloja kako bi ubrizgavanje ovisnosti ispravno funkcioniralo.

Prvo se dohvaća konekcijski *string* iz konfiguracije aplikacije kako bi se uspostavila veza s bazom podataka. Zatim se implementiraju funkcije za različite funkcionalnosti kao na primjer: dohvaćanje svih podataka iz tablice, brisanje postojećeg podatka, dohvaćanje jednog podatka ili izmjena postojećeg podatka.

Na slici 4.6 je prikazan primjer dohvaćanja specifične lokacije koristeći *id* parametar koji se predaje funkciji. Funkcija koristi *Npgsql* za povezivanje s PostgreSQL bazom podataka. Koristi se *using* za definiranje bloka u kojem objekt živi te na kraju kada se zatvori taj blok objekt se automatski oslobađa. *GetLocationAsync* sadržava tri *using* bloka. Prvi se koristi za otvaranje veze s bazom podataka koristeći *NpgsqlConnection*. Drugi *NpgsqlCommand* se koristi za izvršavanje različitih SQL naredbi kao što su *SELECT, INSERT, UPDATE, DELETE* i druge. Treći blok je *NpgsqlDataReader* koji iščitava podatke red po red koji su dohvaćeni iz baze podataka.

SQL naredba na slici 4.6 formirana je tako da dohvaća preko *SELECT* naredbe sve podate iz tablice lokacija. Također koristeći *LEFT JOIN* dohvaća i podatke svih tablica koje su povezane s lokacijom. Ključna riječ *LEFT JOIN* spaja tablicu lokacije s tablicama geolokacija, događaj, priča, turistička lokacija preko primarnih i stranih ključeva tako da uključi sve zapise iz lokacije čak i ako nema odgovarajućeg zapisa u drugim tablicama.



```
public async Task<ILocationModel> GetLocationAsync(Guid id)
{
    ILocationModel location = null;
    using (var con = new NpgsqlConnection(_connectionString))
    {
        await con.OpenAsync();
        using (var cmd = con.CreateCommand())
        {
            cmd.Connection = con;

            StringBuilder query = new StringBuilder();
            query.Append("SELECT l.\"Id\", l.\"Country\", l.\"City\", l.\"Village\", l.\"Address\", l.\"NameOfPlace\", l.\"IsActive\", l.\"JambaseIdentifier\", ");
            query.Append("g.\"Id\" AS GeoLocationId, g.\"Latitude\", g.\"Longitude\", g.\"LocationId\", g.\"IsActive\" AS GeoLocationIsActive, ");
            query.Append("e.\"Id\" AS EventId, e.\"Name\", e.\"Url\", e.\"IsActive\" AS EventIsActive, e.\"EventStatus\", e.\"Image\", e.\"StartDate\", e.\"EndDate\"," +
                " e.\"IsAccessibleForFree\", e.\"Type\", e.\"TicketInformationId\" AS EventTicketInformationId, e.\"LocationId\" AS EventLocationId, ");
            query.Append("s.\"Id\" AS StoryId, s.\"StoryText\", s.\"Date\", s.\"LocationId\" AS StoryLocationId, s.\"UserId\" AS StoryUserId, s.\"IsActive\" AS StoryIsActive, ");
            query.Append("t.\"Id\" AS TouristSitesId, t.\"SiteName\", t.\"Link\", t.\"LocationId\" AS TouristSiteLocationId, t.\"IsActive\" AS TouristSiteIsActive, t.\"Faq_Id\"," +
                " t.\"Popularity\", t.\"Rating\", t.\"Description\", t.\"WebsiteUrl\", t.\"Email\", t.\"HoursOpen\", t.\"FacebookId\", t.\"Instagram\", t.\"Twitter\" ");
            query.Append("FROM \"Location\" l ");
            query.Append("LEFT JOIN \"GeoLocation\" g ON l.\"Id\" = g.\"LocationId\" ");
            query.Append("LEFT JOIN \"Event\" e ON l.\"Id\" = e.\"LocationId\" ");
            query.Append("LEFT JOIN \"Story\" s ON l.\"Id\" = s.\"LocationId\" ");
            query.Append("LEFT JOIN \"TouristSites\" t ON l.\"Id\" = t.\"LocationId\" ");
            query.Append("WHERE l.\"Id\" = @locationId ");
            query.Append("AND l.\"IsActive\" = true");


            cmd.Parameters.AddWithValue("@locationId", id);
            cmd.CommandText = query.ToString();

            using (var reader = await cmd.ExecuteReaderAsync())
            {
                while (await reader.ReadAsync())
                {
                    if (location == null)
                    {
                        location = MapLocation(reader);
                    }
                    if (reader["GeoLocationId"] != DBNull.Value)
                    {
                        if (reader["GeoLocationIsActive"] != DBNull.Value && (bool)reader["GeoLocationIsActive"])
                        {
                            var geoLocation = MapGeoLocation(reader);
                            location.GeoLocations.Add(geoLocation);
                        }
                    }
                    if (reader["EventId"] != DBNull.Value)
                    {
                        if (reader["EventIsActive"] != DBNull.Value && (bool)reader["EventIsActive"])
                        {
                            IEventModel eventModel = MapEvent(reader);
                            location.Events.Add(eventModel);
                        }
                    }
                    if (reader["StoryId"] != DBNull.Value)
                    {
                        if (reader["StoryIsActive"] != DBNull.Value && (bool)reader["StoryIsActive"])
                        {
                            IStoryModel story = MapStory(reader);
                            location.Stories.Add(story);
                        }
                    }
                    if (reader["TouristSitesId"] != DBNull.Value)
                    {
                        if (reader["TouristSiteIsActive"] != DBNull.Value && (bool)reader["TouristSiteIsActive"])
                        {
                            ITouristSitesModel touristSite = MapTouristSite(reader);
                            location.Sites.Add(touristSite);
                        }
                    }
                }
            }
        }
    }
    return location;
}
```

Slika 4.6 *GetLocationAsync funkcija unutar LocationRepository*

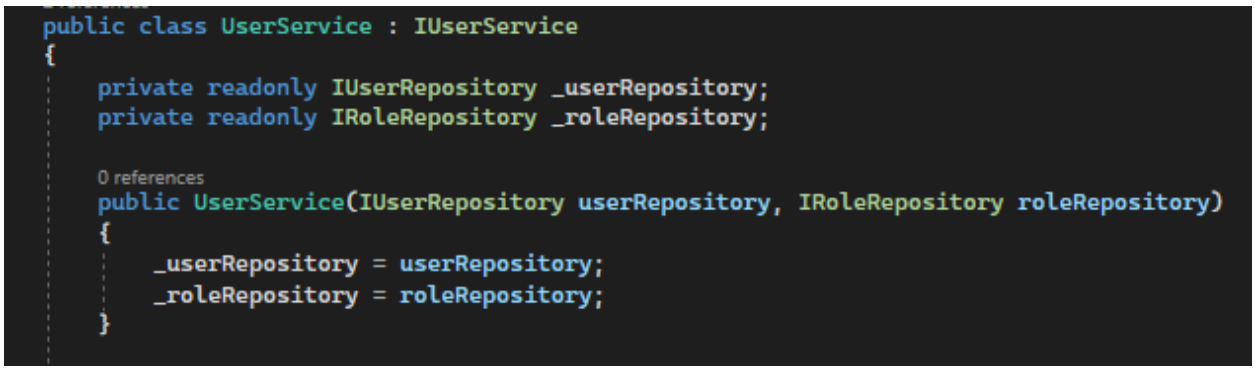
Nakon sastavljene SQL naredbe, s pomoću *NpgsqlDataReader* objekta svi podaci koji su dohvaćeni iz baze podataka čitaju se unutar *while* petlje te se mapiraju. Prolaskom kroz *while* petlju provjeravaju se jedinstveni identifikatori, odnosno primarni i strani ključevi. Ako ti identifikatori nisu prazni i ako nisu izbrisani iz baze, odnosno ako im aktivnost nije negativna pozivaju se metode za mapiranje modela (slika 4.7).

Slika 4.7 *Funkcije mapiranja lokacije i geolokacije*

4.2.4. Servis

Sloj poslovne logike ili *Servis* implementira sučelja koja se nalaze u *Servis.Common* sloju, a ta sučelja definiraju ponašanje servisa. Tako, ako u *Service* sloju postoji klasa *UserService* ona mora implementirati *IUserService* koji se nalazi u *Service.Common*. On je zadužen za obradu podataka koji pristižu iz *WebApi* sloja.

Na slici 4.8 prikazan je *UseService* koji u početku dohvaća instance za *RoleRepository* i *UserRepository*.



Slika 4.8 *UserService instance za repozitorij*

Unutar *UserService* klase postavljene su metode za registraciju, dohvaćanje svih korisnika, dohvaćanje korisnika po jedinstvenom identifikatoru, validaciju korisnika prilikom prijave u aplikaciju i druge.

RegisterUserAsync je metoda koja se koristi za registraciju korisnika u sustav (slika 4.9). Metoda prima kao parametar *user*, a to je instanca objekta *UserModel* koji implementira sučelje *IUserModel* te taj objekt sadržava sve informacije o korisniku. Prvo se poziva metoda *FindRoleByTypeAsync* u *RoleRepository*-u kako bi se pronašao *id* uloge *User*. Zatim se *user* objektu dodjeljuje u *roleId* jedinstveni identifikator koji je dohvaćen s repozitorija.

```
2 references
public async Task RegisterUserAsync(IUserModel user)
{
    Guid roleId = await _roleRepository.FindRoleByTypeAsync("User");
    user.RoleId = roleId;
    var validationContext = new ValidationContext(user, serviceProvider: null, items: null);
    Validator.ValidateObject(user, validationContext, validateAllProperties: true);

    if (await IsUsernameTakenAsync(user.Username))
    {
        throw new ArgumentException("Username is already taken.");
    }

    if (await IsEmailTakenAsync(user.Email))
    {
        throw new ArgumentException("Email is already taken.");
    }

    user.Password = HashPassword(user.Password);

    await _userRepository.AddUserAsync(user);
}
```

Slika 4.9 *RegisterUserAsync funkcija u UserService klasi*

Klasa *Validator* se koristi kako bi se izvršila validacija objekta *user*. Svojstva koja su navedena u klasi *UserModel* prate pravila, odnosno atribute koji su postavljeni iznad njih. Atributi su koristan alat unutar .NET okvira koji pruža mogućnost dodavanja metapodataka na klase, metode, svojstva i druge članove. Koriste se za postavljanje dodatnih informacija koje mogu biti korištene tijekom izvršavanja programa. Obavljaju raznolike funkcionalnosti kao što su: validacija, formatiranje, kontrola ponašanja i druge.

Nakon validacije unesenih podataka provjeravaju se *Username* i *Email* vrijednosti. Važno je da oni uvijek budu jedinstveni, odnosno da nemaju već iste takve vrijednosti unutar baze podataka.

Kada se sva validacija podataka izvrši preostaje još samo jedan korak, a to je zaštita lozinke (slika 4.10). Za izvršavanje te funkcionalnosti upotrebljava se *BCrypt* klasa.

```
2 references
private string HashPassword(string password)
{
    string hashedPassword = BCrypt.Net.BCrypt.HashPassword(password);

    return hashedPassword;
}
```

Slika 4.10 *HashPassword metoda u UserService klasi*

UserService klasa sadržava i metodu *ValidateUserAsync* koja se koristi prilikom prijave korisnika u aplikaciju (slika 4.11). Prvo se dohvaća korisnik s repozitorija koristeći *_userRepository* instancu koja poziva *FindUserAsync* metodu. Korisnikovi podaci se u repozitoriju dohvaćaju s baze podataka po *Username* svojstvu jer je svaki *Username* jedinstven. Nakon toga provjerava se postoji li takav korisnik. Ako postoji, pomoću *BCrypt* klase i njene metode *Verify* uspoređuju se unesena lozinka i lozinka koja je u bazi podataka te ako se podudaraju kod prolazi dalje.

```
2 references
public async Task<IUserModel> ValidateUserAsync(UserFilter user)
{
    var currentUser = await _userRepository.FindUserAsync(user);

    if (currentUser != null && BCrypt.Net.BCrypt.Verify(user.Password, currentUser.Password))
    {
        return currentUser;
    }

    return null;
}
```

Slika 4.11 *ValidateUserAsync metoda u UserService klasi*

4.2.5. WebApi

Najviši sloj u višeslojnoj arhitekturi je prezentacijski sloj ili u ovom slučaju *WebApi*. Koristi se za komunikaciju s klijentskom stranom aplikacije. Ovaj sloj sadržava cijelu konfiguraciju aplikacije, kontrolere koji komuniciraju preko API poziva, autorizaciju, raspoređivanje poslova te također sadržava i *rest view* modele.

Kontroleri su klase koje nasljeđuju *ApiController* te on pruža funkcionalnosti za rukovanje *HTTP* zahtjevima. Oni s klijentske strane primaju *HTTP* zahtjeve poput *GET*, *POST*, *PUT* i *DELETE*, obrađuju te zahtjeve i prosljeđuju ih prema servis sloju korištenjem ubrizgavanja ovisnosti. Osim klasičnih kontrolera za rukovanje turističkim mjestima, lokacijama, slikama i sličnim podacima, uspostavljena su dva kontrolera koja se koriste za dohvaćanje podataka s drugih baza korištenjem API-ja. Jedan se koristi za dohvaćanje podataka o događajima s *JamBase* baze podataka, a drugi se koristi za dohvaćanje podataka o turističkim mjestima s *Foursquare* baze podataka.

Prvo se postavlja atribut *Authorize* koji ograničava pristup metodama unutar kontrolera. Diktira da samo autorizirani korisnici, odnosno oni koji su prijavljeni u aplikaciju mogu pristupiti toj metodi. Iznad metode *GetAsync* prikazan je još jedan primjer *Authorize* atributa, no ovaj puta je postavljen i *Roles* što znači da će samo korisnici koji imaju ulogu *Admin* moći iskoristiti tu metodu. Isto to vrijedi i za ulogu *User*, a može se postaviti i da obadvije uloge imaju pristup istoj metodi.

RoutePrefix atribut se koristi da se definira ruta API-ja unutar cijelog *UserContorllera* i to znači da će rute unutar te klase uvijek započinjati s *api/user*.

Metoda *GetAsync* je *GET* metoda koja dohvaća sve korisnike unutar baze podataka. Prima opcionalne parametre poput *pageSize, pageNumber, sortOrder, orderBy* i druge, te se oni postavljaju u klase koje se nalaze unutar *Common* sloja, kojem svi slojevi na poslužiteljskoj strani mogu pristupiti. *Sorting, paging* i *filtering* koriste se prilikom sastavljanja SQL upita za dohvaćanje svih podataka iz tablice.

Na osnovu primljenih opcionalnih parametara, podaci se filtriraju po korisničkom imenu, imenu ili prezimenu. Sortiranje podataka koristi dva parametra *sortOrder,* koji može biti uzlazni ili silazni, te *orderBy,* koji je u ovoj metodi predefinirano postavljen na *Username*, ali može biti bilo koji stupac unutar tablice. *Paging* koristi parametre *pageSize* i *pageNumber* koji postavljaju koliko će se podataka dohvatiti u jednom pozivu te koju stranicu prikazati.

Kako bi se pozvala metoda za dohvaćanje korisnika na servis sloju, koristi se instanca *_userService*, a kao rezultat se vraća objekt klase *PagingInfo* koja se također nalazi u *Common* sloju. Taj objekt sadrži četiri svojstva: *List*, *TotalSize*, *RPP (eng. Records per Page)* i *PageNumber*, koji daju informacije o listi podataka koja je dohvaćena.

Na slici 4.12 je prikazana metoda iz klase *ApiEventDataInsertController* za dohvaćanje događaja korištenjem API poziva s *JamBase*-a. Dohvaćanje podataka izvršava se unutar beskonačne *while* petlje. Prvo se postavlja *URL* s API ključem, koji je dohvaćen iz konfiguracijske datoteke aplikacije, i s trenutnom stranicom. Nakon toga uspostavlja se *using* blok za *HTTP* klijenta koji se koristi za slanje zahtjeva. Odgovor se sprema u *responseBody* te se nakon toga deserijalizira *JSON* odgovor u *JObject*.

Metoda *ExtractEvents* se koristi za izvlačenje svih podataka iz *JObject*, te ih sprema u objekte koji predstavljaju informacije o događaju, lokaciji, informacijama o kartama, geolokaciji i listi izvođača na tom događaju. Ti podaci se zatim spremaju u listu *totalEvents*.

Nakon toga provjerava se postoji li sljedeća stranica podataka u *JamBase* API odgovoru. Ako sljedeća stranica ne postoji, onda se s pomoću *break* prekida beskonačna petlja.

Podaci koji su prikupljeni u *totalEvents* listi se zatim jedan po jedan spremaju u bazu podataka. Prilikom spremanja koristi se transakcija koja pazi da se svaki dio koda unutar njenog bloka izvrši ispravno. Ako kojim slučajem dođe do greške sve se prekida i nijedan podatak se neće spremiti u

bazu. U suprotnom slučaju, ako sve bude ispravno, koristi će se instance za odgovarajuće klase u servis sloju te će se podaci prosljeđivati dok se ne spreme u bazu.

```
[HttpPost]
[Authorize(Roles = "Admin")]
[Route("jambase")]
1 reference
public async Task<IHttpActionResult> GetJambaseEventsAsync()
{
    try
    {
        var currentPage = 1;
        var totalEvents = new List<(EventModel, TicketInformationModel, List<PerformerModel>, LocationModel, GeoLocation)>();

        while (true)
        {
            var jambaseApiUrl = $"https://www.jambase.com/jb-api/v1/events?apikey={JambaseApiKey}&page={currentPage}";

            using (var client = new HttpClient())
            {
                var response = await client.GetAsync(jambaseApiUrl);
                response.EnsureSuccessStatusCode();

                var responseBody = await response.Content.ReadAsStringAsync();
                var jambaseEvents = JsonConvert.DeserializeObject<JObject>(responseBody);

                var events = ExtractEvents(jambaseEvents);
                totalEvents.AddRange(events);
                if (!HasNextPage(jambaseEvents))
                    break;

                currentPage++;
            }
        }
```

Slika 4.12 *Metoda za dohvaćanje događaja sa JamBase-a u ApiEventDataInsertController klasi*

Poslužiteljska strana aplikacije sadržava sloj *Model* u kojem se nalaze sve model klase poput *UserModel*, *LocationModel* i tako dalje. Modeli predstavljaju entitete u aplikaciji i njihovu strukturu. Oni su zapravo apstraktni prikaz stvarnih podataka.

Na primjer, model korisnika, odnosno *UserModel*, sadržavat će svojstva koja opisuju korisnika, kao što su ime, prezime, korisničko ime, email i tako dalje. Takve klase omogućuju da manipuliramo njihovim objektima i da imamo apstraktnu reprezentaciju stvarnih entiteta unutar koda sa svim svojstvima koja tu stvar opisuju.

Modeli se koriste kroz cijelu poslužiteljsku stranu te prilikom *GET* zahtjeva, odnosno prilikom vraćanja rezultata na korisničku stranu aplikacije vraćaju se modeli ili liste modela. Problem je u tome što neki modeli sadržavaju osjetljive podatke te iz toga razloga nije dobra praksa vratiti cijeli model korisničkoj strani. Na primjer, *UserModel* sadržava osjetljive podatke poput lozinke te je nju potrebno zaštiti. *Rest* i *View* modeli su reprezentacije modela koji se koriste na poslužiteljskoj strani, ali optimizirani za komunikaciju s korisničkom stranom aplikacije.

Rest modeli se koriste kada se podaci šalju na poslužiteljsku stranu s klijentske strane aplikacije. Na primjer, kada se spremaju podaci u bazu podataka prilikom dodavanja događaja ili prilikom

registracije korisnika podaci se šalju u *Rest* modelu te se na kontroleru mapiraju u modele koji se koriste na cijeloj poslužiteljskoj strani (slika 4.13). *Rest* model neće sadržavati svojstva poput *IsActive*, *DateCreated*, *DateUpdated*, *CreatedBy*, *UpdatedBy*, dok će ih modeli poput *UserModel* ili *EventModel* imati. Razlog tomu je što se ti podaci popunjavaju na poslužiteljskoj strani. Ako se kreira događaj on će odmah biti aktivan, isto tako ako se korisnik registrira on će odmah biti aktivan te nije potrebno dohvaćati podatak ni o aktivnosti, ni o datumu, ni o identifikatoru korisnika. Svi ti podaci se postavljaju na poslužiteljskoj strani kao što je prikazano na slici 4.13 gdje se svojstvu *IsActive* dodjeljuje vrijednost *true*. Identifikator korisnika za svojstva *CreatedBy* i *UpdatedBy* se dodjeljuje na servis sloju aplikacije gdje se dohvaća *id* prijavljenog korisnika.

Na slici 4.13 prikazan je primjer dohvaćanja podataka o korisniku koji se registrirao na klijentskoj strani. Podaci su dohvaćeni u *Rest* modelu te su se prije slanja na *UserService* mapirali u *UserModel*.

```
[AllowAnonymous]
[HttpPost]
[Route("register")]
0 references
public async Task<HttpResponseMessage> RegisterAsync([FromBody] UserModelRest userRest)
{
    try
    {
        IUserModel user = MapUser(userRest);
        await _userService.RegisterUserAsync(user);

        return Request.CreateResponse(HttpStatusCode.Created, "User registered successfully");
    }
    catch (Exception ex)
    {
        return Request.CreateResponse(HttpStatusCode.InternalServerError, ex.Message);
    }
}

3 references
private IUserModel MapUser(UserModelRest userRest)
{
    return new UserModel
    {
        Id = userRest.Id,
        Username = userRest.Username,
        FirstName = userRest.FirstName,
        LastName = userRest.LastName,
        Email = userRest.Email,
        Password = userRest.NewPassword,
        Image = userRest.Image ?? null,
        IsActive = true,
    };
}
```

Slika 4.13 *PostAsync metoda i mapiranje UserModel-a UserController klase*

View modeli se koriste za slanje podataka s poslužiteljskog dijela aplikacije na klijentsku stranu. Ovaj tip modela štiti podatke koji se ne bi trebali slati ni prikazivati kao što je lozinka.

Koristeći Microsoft OWIN (eng. Open Web Interface for .NET) okruženje implementira se *OAuth* autentifikacija za API. Unutar *Startup* klase uspostavlja se konfiguracija za OWIN posrednički sloj i *OAuth* autentifikaciju.

Atribut koji je postavljen iznad svega postavlja *Startup* klasu kao ulaznu točku za OWIN konfiguraciju.

Metoda za konfiguraciju *OAuth* autentifikacije *ConfigureOAuth* dohvaća *UserService* instancu iz ubrizgavanja ovisnosti te zatim postavlja *OAuthAuthorizationServerOptions*. Prvo postavlja krajnju točku (eng. Endpoint) na kojoj će se nalaziti token. Zatim postavlja instancu klase *ApplicationAuthProvider* koji će rukovati validacijom korisnika. Treća opcija koja se postavlja je vrijeme trajanja tokena te zadnja opcija *AllowInsecureHttp*, koja je postavljena na *false*, omogućuje samo sigurnu *HTTP* komunikaciju. Zadnje dvije linije koda unutar metode pokreću *OAuth* server i *OAuth* bearer token autentifikaciju.

Klasa *ApplicationAuthProvider* izvršava validaciju korisnika i izdaje token. U početku se koristi atribut koji omogućuje CORS (eng. Cross-Origin Resource Sharing) politiku, a to znači da aplikacija može primati upite s bilo kojeg izvora, s bilo kojim zaglavljima i metodama. Dohvaća se instanca *UserService* koja je potrebna za pristup metodi validacije korisnika.

Metoda *GrantResourceOwnerCredentials* ovisno o ispravnosti korisničkog imena i lozinke izdaje token te postavlja korisnikov identitet na cijeli poslužitelj. Provjera ispravnosti podataka za prijavu izvršava se u *UserService* klasi. Ako je korisnik ispravan, odnosno postojan kreira se novi *ClaimsIdentity* objekt. Tome objektu se zatim dodaju različiti *Claim*-ovi poput korisničkog imena, uloge, jedinstvenog identifikatora i email adrese. S pomoću *Protect* metode kreira se token bez kojeg korisnik nema pristup podacima ni funkcionalnostima. Kada se korisnik prijavi u aplikaciju, vraća se pristupni token sa korisnikovim podacima koji su postavljeni u *identity* objektu.

Podatke o događajima je potrebno ažurirati svaki dan te se iz toga razloga postavlja okidač koji će svaka 24 sata ponovno dohvaćati podatke s *JamBase* API-ja. Ova funkcionalnost se ostvaruje s pomoću *Quartz* sustava za raspoređivanje poslova.

Na slici 4.14 prikazana je klasa *JambaseEventsJob* koja implementira *IJob* sučelje iz *Quartz.NET* biblioteke. Ona definira posao koji će se izvršavati u pozadini. Kada se posao pokrene poziva se asinkrona metoda *Execute*. Unutar nje u *resolver* se postavlja trenutni ubrizgavač ovisnosti za *WebApi*. Postavlja se također *using* blok koji kreira novi opseg trajanja za *scope* u kojem se može razriješiti ovisnosti.

U *eventController* se pomoću *scope.Resolve<ApiEventDataInsertController>()* dohvaća instanca za *ApiEventDataInsertController* klasu te se ona koristi za pozivanje metode *GetJambaseEventsAsync* koja služi za dohvaćanje podataka iz *JamBase* API-ja.

```
1 reference
public class JambaseEventsJob : IJob
{
    0 references
    public async Task Execute(IJobExecutionContext context)
    {
        var resolver = GlobalConfiguration.Configuration.DependencyResolver as AutofacWebApiDependencyResolver;
        if (resolver != null)
        {
            using (var scope = resolver.Container.BeginLifetimeScope())
            {
                var eventController = scope.Resolve<ApiEventDataInsertController>();
                IHttpActionResult result = await eventController.GetJambaseEventsAsync();

                if (result is OkNegotiatedContentResult<IEnumerable<(EventModel, TicketInformationModel,
                    List<PerformerModel>, LocationModel, GeoLocation)>> okResult)
                {
                    var events = okResult.Content;
                }
            }
        }
    }
}
```

Slika 4.14 *JambaseEventsJob klasa*

QuartzScheduler klasa sadržava metodu *Start* koja pokreće raspored. Prvo se dohvaća instanca *IScheduler* sučelja koji upravlja zadacima i okidačima. Nakon dohvaćanja instance, ona se odmah koristi za pokretanje raspoređivača. *JambaseEventsJob* posao se kreira korištenjem *JobBuilder* klase i njegove metode *Create*.

Okidač se uspostavlja s pomoću *TriggerBuilder* klase i njegove metode *Create*. Dodatno se pokreću i druge metode *TriggerBuilder* klase kako bi se okidač pravilno uspostavio. *WithIdentity* je metoda koja postavlja jedinstveni identifikator okidača i grupu kojoj taj okidač pripada. *StartNow* metoda aktivira okidač, odmah poslije uspostavljanja raspoređivača. *WithSimpleSchedule* definira raspored koji se izvršava svaka 24 sata i tako u beskonačnost. Na kraju metoda *ScheduleJob* raspoređuje posao u *scheduler* koristeći definirani *trigger*.

4.3. Klijentski dio

Klijentski dio aplikacije sastavljen je korištenjem *Reacta*. Jedna od karakteristika *Reacta* je u tome što se kreirane komponente mogu iskorištavati više puta. Glavna komponenta klijentskog dijela aplikacije je *App* od koje se sve ostale komponente pokreću. Zatim je struktura aplikacije podijeljena na 3 dijela dio u kojem su postavljene sve komponente, dio u kojem se nalaze servisi za komunikaciju s poslužiteljskom stranom te dio u kojem se nalaze sve stilizacije komponenata. Podaci se dohvaćaju putem API poziva prema poslužiteljskoj strani. *GeoTagMap* prikazuje

podatke na globusu zemaljske kugle sa svim geografskim podacima. Za prikaz takvog globusa koristi se *CesiumJS* API.

4.3.1. CesiumJS globus mapa

CesiumJS je *JavaScript* biblioteka otvorenog koda za kreiranje 3D globusa i mapa s najboljim mogućim performansama, preciznošću, vizualnom kvalitetom i lakoćom korištenja. [11]

Na slici 4.15 prikazana je inicijalizacija *Cesium* mape. Prvo se postavlja pristupni token koji je pohranjen u *.env* mapi aplikacije. Zatim se inicijalizira *Viewer,* odnosno *viewerRef.current* se postavlja kao novi *Cesium.Viewer* objekt. Funkcije koje dolaze s inicijalizacijom globusa postavljene su sve na *false* jer nisu potrebne. Sljedeće se uspostavlja objekt *imageryProvider* koji pruža slike za mapu ili globus. S pomoću njega se vizualizira površina Zemlje. Postoji više različitih *imagery providera* u *CesiumJS* kao što su *ArcGisMapServerImageryProvider*, *BingMapsImageryProvider* i drugi, ali u ovoj aplikaciji je korišten *Cesium Ion Imagery Provider* za dohvaćanje slika s *Cesium Ion* servisa korištenjem određenog *Asset* id-a. Nakon što se učita *imageryProvider* objekt on se dodaje u sloj *imageryLayers* trenutnog *viewerRef* objekta i dohvaćene slike se prikazuju na mapi.

Zatim se uspostavlja kamera, odnosno početne postavke kamere s *Cesium.Cartesian3.fromDegrees* funkcijom. Kamera se postavlja na udaljenost od 19 milijuna metara, a globus se centrira na lokaciju s koordinatama dužine (eng. Longitude) 4 i širine (eng. Latitude) 20. Nakon uspostavljanja kamere pokreće se animacija rotacije Zemlje.

```
const initializeMap = async () => {
  Cesium.Ion.defaultAccessToken = process.env.REACT_APP_CESIUM_MAPS_API_KEY;

  viewerRef.current = new Cesium.Viewer(mapContainer.current, {
    animation: false,
    baseLayerPicker: false,
    fullscreenButton: false,
    geocoder: false,
    homeButton: false,
    infoBox: false,
    sceneModePicker: false,
    selectionIndicator: true,
    timeline: false,
    navigationHelpButton: false,
    navigationInstructionsInitiallyVisible: false,
  });

  try {
    const imageryProvider = await Cesium.IonImageryProvider.fromAssetId(3);
    viewerRef.current.imageryLayers.addImageryProvider(imageryProvider);
  } catch (error) {
    console.error("Error adding imagery provider:", error);
  }

  viewerRef.current.camera.setView({
    destination: Cesium.Cartesian3.fromDegrees(4, 20, 19000000),
  });

  startEarthRotationAnimation();

  setIsViewerReady(true);
};
```

Slika 4.15 *Inicijalizacija globusa koristeći CesiumJS API*

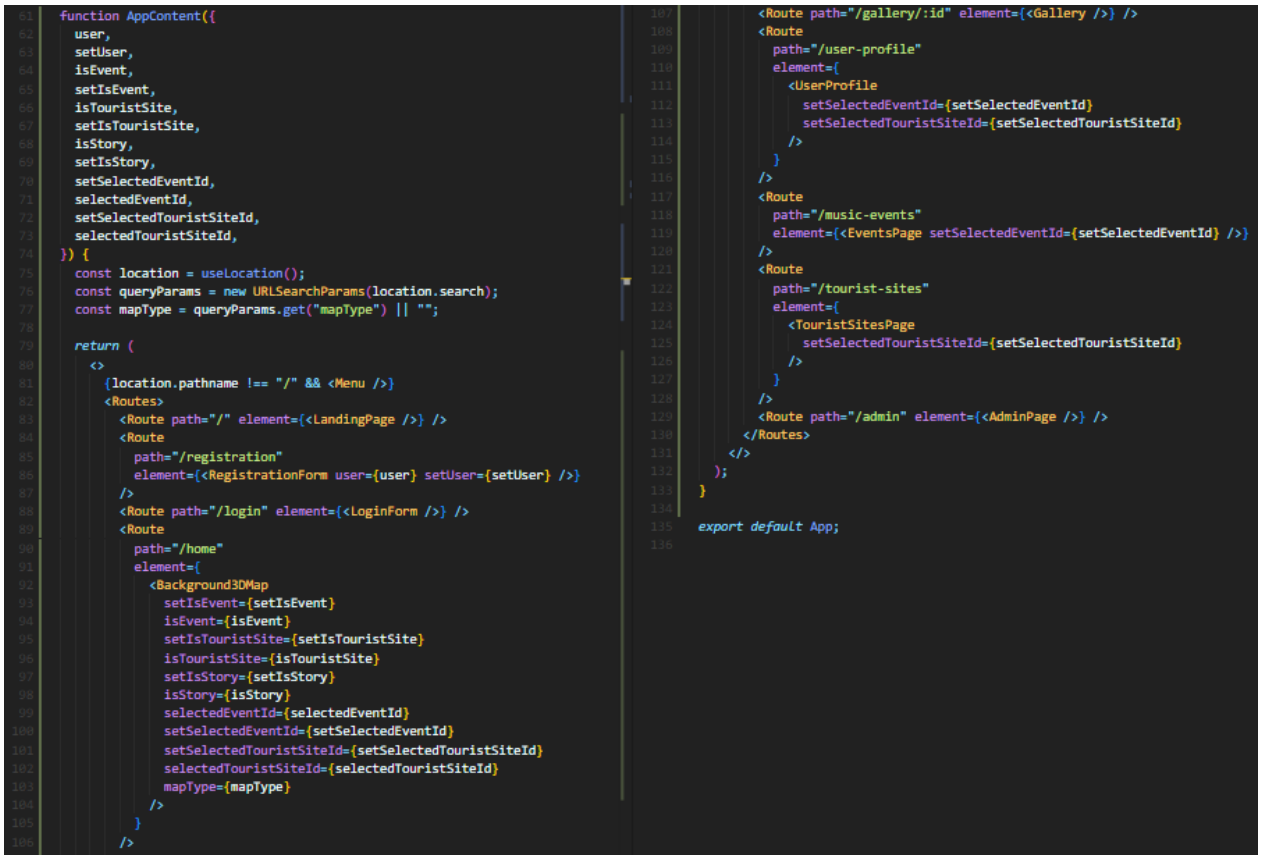
4.3.2. Komponente

Kao što je već spomenuto najvažnija komponenta je *App* od koje se sve pokreće. Unutar nje se postavljaju i definiraju sve rute klijentskog dijela aplikacije. Na slici 4.16 i 4.17 prikazana je *App* komponente. Rute su definirane koristeći *react-router-dom* biblioteku te ona osposobljava navigaciju unutar aplikacije bez osvježavanja stranice. *Router* komponenta omotava cijeli sadržaj aplikacije te omogućava korištenje ruta u njoj.

```
function App() {
  const [user, setUser] = useState({
    username: "",
    firstName: "",
    lastName: "",
    email: "",
    password: "",
  });

  const [isEvent, setIsEvent] = useState(true);
  const [isTouristSite, setIsTouristSite] = useState(false);
  const [isStory, setIsStory] = useState(false);
  const [selectedEventId, setSelectedEventId] = useState(uuidv4());
  const [selectedTouristSiteId, setSelectedTouristSiteId] = useState(uuidv4());

  return (
    <div className="App">
      <header className="App-header">
        <Router>
          <AppContent
            user={user}
            setUser={setUser}
            isEvent={isEvent}
            setIsEvent={setIsEvent}
            isTouristSite={isTouristSite}
            setIsTouristSite={setIsTouristSite}
            isStory={isStory}
            setIsStory={setIsStory}
            setSelectedEventId={setSelectedEventId}
            selectedEventId={selectedEventId}
            setSelectedTouristSiteId={setSelectedTouristSiteId}
            selectedTouristSiteId={selectedTouristSiteId}
          />
        </Router>
      </header>
    </div>
  );
}
```

Slika 4.16 *App komponenta*

Unutar *Routes* komponente se definiraju rute. *Route* komponenta definira put i komponentu koja će biti prikazana kada *URL* bude odgovarao tom putu. Na slici 4.17 prikazane su sve rute unutar klijentskog dijela aplikacije. Na primjer, kada se uspostavi aplikacija i kada se pokrene poslužiteljski dio i klijentski dio, prva ruta će biti „ / “ te će se prikazati *LandingPage* komponenta. Zatim nakon prijave u aplikaciju sljedeća ruta će biti „/home“ i onda se prikazuje *Background3DMap* komponenta i tako dalje.



```
function AppContent({
  user,
  setUser,
  isEvent,
  setIsEvent,
  isTouristSite,
  setIsTouristSite,
  isStory,
  setIsStory,
  setSelectedEventId,
  selectedEventId,
  setSelectedTouristSiteId,
  selectedTouristSiteId,
}) {
  const location = useLocation();
  const queryParams = new URLSearchParams(location.search);
  const mapType = queryParams.get("mapType") || "";

  return (
    <>
      {location.pathname !== "/" && <Menu />}
      <Routes>
        <Route path="/" element={<LandingPage />} />
        <Route
          path="/registration"
          element={<RegistrationForm user={user} setUser={setUser} />}
        />
        <Route path="/login" element={<LoginForm />} />
        <Route
          path="/home"
          element={
            <Background3DMap
              setIsEvent={setIsEvent}
              isEvent={isEvent}
              setIsTouristSite={setIsTouristSite}
              isTouristSite={isTouristSite}
              setIsStory={setIsStory}
              isStory={isStory}
              selectedEventId={selectedEventId}
              setSelectedEventId={setSelectedEventId}
              setSelectedTouristSiteId={setSelectedTouristSiteId}
              selectedTouristSiteId={selectedTouristSiteId}
              mapType={mapType}
            />
          }
        />
        <Route path="/gallery/:id" element={<Gallery />} />
        <Route
          path="/user-profile"
          element={
            <UserProfile
              setSelectedEventId={setSelectedEventId}
              setSelectedTouristSiteId={setSelectedTouristSiteId}
            />
          }
        />
        <Route
          path="/music-events"
          element={<EventsPage setSelectedEventId={setSelectedEventId} />}
        />
        <Route
          path="/tourist-sites"
          element={
            <TouristSitesPage
              setSelectedTouristSiteId={setSelectedTouristSiteId}
            />
          }
        />
        <Route path="/admin" element={<AdminPage />} />
      </Routes>
    </>
  );
}

export default App;
```

Slika 4.17 *Funkcija AppContent unutar App komponente*

Kao što je već spomenuto, pokretanjem aplikacije prvo se prikazuje *LandingPage* komponenta. Ova komponenta korisniku nudi, osim informacijskih podataka, samo dvije opcije. Prva je prijava u aplikaciju, a druga je registracija korisnika.

Odabirom opcije za registraciju korisnika, učitava se *RegistrationForm* komponenta. Ova komponenta predstavlja formu koju korisnik ispunjava, nakon čega ispunjene podatke šalje na poslužiteljski dio putem *UserService* servisa.

Ako korisnik odabere prijavu, otvorit će se *LoginForm* koji od korisnika traži da unese korisničko ime i lozinku. Nakon unosa podataka poziva se metoda *handleSubmit* u kojoj se uneseni podaci šalju na poslužiteljski dio. Ako je prijava bila uspješna dohvaćaju se korisnikovi podaci i razina pristupa. Funkcija provjerava koju ulogu korisnik ima i usmjerava ga na *AdminPage* komponentu ako je njegova uloga *Admin*. Ako je uloga *User* onda ga usmjerava na *Background3DMap* komponentu i taj korisnik ne može pristupiti *AdminPage* komponenti

Korisnici s *Admin* ulogom pristupaju *AdminPage* komponenti na kojoj se provjerava njihova uloga. Ako uloga nije *Admin* odmah se vodi na rutu „/home“, odnosno na *Background3DMap* komponentu (slika 4.18).

```
useEffect(() => {
  const userRole = getUserRole();
  if (userRole !== "Admin") {
    navigate("/home");
  } else {
    getLoggedAdminUser();
  }
}, [navigate]);
```

Slika 4.18 *useEffect kuka za provjeravanje razine pristupa*

Administrator na raspolaganju ima mogućnost upravljanja podacima priča, korisnika i komentara. Podaci se prikazuju u tablici u kojoj su postavljeni i akcijski gumbovi poput brisanja. Kod tablice korisnika postoji i gumb za postavljanje određenog korisnika kao administratora. Priče i komentari mogu biti prijavljeni od strane svakog korisnika te se to prikazuje na tablici. Pružena je mogućnost filtriranja podataka te administrator može odabrati da mu se u tablici prikažu samo prijavljene priče ili prijavljeni komentari.

Komponenta *Background3DMap* prima različita svojstva, odnosno *props* koji se koriste u njoj. *UseState* kuke za stanja *isEvent*, *isTouristSite* i *isStory* postavljene su u *App* komponenti. Ovo je zato što se ta stanja mijenjaju na više mjesta u aplikaciji i uvijek ih je potrebno postaviti na najvišu komponentu kako bi se njihovo ažuriranje olakšalo. *Background3DMap* komponenta sadrži inicijalizaciju *CesiumJS* globusa Zemlje. *MapDataPicker* komponenta se koristi za ažuriranje stanja *isEvent*, *isToursitSite* i *isStory*.

Background3DMap prikazuje trodimenzionalni globus Zemlje. Ovisno o vrijednostima navedenih stanja, pozivaju se različite komponente koje postavljaju sadržaj na globus. Na primjer, ako je vrijednost za stanje *isEvent true*, poziva se komponenta *EventMap*. Ako je vrijednost *false* onda se traži stanje koje ima vrijednost *true* te se zatim poziva odgovarajuća komponenta, bilo *TouristeSiteMap* ili StoryMap.

EventMap, *TouristSiteMap* i *StoryMap* komponente postavljaju sadržaj na globus Zemlje. Sadržaj koji se postavlja su oznake. Oni se postavljaju na kartu s pomoću podataka koji su dohvaćeni s poslužiteljske strane, točnije podaci koji se upotrebljavaju su geografska dužina i širina.

Učitavanjem jedne od ovih komponenti na primjer, *EventMap* komponente prvo se dohvaćaju događaji putem *EventService* servisa (slika 4.19). Funkcija *fetchAndFilterEvents* nakon

dohvaćanja svih filtriranih događaja izdvaja sve lokacije i geolokacije i sprema ih u stanja *locations* i *geolocations*.

```
const fetchAndFilterEvents = useCallback(async () => {
  try {
    const response = await EventService.getEventsFiltered(filterForm);

    if (response && Array.isArray(response.List)) {
      const eventsList = response.List;
      setEvents(eventsList);

      const uniqueLocationsMap = new Map();
      eventsList.forEach((event) => {
        if (!uniqueLocationsMap.has(event.Location.Id)) {
          uniqueLocationsMap.set(event.Location.Id, {
            ...event.Location,
            Events: [],
          });
        }
        uniqueLocationsMap.get(event.Location.Id).Events.push(event);
      });
      const uniqueLocations = Array.from(uniqueLocationsMap.values());
      setLocations(uniqueLocations);

      const geoLocations = eventsList.flatMap((event) => {
        return event.GeoLocations.map((geoLocation) => ({
          ...geoLocation,
          Location: { ...event.Location },
        }));
      });
      setGeoLocations(geoLocations);
    } else {
      setEvents([]);
      setGeoLocations([]);
    }
  } catch (error) {
    console.error("Error fetching events:", error);
    setEvents([]);
    setGeoLocations([]);
  }
}, [filterForm]);
```

Slika 4.19 *Funkcija fetchAndFilterEvents EventMap komponente*

Funkcija *addPinsToMap* postavlja oznake na kartu koristeći geolokacijske podatke dužine i širine. Oznake se neće uspostavljati sve dok se objekt *viewerRef* ne inicijalizira (slika 4.20). Nakon toga dohvaća se *MapPin* komponenta u kojoj se nalazi *svg* za oznaku koja će prikazivati određenu lokaciju. Sljedeće se koristi lista *locations* kako bi se prošlo kroz svaku lokaciju unutar te liste. Zatim se prolazi kroz sve događaje na toj lokaciji i na kraju kroz sve geolokacije za svaki događaj. Za svaku geografsku lokaciju stvara se objekt *pin* koji sadrži jedinstveni identifikator, ime, poziciju i sliku oznake te se sve oznake spremaju u listu *pins*.

Na kartu se oznake dodaju koristeći *viewerRef.current.entities.add*.

```
if (viewerRef.current) {
  const customPinSvg = ReactDOMServer.renderToStaticMarkup(<MapPin />);
  const pinDataUrl = `${btoa(customPinSvg)}`;

  const pins = locations.flatMap((location) =>
    location.Events.flatMap((event) =>
      event.GeoLocations.map((geoLocation) => ({
        id: Cesium.createGuid(),
        name: "Custom Pin",
        position: Cesium.Cartesian3.fromDegrees(
          geoLocation.Longitude,
          geoLocation.Latitude
        ),
        billboard: {
          image: pinDataUrl,
          verticalOrigin: Cesium.VerticalOrigin.BOTTOM,
        },
        Events: [event],
      }))
    )
  );

  const addedPinsEntities = pins.map((pin) =>
    viewerRef.current.entities.add(pin)
  );
  setAddedPins(addedPinsEntities);

  viewerRef.current.selectionIndicator.viewModel.selectionIndicatorElement.style.visibility =
    "hidden";

  const handler = new Cesium.ScreenSpaceEventHandler(
    viewerRef.current.scene.canvas
  );
```

Slika 4.20 *Postavljanje oznaka na kartu*

Slika 4.21 prikazuje dio koda koji postavlja *ScreenSpaceEventHandler* za detekciju klikova na mapi. Kada se klikne na neku od oznaka, funkcija provjerava status klika. Ako se neka oznaka kliknula učitava se *Event* komponenta na kojoj se nalaze sve informacije o tom specifičnom događaju, svi komentari, izvođači i tako dalje.

```
const handler = new Cesium.ScreenSpaceEventHandler(
  viewerRef.current.scene.canvas
);

handler.setInputAction((click) => {
  const pickedObject = viewerRef.current.scene.pick(click.position);
  if (
    Cesium.defined(pickedObject) &&
    pickedObject.id &&
    pickedObject.id.Events
  ) {
    const handleClose = () => {
      ReactDOM.unmountComponentAtNode(
        document.getElementById("event-container-root")
      );
    };
    const pickedEvents = pickedObject.id.Events.map((event) => (
      <Event key={event.id} event={event} />
    ));

    ReactDOM.render(
      <div className="event-overlay">
        <Event slides={pickedEvents} onClose={handleClose} />
      </div>,
      document.getElementById("event-container-root")
    );
  }
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);

viewerRef.current.screenSpaceEventHandler.removeInputAction(
  Cesium.ScreenSpaceEventType.LEFT_DOUBLE_CLICK
);
```

Slika 4.21 *Učitavanje Event komponente nakon klika na neku oznaku*

4.3.3. Servisi

Servisi unutar klijentskog dijela aplikacije koriste se za komunikaciju s poslužiteljskom stranom aplikacije. U njima su s pomoću *axios* biblioteke omogućeni *HTTP* zahtjevi. Sadržavaju metode za dohvaćanje svih podataka, za dohvaćanje podataka po jedinstvenom identifikatoru, za brisanje podataka i za ažuriranje podataka.

Na slici 4.22 prikazana je funkcija *getEventsFiltered EventService* servisa. Funkcija prvo dohvaća pristupni token koji će se postaviti u *Authorization* zaglavlje. Nakon toga provjerava koji se podaci nalaze u *formData* objektu. Svaki se podatak provjerava zasebno u *if* uvjetu. Ako podatak postoji dodaje se u *URL* adresu koja se kasnije koristi u API ruti za pozivanje funkcije za dohvaćanje svih filtriranih događaja na poslužiteljskoj strani.

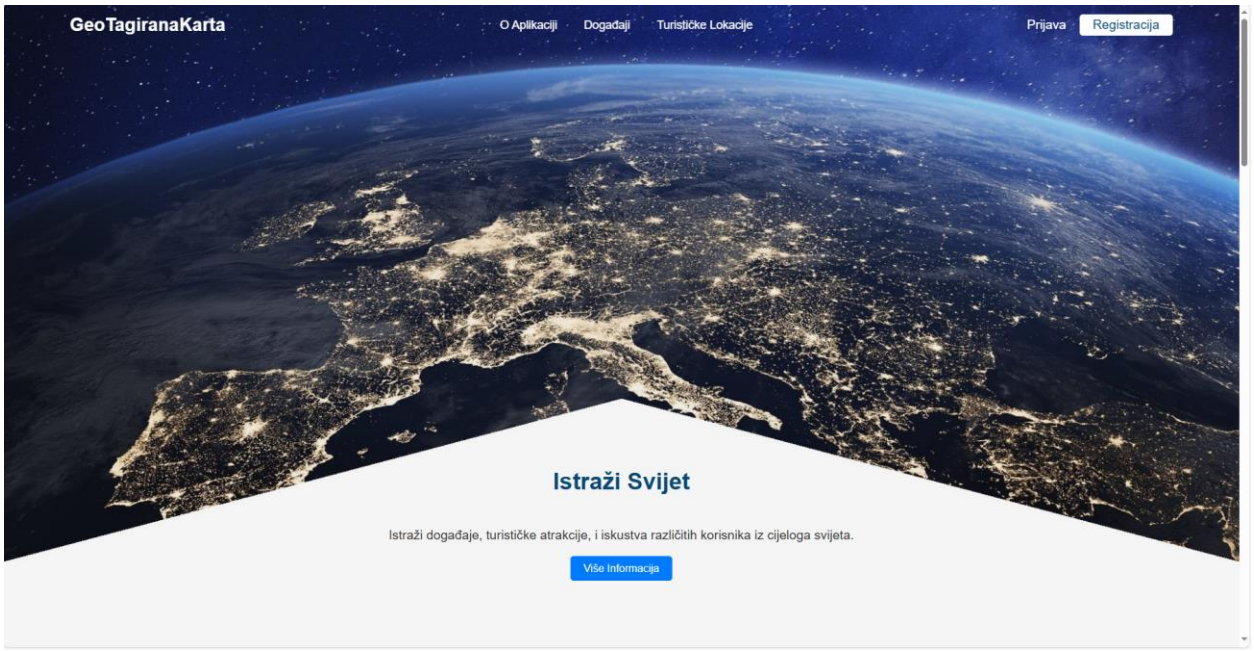
```
getEventsFiltered: async (formData) => {
  try {
    const token = getAccessToken();
    if (!token) {
      console.error("Access token not found");
      return;
    }

    const queryParams = new URLSearchParams();

    if (formData.name) {
      queryParams.append("name", formData.name);
    }

    if (formData.startDate) {
      queryParams.append("startDate", formData.startDate);
    }

    if (formData.status) {
      queryParams.append("status", formData.status);
    }

    if (formData.isAccessibleForFree) {
      queryParams.append("isAccessibleForFree", formData.isAccessibleForFree);
    }
    if (formData.type) {
      queryParams.append("type", formData.type);
    }
    if (formData.searchKeyword) {
      queryParams.append("searchKeyword", formData.searchKeyword);
    }
    if (formData.country) {
      queryParams.append("country", formData.country);
    }
    if (formData.city) {
      queryParams.append("city", formData.city);
    }
    if (formData.sortField) {
      queryParams.append("OrderBy", formData.sortField);
    }

    if (formData.sortOrder) {
      queryParams.append("SortOrder", formData.sortOrder);
    }

    if (formData.pageSize) {
      queryParams.append("pageSize", formData.pageSize);
    }

    if (formData.pageNumber) {
      queryParams.append("pageNumber", parseInt(formData.pageNumber, 10));
    }

    const urlWithParams = `${BASE_URL}?${queryParams.toString()}`;
    const response = await axios.get(urlWithParams, {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });

    return response.data;
  } catch (error) {
    console.error("Error while fetching songs:", error);
    throw error;
  }
},
```

Slika 4.22 *Funkcija getEventsFiltered EventService servisa*

5. UPOTREBA APLIKACIJE

Ovo poglavlje će opisati sve funkcionalnosti i mogućnosti aplikacije. Također, prikazat će se izgled cijele aplikacije.

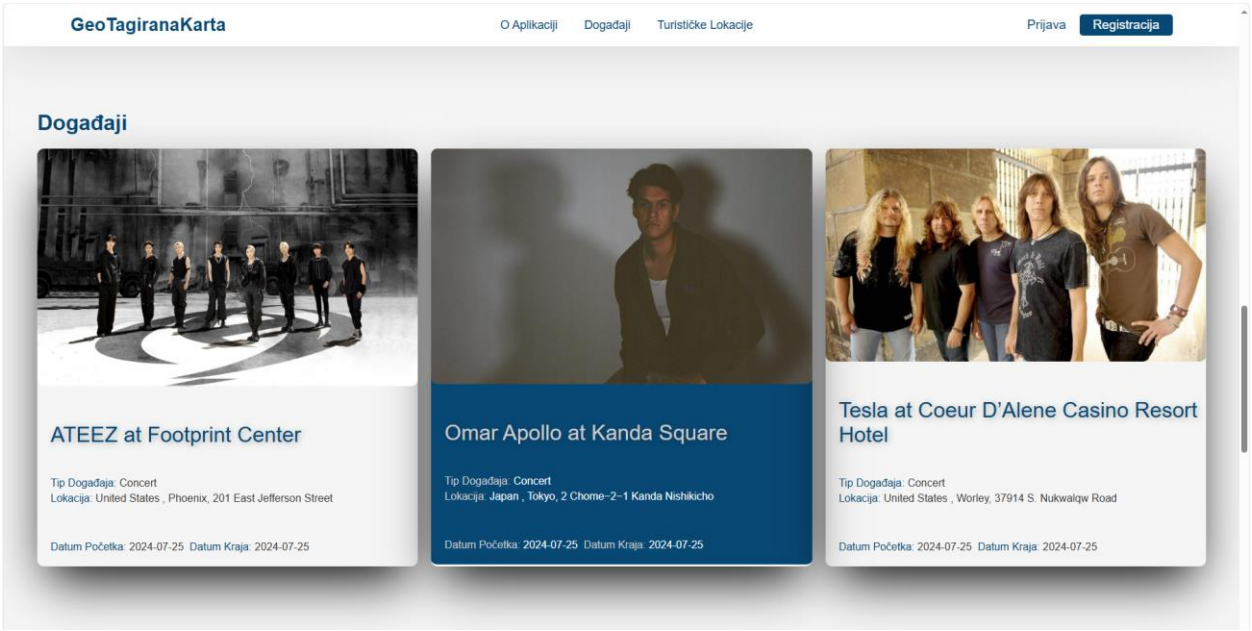
Kao što je već spomenuto nakon pokretanja aplikacije prvo se prikazuje *LandingPage* komponenta, odnosno prikazuje se stranica koja dočekuje sve korisnike i one koji su registrirani u aplikaciju i one koji nisu (slika 5.1).



Slika 5.1 *Početna stranica aplikacije*

U navigacijskoj traci stranice, pružene su mogućnosti koje korisnik može odabrati, no samo prijava i registracija će omogućiti ulazak u aplikaciju. Ostale opcije unutar navigacijske trake, kao što su događaji i turističke lokacije, odvode korisnika do tog dijela stranice.

Na slici 5.2 prikazano je što se dogodi kada korisnik pritisne na opciju događaji unutar navigacijske trake. Prikazane su tri kartice koje sadrže generalne podatke o događajima. Na isti način funkcionira i za turističke lokacije.

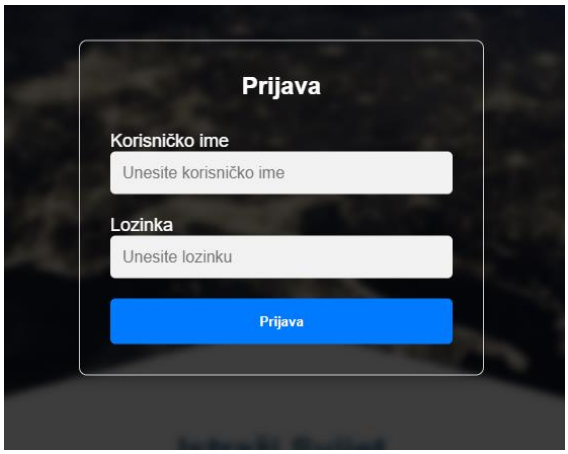


Slika 5.2 *Odabir opcije Događaji na navigacijskoj traci*

Ako korisnik odabere opciju da se registrira u aplikaciju ona ga vodi na obrazac prikazan na slici 5.3. Korisnik ovdje može odabrati da ode na prijavu u aplikaciju ako je već registriran. Obrazac sadržava klasične informacije koje se trebaju ispuniti. Korisničko ime mora biti unikatno te ako već postoji takvo korisničko ime u bazi podataka izbacuje se greška. Email adresa mora biti odgovarajućeg formata da bi se korisnik uspješno registrirao. Postavljanje profilne slike je opcionalno.

Slika 5.3 *Obrazac za registraciju u aplikaciju*

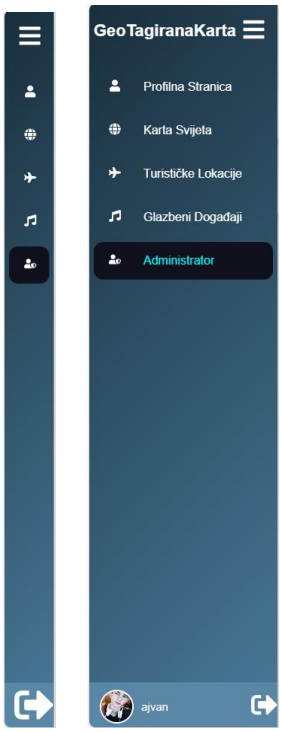
Odabirom opcije prijavi se prikazuje se obrazac za unos korisničkog imena i lozinke (slika 5.4).



Slika 5.4 *Obrazac za prijavu u aplikaciju*

Ako su podaci unijeti u obrazac za prijavu ispravni, korisnik ulazi u aplikaciju. Prvo se provjerava uloga korisnika. Ako je korisnik administrator, aplikacija ga usmjerava na administratorsku stranicu. Ako nije administrator, onda se njega vodi na glavnu stranicu aplikacije.

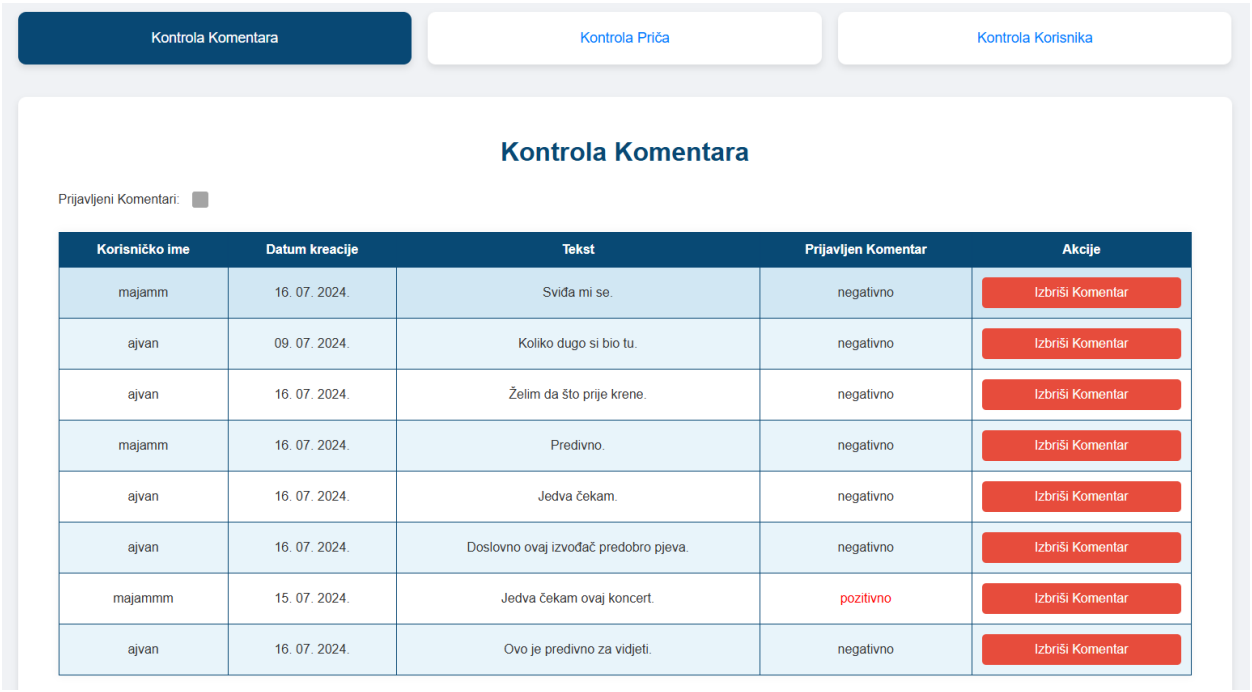
Kroz cijelu aplikaciju postavljena je navigacijska traka s lijeve strane ekrana. Navigacijska traka kada je zatvorena prikazuje samo ikone, a kada se proširi prikazuju se i nazivi stranica koje korisnik može odabrati. Zadnja ikona, odnosno *Administrator* opcija na navigacijskoj traci je prisutna samo ako je korisnik administrator.



Slika 5.5 *Navigacijska traka zatvorena i otvorena*

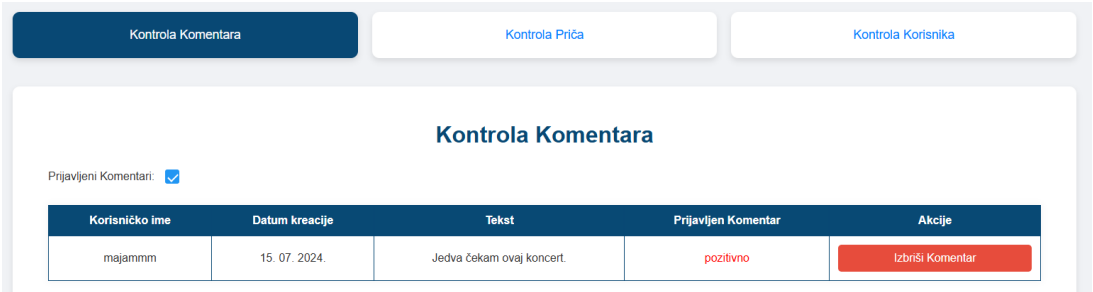
Administratorska stranica pruža odabir između tri opcije za prikazati. Svaka opcija prikazuje podatke unutar tablice u kojoj su također postavljeni gumbovi za izvršavanje neke od radnji.

Prva opcija za odabrati je *Kontrola Komentara* (slika 5.6) u kojoj su prikazani svi komentari. Ovdje je moguće pregledati komentare svih priča, događaja i turističkih mjesta. Komentari mogu biti prijavljeni. Iz tog razloga je postavljen stupac unutar tablice u kojem se prikazuje status prijavljenosti. Ako je komentar prijavljen u redu tog stupca će crvenim slovima pisati pozitivno u suprotnom prikazuje se negativno.



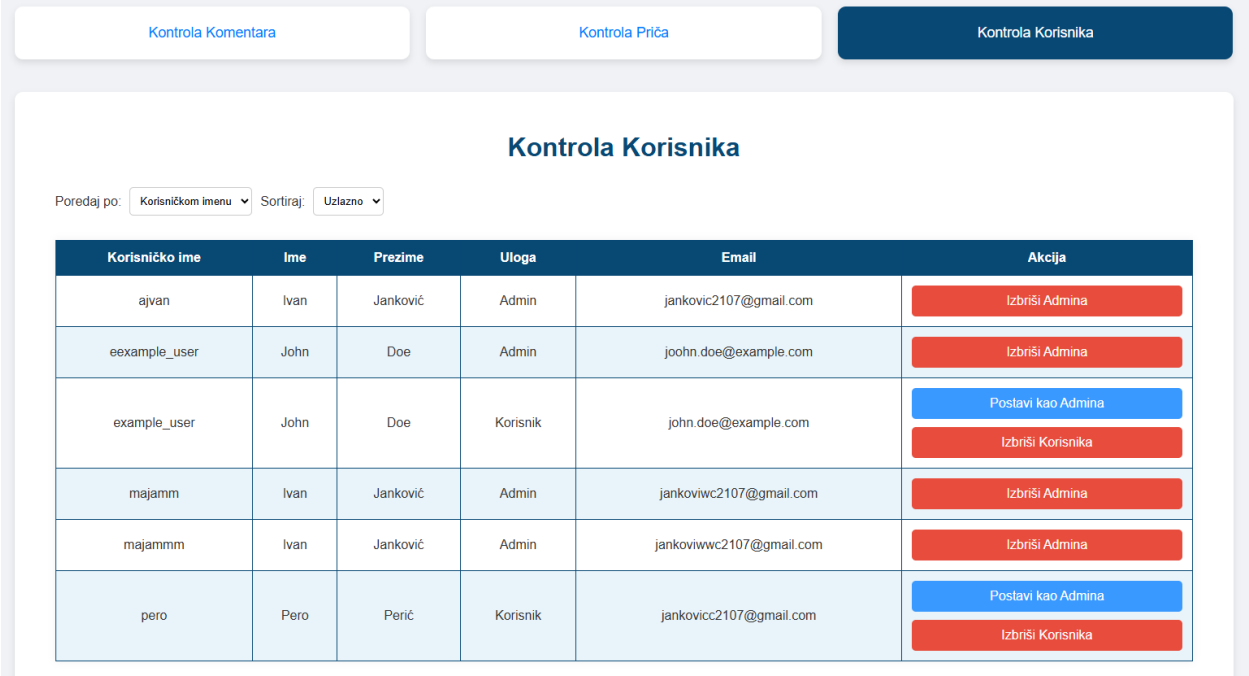
Slika 5.6 *Kontrola komentara na administratorskoj stranici*

Dodatno je postavljena mogućnost da se filtriraju komentari te da se prikazuju samo oni koji su prijavljeni (slika 5.7). Pritiskom na gumb izbriši komentar, komentar se briše iz baze te se više ne prikazuje u aplikaciji.



Slika 5.7 *Kontrola komentara na administratorskoj stranici sa filtriranim prijavljenim komentarima*

Odabirom opcije za *Kontrola Korisnika* u tablici se prikazuju svi korisnici koji su prijavljeni u aplikaciju (slika 5.8). Omogućeno je filtriranje korisnika, odnosno sortiranje po korisničkom imenu, imenu i prezimenu. Postavljen je gumb za brisanje te također je postavljen i gumb za postavljanje korisnika kao administratora.



Slika 5.8 *Kontrola korisnika na administratorskoj stranici*

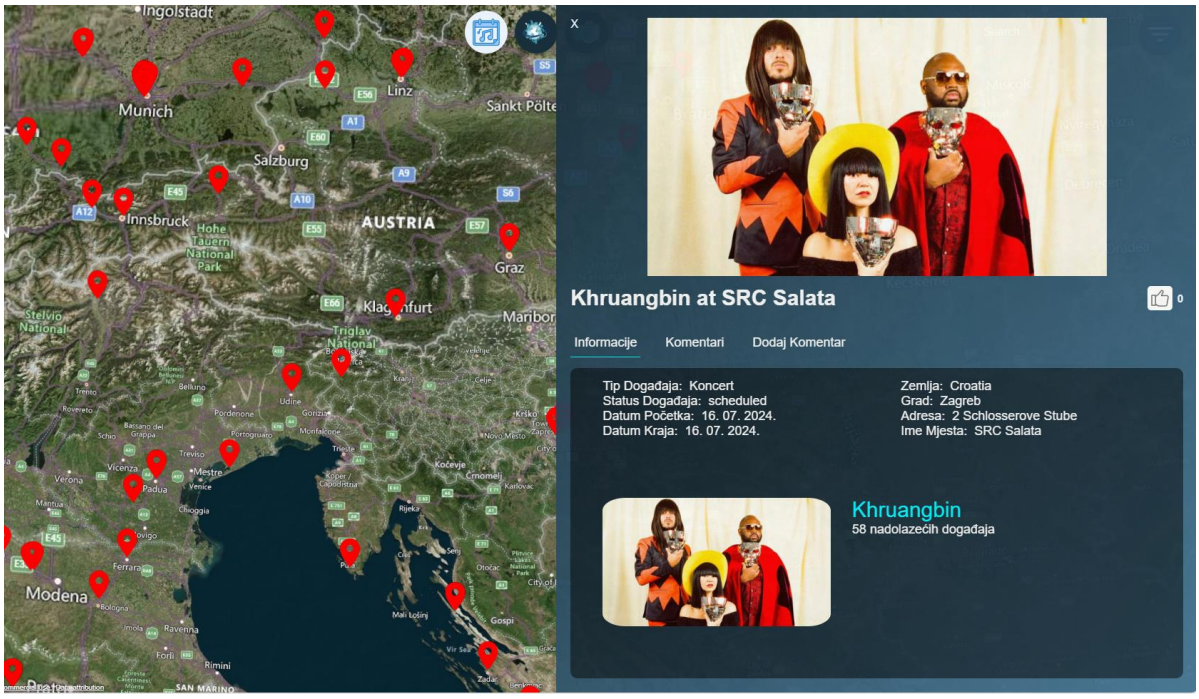
Glavna stranica aplikacije je globus Zemlje na kojem se nalaze oznake za događaje ili turističke lokacije ili priče, ovisno o korisnikovom odabiru. Trenutno su na slici 5.9 prikazane oznake za događaje. Korisnik ima mogućnost približavanja i udaljavanja globusa. Također, može okretati globus tako da pritisne na njega mišem i okreće ga lijevo, desno, gore, dolje.

Slika 5.9 *Glavna stranica aplikacije globus Zemlje*

Prelaskom preko neke od oznaka otvara se skočni prozor koji prikazuje generalne podatke o događaju (slika 5.10).

Slika 5.10 *Skočni prozor prelaskom miša preko oznake*

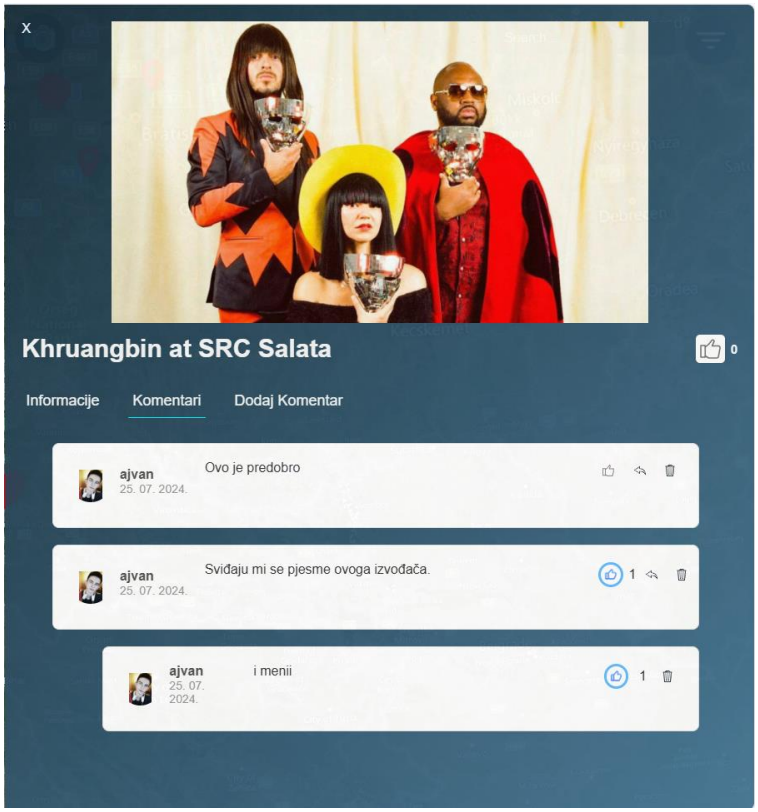
Klikom na neku od oznaka otvara se prozor s desne strane ekrana u kojem se nalaze sve informacije o događaju. Korisnik može kliknuti na gumb pokraj naslova te tom akcijom će postaviti da mu se sviđa ovaj događaj. Tom akcijom će se broj oznaka *sviđa mi se* s 0 povećati na 1. Korisnik može samo jednom pritisnuti taj gumb. Ako ga ponovno pritisne maknut će se jedna oznaka *sviđa mi se* s događaja.



Slika 5.11 *Informacije o događaju nakon klika miša na oznaku*

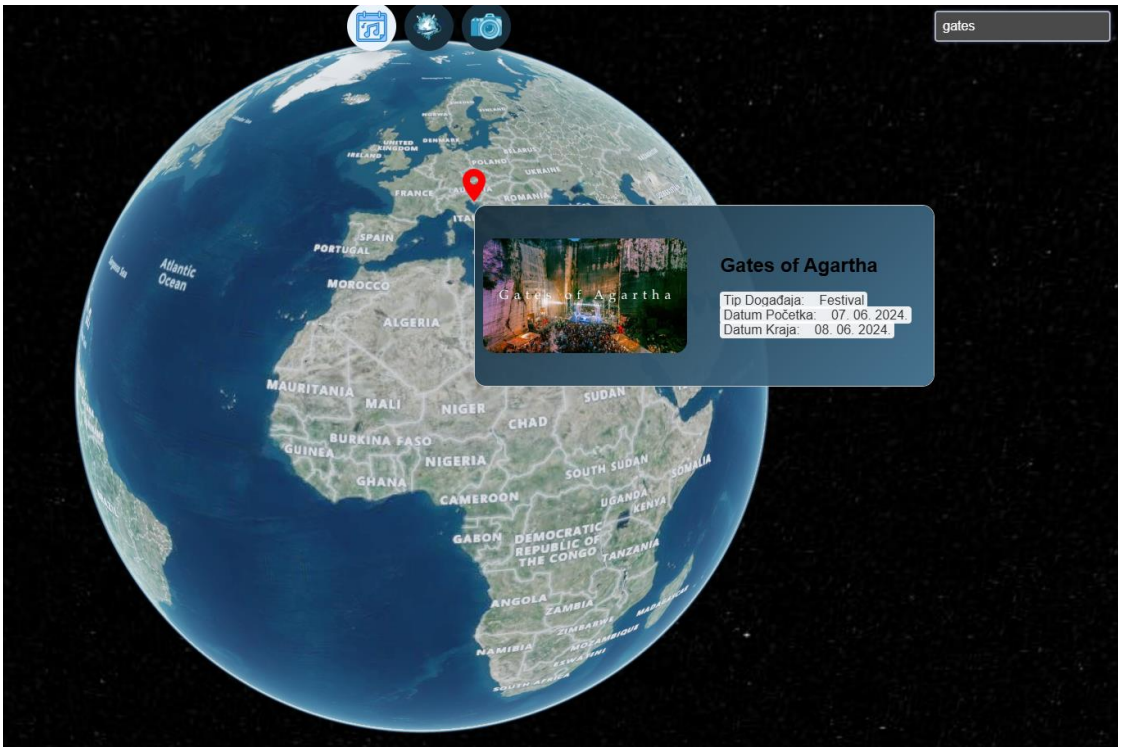
Ispod naslova događaja postavljena je navigacijska traka koja nudi mogućnosti da se pregledaju sve informacije događaja, svi komentari događaja te dodavanja komentara za taj određeni događaj.

Pritiskom na komentare u navigacijskoj traci prikazuju se svi komentari vezani za taj događaj (slika 5.12). U gornjem desnom kutu komentara postavljene su tri ikone. Prva ikona se koristi za postavljanje *sviđa mi se* na komentar, te isto funkcionira kao prethodno opisani gumb za događaj. Korisnik može dodijeliti samo jedan *sviđa mi se* komentaru, sljedeći puta kada se klikne na tu ikonu uklanja se. Druga ikona se koristi za odgovaranje na komentar. Treća ikona može biti ikona za brisanje ili ikona za prijavu komentara ovisno jesi li ti taj koji je postavio komentar na taj određeni događaj ili ne.



Slika 5.12 *Komentari događaja*

Filtriranje događaja se može napraviti na dva načina. Prvi način pruža korisniku mogućnost da unese u tražilicu neku riječ koja je povezana s nazivom događaja i prikazat će se na globusu samo filtrirane oznake kao što je prikazano na slici 5.13.

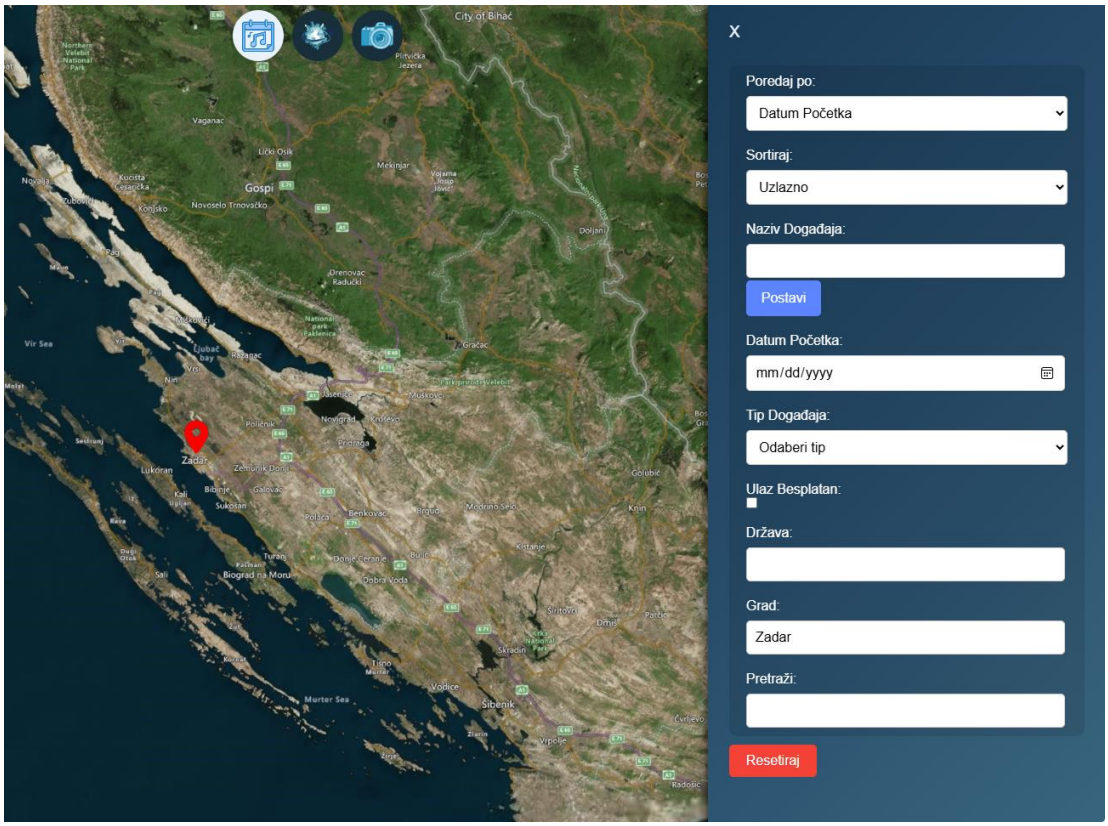


Slika 5.13 *Filtrirani događaji prikazani na globusu*

Druga mogućnost je da korisnik pritisne na gumb, koji je prikazan na slici 5.14, za filtriranje i s desne strane će mu se otvoriti prozor sa svim mogućnostima filtriranja.

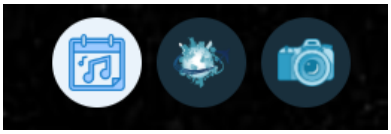
Slika 5.14 *Gumb za prikaz filtera*

Pritiskom na gumb otvara se prozor koji korisniku daje opciju da filtrira podatke po nazivu, po državi, po gradu, po početnom datumu i slično. Oznake na globusu se filtriraju dinamički, odnosno odmah prilikom unosa podatka u neko polje filtera, lista događaja se filtrira. Na slici 5.15 prikazan je primjer filtriranja događaja po gradu. U polje za grad unesen je podatak Zadar te se vraća oznaka koja prikazuje događaj koji se nalazi u Zadru.



Slika 5.15 *Filtrirani događaji po gradu prikazani na globusu*

Slika 5.16 prikazuje tri gumba te oni predstavljaju Događaje, Turističke Lokacije i Priče. Ovisno o tome koji je gumb odabran, taj sadržaj će se prikazati na globusu. Pozadina odabranog gumba će biti drugačija od drugih, odnosno njegova pozadina će biti svijetlo plave boje, dok će ostalima boja biti tamno plave boje. Kada se prijeđe mišem preko nekog gumba, on se proširi te prikazuje što zapravo taj gumb predstavlja.



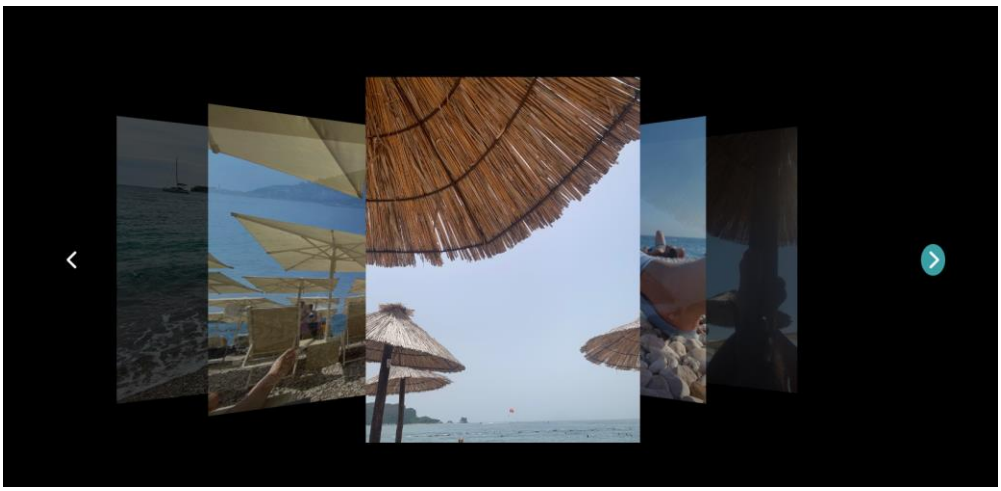
Slika 5.16 *Akcijski gumbovi za promjenu sadržaja na karti*

Turističke lokacije i priče imaju iste funkcionalnosti kao što imaju i događaji. Razlikuju se u dvije stvari. Prvo, svi korisnici imaju mogućnost dodavanja slika u turističkim lokacijama, a u pričama samo oni korisnici koji su postavili priču mogu nadodati još slika. Druga stvar je u količini slika. Događaji prikazuju samo jednu sliku, dok priče i turističke lokacije mogu imati više slika. Ako

postoji više slika, na karti se, nakon odabira neke oznake, prikazuje samo zadnjih pet postavljenih (slika 5.17). Ostale slike korisnik može pregledati u galeriji do koje može doći odabirom *Prikaži Galeriju* opcije. Slike se prikazuju s pomoću klizača koji je postavljen automatski da izmjenjuje slike svake tri sekunde. Korisnik može pritiskom na kružiće ispod odabrati koju sliku želi pogledati.

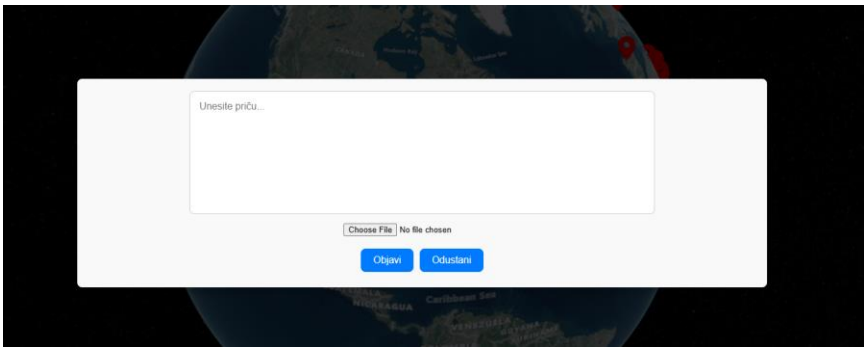
Slika 5.17 *Klizač slika*

Ako korisnik odabere opciju Prikaži Galeriju u njoj su prikazane sve slike te turističke lokacije ili priče. Slike se izmjenjuju korištenjem strelica za lijevo i desno (slika 5.18).



Slika 5.18 *Galerija slika*

Priča se u aplikaciju dodaje kada se dva puta klikne na neko mjesto na karti. Zatim se otvara obrazac, prikazan na slici 5.19. U obrazac se postavlja tekst priče koji je obavezan te slika koja je opcionalna.



Slika 5.19 *Obrazac za postavljanje priče*

Stranica korisnikovog profila sadržava podatke njegove aktivnosti na aplikaciji i njegove službene podatke poput korisničkog imena, email adrese i slično. Prikazane su njegove zadnje dvije priče koje je objavio te događaji i turističke lokacije koje je zadnje označio sa *sviđa mi se*.

Stranice koje prikazuju turističke lokacije i događaje su sastavljene na isti način. Na primjer, stranica koja prikazuje događaje, prikazuje tri dijela. U prvom dijelu su prikazani svi događaji te se taj dio može filtrirati isto kao što je prikazano filtriranje na karti. Drugi dio prikazuje događaje koje je korisnik označio sa *sviđa mi se*. Zadnji dio te stranice prikazuje događaje koji su najpopularniji odnosno koji imaju najviše oznaka *sviđa mi se*.

Sva tri dijela su sastavljena od kartica koje su već spomenute na prvoj stranici aplikacije (slika 5.20). Kartice sadržavaju generalne podatke o događaju. Postavljena je mogućnost da se klikne na tu karticu te tom akcijom se korisnika vodi na kartu na kojoj je prikazana oznaka tog događaja. Zatim se otvara, automatski prozor sa svim informacijama tog događaja.

Slika 5.20 *Kartica jednog od događaja*

6. ZAKLJUČAK

Ovim diplomskim radom predstavljen je razvoj i implementacija kartografske aplikacije, koja spaja geografske podatke s turističkim mjestima, glazbenim događajima i pričama različitih korisnika. U radu je opisan tijek razvoja aplikacije koji kreće od *PostgreSQL* baze podataka. Zatim je prikazana višeslojna arhitektura poslužiteljskog dijela, sastavljena u *ASP.NET* okruženju koristeći *C#* programski jezik. Na kraju je opisana klijentska strana aplikacije, koja je napravljena u *Reactu*. Geo označena pripovjedačka karta prikazuje kako objedinjavanjem informativnih i interaktivnih elemenata za korisnike, iskustvo putovanja postaje puno jednostavnije, bogatije te korisnici ostvaruju dublju vezu s destinacijama.

Turizam je jedna od vodećih grana industrije te iskorištavanje moderne tehnologije, poput kartografskih aplikacija, pruža veliku prednost u daljnjem razvoju. Osim što aplikacija pruža jedinstven način istraživanja svijeta, ona također stvara i zajednicu ljudi gdje svi mogu sudjelovati u dijeljenju vlastitih iskustava i preporuka drugim korisnicima. Sjedinjuju se podaci sa vlastitim iskustvima korisnika što uvelike poboljšava planiranje i organiziranje putovanja.

Ovu aplikaciju je moguće nadograditi uvođenjem nekih novih inovativnih ideja i funkcionalnosti. Na primjer, uvođenjem kompleksnih algoritama umjetne inteligencije za preporuku turističkih lokacija i glazbenih događaja na osnovu korisnikovog ponašanja na aplikaciji. Dodatno unaprjeđenje moglo bi uključivati ostvarivanje direktne komunikacije među korisnicima, što bi učinilo zajednicu jačom i bližom.

LITERATURA

[1] Google Maps, https://www.google.com/maps/ [pristupljeno 11.6.2024.]

[2] Sygic Maps, https://maps.sygic.com/ [pristupljeno 11.6.2024]

[3] SongKick, https://www.songkick.com/ [pristupljeno 11.6.2024.]

[4] PostgreSql, https://www.postgresql.org/ [pristupljeno 11.6.2024.]

[5] Microsoft, "What is .NET Framework?", .NET Learn, https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework, [pristupljeno 11.6.2024.]

[6] GeeksforGeeks, "Introduction to ASP.NET", https://www.geeksforgeeks.org/introduction-to-asp-net/, [pristupljeno 11.6.2024.]

[7] Gillis, A. S., Lewis, S., "Object-oriented programming (OOP)", TechTarget - SearchAppArchitecture, https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP, [pristupljeno 12.6.2024.]

[8] Dubey, M. K., Sharma, N. (urednici), *Object-Oriented Programming*, Lovely Professional University, ISBN: 978-93-87034-82-2, Excel Books Private Limited, Delhi, www.lpude.in.

[9] Naik, P. G., Oza, K. S., *Awesome React.js (Unleash the Power of Modern UI Building)*, Vol. 1, International Institute of Organized Research (I2OR), ISBN: 978-81-961331-4-6, Green ThinkerZ, Chandigarh, 2023.

[10] Packt Editorial Staff, "What is a multi layered software architecture?", Packt Hub, 17. svibnja 2018., https://hub.packtpub.com/what-is-multi-layered-software-architecture/, [pristupljeno 19. srpnja 2024.]

[11] CesiumJS, "CesiumJS Documentation", Esri, https://cesium.com/platform/cesiumjs/, [pristupljeno 23.7.2024.]

SAŽETAK

U ovom radu izrađena je kartografska aplikacija koja pruža korisnicima jednostavan i interaktivan način pronalaženja turističkih lokacija, budućih glazbenih događaja i razgledavanja priča različitih korisnika. Glavni dio aplikacije implementira *CesiumJS* za prikaz trodimenzionalnog globusa Zemlje na kojem se prikazuju oznake postavljene prema geografskoj dužini i širini. Svaka oznaka predstavlja jedan događaj ili turističku lokaciju ili korisnikovu priču ovisno o podacima koji se prikazuju na globusu. Pritiskom na jednu od tih oznaka prikazuju se sve detaljne informacije, komentari i slike. Aplikacija pruža platformu na kojoj korisnici mogu dijeliti osobna iskustva i razgledavati iskustva drugih korisnika te samim time pomaže u donošenju boljih i informiranih odluka prilikom planiranja i organiziranja nekog putovanja. Razvoj aplikacije kreče od razvoja relacijske, *PostgreSQL* baze podataka. Zatim slijedi implementacija višeslojne arhitekture poslužiteljskog sloja koristeći *ASP.NET* okruženje. Unutar poslužiteljskog sloja obrađuju se podaci, uspostavlja se autorizacija korisnika, kontrolira se razina pristupa određenim podacima i funkcionalnostima. Na kraju koristeći *React*, uspostavlja se klijentsko sučelje.

Ključne riječi: aplikacija, ASP.NET, CesiumJS, PostgreSQL, React

ABSTRACT

Title: *Geo tagged narrative map*

In this paper, a cartographic application was created that provides users with a simple and interactive way to find tourist locations, future music events and browse through stories of different users. The main part of the application implements CesiumJS to display a three-dimensional globe of the Earth on which tags are placed according to longitude and latitude. Each tag represents one event or tourist location or user's story depending on the data displayed on the globe. Clicking on one of these tags displays all the detailed information, comments and images. The application provides a platform where users can share personal experiences and view the experiences of other users, thus helping to make better and informed decisions when planning and organizing a trip. The development of the application starts with the development of a relational, PostgreSQL database. This is followed by the implementation of a multi-layered server architecture using ASP.NET environment. Within the server layer, data is processed, user authorization is established, the level of access to certain data and functionalities is controlled. Finally, using React, a client interface is established.

Keywords: application, ASP.NET, CesiumJS, PostgreSQL, React

ŽIVOTOPIS

Ivan Janković, rođen 21. srpnja 1999. godine u Slavonskom Brodu. Osnovnu školu pohađao je u Slavonskom Šamcu nakon koje upisuje gimnaziju „Matija Mesić“ Slavonski Brod. Tijekom osnovnoškolskog i srednjoškolskog obrazovanja aktivno se bavio nogometom te sudjelovao u mnogobrojnim natjecanjima. Nakon srednje škole 2018. godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Nakon završenog preddiplomskog studija stječe titulu prvostupnika inženjera računarstva te 2022. godine upisuje diplomski studij Računarstva, izborni blok Informacijske i podatkovne znanosti.

_____

Potpis Autora

PRILOZI

PRILOG 1 – programski kod *Geo Označene Pripovjedačke Karte* na github repozitoriju: https://github.com/Ivan2199/DIPLOMSKI_RAD_Geo_Oznacena_Pripovjedacka_Karta.git