Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Vasja Škarabot

# Spletna aplikacija za glasbene festivale

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Aljaž Zrnec

Ljubljana, 2024



*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

**Kandidat:** Vasja Škarabot
**Naslov:** Spletna aplikacija za glasbene festivale
**Vrsta naloge:** Diplomska naloga na visokošolskem programu prve stopnje Računalništvo in informatika
**Mentor:** viš. pred. dr. Aljaž Zrnec

**Opis:**
Besedilo teme diplomskega dela študent prepiše iz študijskega informacijskega sistema, kamor ga je vnesel mentor. V nekaj stavkih bo opisal, kaj pričakuje od kandidatovega diplomskega dela. Kaj so cilji, kakšne metode naj uporabi, morda bo zapisal tudi ključno literaturo.

**Title:** Web application for music festivals

**Description:**
opis diplome v angleščini

*Na tem mestu zapišite, komu se zahvaljujete za pomoč pri izdelavi diplomske naloge oziroma pri vašem študiju nasploh. Pazite, da ne boste koga pozabili. Utegnil vam bo zameriti. Temu se da izogniti tako, da celotno zahvalo izpustite.*

# Kazalo

# Seznam uporabljenih kratic

| kratica | angleško | slovensko |
|---|---|---|
| **API** | application programming interface | aplikacijski programski vmesnik |
| **CRUD** | create, read, update and delete | ustvarjanje, branje, posodabljanje in brisanje |
| **DBMS** | database management system | sistem za upravljanje podatkovnih baz |
| **ORM** | object-relational mapping | objektno-relacijsko preslikovanje |
| **SSR** | server-side rendering | upodabljanje na strani strežnika |
| **SVM** | support vector machine | metoda podpornih vektorjev |
| **MVC** | model-view-controller | model-pogled-kontroler |
| **MVT** | model-view-template | model-pogled-predloga |

# Povzetek

**Naslov:** Spletna aplikacija za glasbene festivale

**Avtor:** Vasja Škarabot

Diplomska naloga predstavlja razvoj in implementacijo spletne aplikacije za glasbene festivale. Glavni namen aplikacije je, uporabnikom olajšati dostop do vseh pomembnih informacij, ter jim omogočiti medsebojno komunikacijo in preko le te graditi skupnosti s pomočjo forumov in klepetov.

Spletna aplikacija uporabniku omogoča pregled nad festivali, vključno z iskanjem prenočišč v bližini ter navodili za pot do prizorišča, forum na katerem lahko uporabniki objavljajo novice, delijo nasvete ter postavljajo morebitna vprašanja, ki se jim porajajo, ter klepeti, katere lahko uporabniki uporabijo z namenom razpravljanja in dogovarjanja za skupni obisk.

Za implementacijo čelnega dela smo uporabili ogrodje Nuxt.js, za zaledni del pa Django z razširitvijo REST Framework in podatkovno bazo PostgreSQL. Vse skupaj poganjamo v Dockerju. Zemljevidi so implementirani s pomočjo platforme Mapbox, realnočasovno komunikacijo v klepetih omogoča Pusher, slike pa shranjujemo s pomočjo Firebase shrambe.

**Ključne besede:** glasba, festival, socialno omrežje, spletna aplikacija.

# Abstract

**Title:** Web application for music festivals

**Author:** Vasja Škarabot

The thesis presents the development and implementation of a web application for music festivals. The main purpose of the application is to make it easier for users access to all relevant information, and to allow users to communicate with each other and build communities through forums and chats.

The web application provides the user with an overview of the festivals, including a search for nearby accommodation and directions to the venue, a forum where users can post news, share tips and ask any questions they may have, and chat rooms which users can use to discuss and arrange to visit together.

For the front-end implementation, we used the Nuxt.js framework, and for the back-end we used Django with the REST Framework extension and the PostgreSQL database. We run everything inside Docker containers. Maps are implemented using the Mapbox platform, real-time communication in chats is enabled by Pusher, and images are stored using the Firebase Storage.

**Keywords:** music, festival, social network, web application.

# Poglavje 1

# Uvod

## 1.1 Motivacija

Veliko ljudi vsako leto obišče kakšen glasbeni festival. Pridobivanje informacij za načrtovanje samega obiska festivala pa lahko kar hitro postane nadležno, in lahko hitro pokvari izkušnjo še pred samim festivalom. Običajno uporabnik najprej izbere glasbeni festival, ki se ga bo udeležil. Ponavadi to stori na podlagi različnih dejavnikov, kot so lokacija, datumi in žanri, pri končni izbiri pa veliko pripomorejo tudi nasveti ostalih ljudi, predvsem tistih ki so v preteklosti že obiskali izbrani festival. Problem pri tem pa je, da je potrebno vse izmed naštetih informacij iskati po različnih spletnih straneh saj rešitve, ki bi ponujala vse to na enem mestu ni. Zato bi bilo smiselno implementirati rešitev, ki bo uporabnikom omogočila vpogled v vse na enem mestu, s tem da si uporabniki pomagajo med sabo.

## 1.2 Cilji

Cilj diplomske naloge je torej razviti sodobno spletno aplikacijo, ki bo uporabnikom olajšala iskanje glasbenih festivalov in udeležbo na njih. Želimo tudi skrajšati proces planiranja tako, da uporabniku omogočimo iskanje poti do glasbenega festivala, ter bližnjih prenočišč. Za vsa vprašanja, ki se uporab-

niku pojavljajo bodo na voljo forumi, na katerih bodo uporabniki ustvarjali skupnosti za glasbene festivale. Želimo pa tudi vzpodbuditi interakcijo med obiskovalci festivalov, ki morda potujejo sami ali pa tistih, ki so že na prizorišču in s tem izboljšati festivalsko izkušnjo, za kar bomo razvili sistem klepetov.

## 1.3 Struktura diplomske naloge

Diplomsko nalogo sestavlja 7 poglavij. Najprej bomo preverili obstoječe rešitve, ter njihove pomankljivosti, pa tudi morebitne prednosti. Za tem bo sledil zajem zahtev, kjer bomo opisali funkcionalnosti sistema, ter predstavili diagram primerov uporabe. Sledilo bo poglavje v katerem bomo predstavili uporabljene tehnologije in orodja, ter razlogi zakaj smo se odločili za uporabo teh. V petem poglavju bomo predstavili razvoj rešitve, v katerem bomo vključili samo arhitekturo aplikacije, potem načrtovanje podatkovne baze, ter implementacija čelnega in zalednega dela. V naslednjem poglavju bomo nato predstavili še našo končno rešitev s pomočjo slik uporabniškega vmesnika, ter opisom delovanja vseh pomembnejših funkcionalnosti. V zaključku, pa bomo povzeli našo nalogo, ter podali nekaj mogočih izboljšav v nadaljnjem razvoju.

# Poglavje 2

# Obstoječe rešitve

Na trgu trenutno ni nobene podobne rešitve, zaradi česar se obiskovalci festivalov poslužujejo različnih aplikacij in spletnih strani. Za postavljanje vprašanj sta to običajno Reddit ali pa Tripadvisor. Ti rešitvi nista primarno namenjeni glasbenim festivalom, zato težko najdemo koristne podatke, saj vsebujeta le forume za večje festivale. Poleg tega mora vsak uporabljati več aplikacij za različne namene in zato nimamo vsega na eni platformi. Uporabnik mora tako npr. za iskanje informacij o festivalu najprej na spletno stran festivala, nato prebira različne forume (Reddit, Tripadvisor), zraven tega še išče prenočišča na Bookingu, potem pa se še na tretji platformi z ostalimi pogovarja/dogovarja (npr. Messenger). V tem poglavju bodo opisane naštete alternative, njihove prednosti in slabosti.

## 2.1 Tripadvisor

Tripadvisor je spletna platforma za iskanje prenočišč [17]. Je ena večjih platform namenjenih planiranju potovanj, ki uporabnikom omogoča tudi podajanje kritik in ocen.

- **Prednosti**:
  - Velik nabor uporabnikov in posledično kritik in ocen za prenočišča in razne dogodke.

  - Širok nabor funkcionalnosti. Poleg osnovnih funkcionalnosti še orodja za načrtovanje poti, vodniki in priporočila.

- **Slabosti**:
  - Vsebuje le najpopularnejše festivale.
  - Ocene bolj osredotočene na nastanitve in potovanje kot na sam festival.
  - Ne vsebuje nekaterih pomembnih informacij, kot so datumi, informacije o cenah in izvajalcih.
  - Uporabnik lahko z ostalimi uporabniki komunicira le posredno preko foruma. Za neposredno komunikacijo se mora poslužiti drugih platform.

## 2.2 Reddit

Reddit je forumsko socialno omrežje, kjer lahko uporabniki objavljajo, ostali pa nato te objave komentirajo in glasujejo [11]. Objave so razvrščene v podstrani imenovane subredditi oz. skupnosti. Na Redditu najdemo skupnosti za nekatere glasbene festivale, kjer se uporabniki obveščajo o novicah o festivalu in podajajo različna vprašanja. Je pa teh skupnosti precej malo, še tiste ki so, pa so po večini skupnosti največjih glasbenih festivalov na svetu (Coachella, Tomorrowland, UMF).

- **Prednosti**:
  - Pri večjih festivalih ogromen nabor vprašanj, razprav, nasvetov in novic.
  - Možnost realnočasovne komunikacije z ostalimi.

- **Slabosti**:
  - Večino osnovnih informacij mora uporabnik pridobiti na ostalih straneh.

  - Težko je najti skupnosti, ki niso namenjene le največjim festivalom.
  - Čeprav lahko uporabniki komunicirajo v realnem času, lahko to počnejo le v privatnih skupinah, ki niso del subredditov.

## 2.3 Woov

Woov je namenska aplikacija za obiskovalce glasbenih festivalov [28]. Povezani so z več kot 1000 dogodki po 40 državah, uporabnikom pa omogočajo vpogled v časovnice, zemljevid prizorišč in komunikacijo z ostalimi uporabniki. Namen aplikacije je pomagati obiskovalcem, ki so že na prizorišču.

- **Prednosti**:
  - Uporabniki lahko aplikacijo prenesejo na telefon.
  - Pomembne informacije kot so datumi, prizorišče, nastopajoči izvajalci.
  - Uporabniki si lahko pomagajo z zemljevidom, ki vključuje lokacije stojnic, wc-jev, odrov in polnilnih postaj.
  - Realnočasovna komunikacija.
- **Slabosti**:
  - Dogodki so običajno dodani nekaj dni pred začetkom festivala. Posledično si uporabnik za planiranje ne more dosti pomagati.
  - Aplikacija včasih deluje nelogična za nove uporabnike, saj so nekatere funkcionalnosti precej skrite.
  - Uporabniki ne morejo podati svojih mnenj.
  - Aplikaciji lahko prispevajo le partnerji. To ni nujno slabo, vendar omeji aplikacije le na določene festivale.

# Poglavje 3

# Zajem zahtev

Preden smo sploh začeli z načrtovanjem podatkovne baze in kodiranjem, je bilo potrebno definirati vse naloge, ki jih bo sistem opravljal. S tem smo si zadali načrt in ogrodje, kateremu smo pri razvoju nato sledili. Z dobro definicijo zahtev smo dosegli to, da smo zmanjšali verjetnost pomembnih sprememb med razvojem, s čimer smo prihranili čas in trud.

## 3.1 Funkcionalnosti sistema

Naša aplikacija mora uporabniku omogočiti:

- Registracijo in prijavo.
- Pregled glasbenih festivalov z možnostjo iskanja in filtriranja.
- Dodajanje novih festivalov in dodeljevanje moderatorskih pravic.
- Pregled podrobnosti glasbenih festivalov, vključno z datumi, žanri in povezavo do uradne spletne strani.
- Zemljevid z lokacijo glasbenega festivala, ki se lahko razširi v navodila za pot.
- Iskanje prenočišč v okolici glasbenega festivala glede na podane parametre.

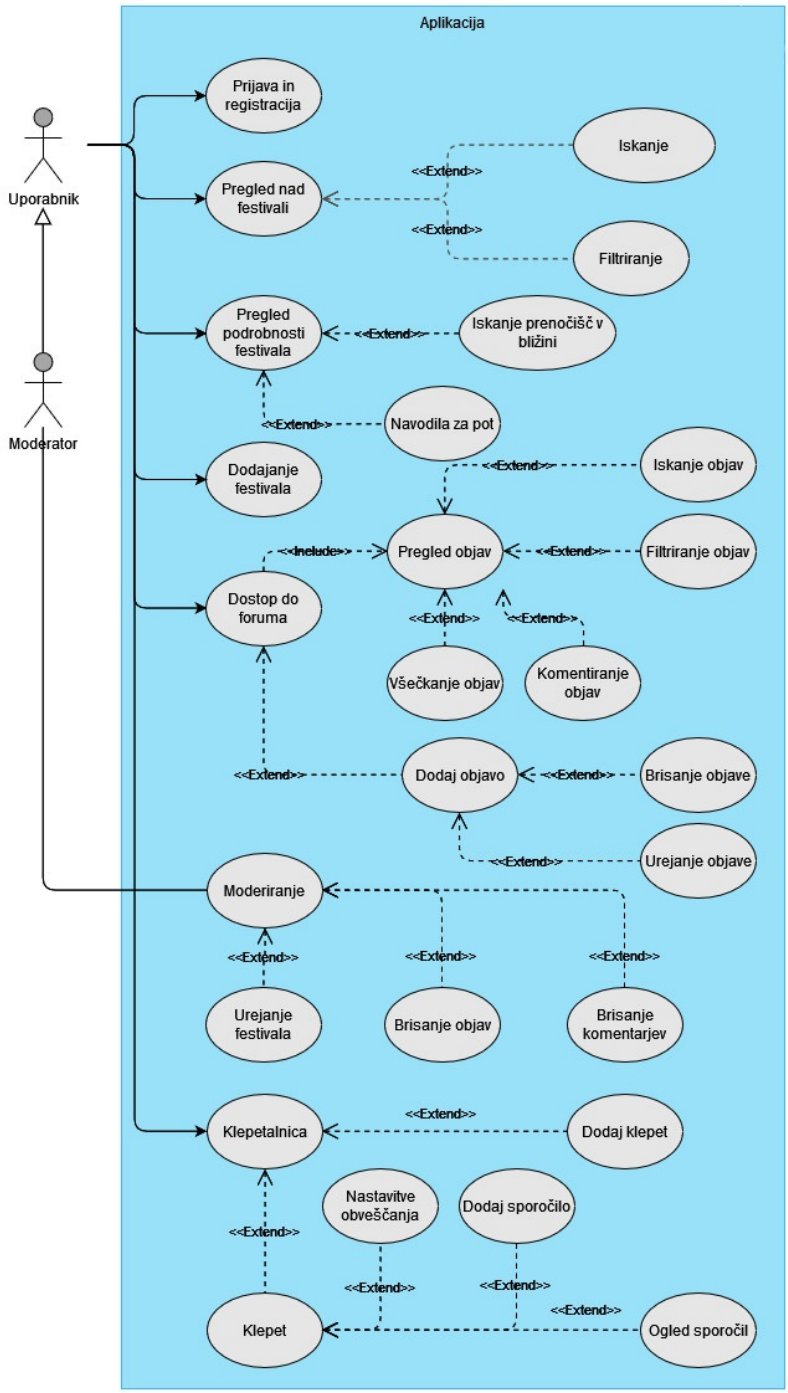
- Branje objav in možnost filtriranja in iskanja na forumu za določen festival.
- Objavljanje na forumih.
- Všečkanje in komentiranje objav.
- Dodajanje javnih klepetov za festivale.
- Pošiljanje sporočil v klepetih.
- Naročanje na obvestila za nova sporočila.

Moderatorjem pa povrh vsega še:

- Urejajanje podatkov za svoje festivale.
- Brisanje objav in komentarjev z namenom odstranjevanja neprimerne vsebine.

## 3.2 Diagram primerov uporabe

Na Sliki **??** je prikazan diagram primerov uporabe za našo aplikacijo. V diagramu nastopata 2 akterja in sicer Uporabnik in Moderator. Iz diagrama je razvidno, kaj lahko oba tipa uporabnikov počneta v aplikaciji. Podrobnejši opis vseh funkcionalnosti pa bomo predstavili v poglavju **??**.




Slika 3.1: Diagram primerov uporabe.

# Poglavje 4

# Uporabljene tehnologije in orodja

Odločili smo se za uporabo modernih, dobro poznanih in pogosto uporabljenih tehnologij in orodij. Tako smo za čelni del uporabili Nuxt [13], ki temelji na ogrodju Vue.js [24]. Zaledni del smo implementirali z uporabo Djanga [7] s knjižnico Django REST Framework [10], ki nam je močno olajšala razvoj API vmesnika in upravljanje s podatkovno bazo. Za podatkovno bazo smo uporabili PostgreSQL [18]. Pri tem smo podatkovno bazo ter Django strežnik poganjali v Dockerju [8]. Realnočasovno komunikacijo smo omogočili s pomočjo Pusher Channels [19], zemljevide in navodila za pot pa smo osnovali na platformi Mapbox [15]. Vse te tehnologije in orodja, ter njihov namen in uporabo v naši rešitvi smo podrobneje predstavili v naslednjem poglavju.

Sicer, pa smo poleg omenjenih glavnih tehnologij še Vuetify.js za oblikovanje spletne strani [25], Git [9] in GitHub [1] za nadzor različic, Firebase Storage [5] za shranjevanje slik in Visual Studio Code [23] za urejanje kode. Poleg tega smo uporabili še nekaj Python knjižnic, med njimi BeautifulSoup4 [4] za strganje podatkov iz Bookinga [2], ter Djoser [12], ki ponuja vnaprej definirane končne točke za registracijo, prijavo, odjavo in aktivacijo uporabniškega računa.

## 4.1 Nuxt.js

Nuxt.js (na kratko Nuxt) je brezplačno odprtokodno orodje za razvoj čelnega dela spletnih aplikacij z Vue.js. V naši aplikaciji smo ga uporabili zato, ker omogoča hiter, enostaven, predvsem pa učinkovit razvoj aplikacij.

Poleg tega Nuxt nudi tudi možnosti SSR, ki omogoča hitrejše čase nalaganja strani in optimizacijo spletnih strani. Pri tem strežnik po začetni zahtevi odjemalca pošlje v celoti renderirano stran nazaj odjemalcu. JavaScript na strani odjemalca nato omogoči, da statična stran postane interaktivna Vue.js aplikacija (hidracija) [21].

Ena večjih prednosti uporabe je tudi enostavna konfiguracija poti na spletni strani, saj Nuxt samodejno razbere strukturo glede na vsebino mape *pages/*. V Vue.js npr. je za vsako novo dodano stran to potrebno storiti ročno.

## 4.2 Django in Django REST Framework

Django je Python ogrodje za razvoj spletnih aplikacij, ustvarjeno z namenom da uporabniku omogoči hiter razvoj, saj vsebuje širok nabor orodij, ki poskrbijo za pogosto uporabljena opravila spletnega razvoja (avtentikacija, administracija, upravljanje z podatkovno bazo...). Temelji na vzorcu MVT, ki je v osnovi zelo podoben bolj znanemu MVC, pri čemer [16]:

- Model (M) skrbi za podatke in vzdržuje povezavo s podatkovno bazo.
- Pogled (View - V) sprejema zahtevke in vrača odgovore.
- Predloga (Template - T) pa definira strukturo in postavitev strani. V naši aplikaciji smo predlogo modela MVT nadomestili z uporabo Nuxt.js.

Django REST Framework je razširitev za Django, posebej namenjena za gradnjo RESTful APIjev. V naši aplikaciji je bil uporabljen predvsem zaradi enostavnega dela s podatkovno bazo, enostavne serializacije podatkov in že vnaprej pripravljenih pogledov za nekatere osnovne CRUD operacije, ki jih z

lahkoto lahko priredimo za naše namene. Poleg tega omogoča tudi enostavno pisanje pravic za dostop do API končnih točk.

## 4.3 PostgreSQL

Za podatkovno bazo smo uporabili PostgreSQL. PostgreSQL je zmogljiv odprtokoden objektno-relacijski DBMS (sistem za upravljanje s podatkovnimi bazami). Kljub temu, da je odprtokoden se lahko kosa z ostalimi ponudniki, kot sta npr. Oracle in MySQL. Prav zaradi stroškov, se zelo pogosto uporablja v različnih startupih in za v raziskovalne namene [14].

## 4.4 Pusher Channels

Realnočasovno komunikacijo v klepetih smo zagotovili s pomočjo storitve Pusher Channels. Pusher Channels deluje na tehnologiji WebSockets, ki omogoča vztrajne TCP povezave med strežnikom in odjemalcem. S tem lahko strežnik in odjemalec dosežeta takojšnjo izmenjavo sporočil na zelo učinkovit način z zelo majhno zakasnitvijo, brez potrebe po tem, da bi odjemalec moral ponovno poslati zahtevo za prejem podatkov [26].

Pusher Channelsi delujejo na modelu objavi/naroči. To pomeni, da se aplikacije lahko naročajo na kanale v sistemu. Ko pride do sprememb, sistem objavi spremembo v ta isti kanal in vse aplikacije, ki so naročene na ta kanal, so potem obveščene [19].

## 4.5 Mapbox

Mapbox je spletna platforma, ki razvijalcem ponuja širok nabor orodij za delo z zemljevidi. Podatke črpa iz različnih virov, med drugim tudi iz MapStreetBoxa in NASE. V naši aplikaciji smo uporabili večino glavnih storitev, ki jih omogoča in sicer [20]:

- **Mapbox Maps**, ki omogoča prikaz zemljevidov z lokacijami.

- **Mapbox Directions API**, ki omogoča iskanje poti. Naši aplikaciji vrača vse podatke za pot od vnešene lokacije do festivala.
- **Geocoding API**, ki spreminja geografske koordinate v naslove in obratno. V naši aplikaciji je uporabljen, da koordinate zemljepisne širine in dolžine pretvori v človeku prijazno predstavitev.
- **Mapbox Search Box**, ki omogoča iskanje lokacij. V naši aplikaciji je uporabljen za iskanje začetnih lokacij pri navigaciji in za interaktivno iskanje po zemljevidu.

## 4.6 Docker

Docker je platforma, ki omogoča ustvarjanje, izvajanje in nameščanje aplikacij v kontejnerjih/vsebnikih. Temu procesu pravimo kontejnerizacija. Pri kontejnerizaciji zapakiramo programsko kodo s knjižnicami operacijskega sistema in odvisnostmi, ki so potrebne za izvajanje kode v lahko, izvedljivo datoteko, ki se lahko enako izvaja na kateri koli infrastrukturi [27]. Prednosti uporabe Dockerja je veliko, v našem primeru pa smo Docker uporabili predvsem zaradi izolacije okolja in lahke prenosljivosti. V Dockerju smo poganjali Django aplikacijo ter podatkovno bazo.

# Poglavje 5

# Razvoj rešitve

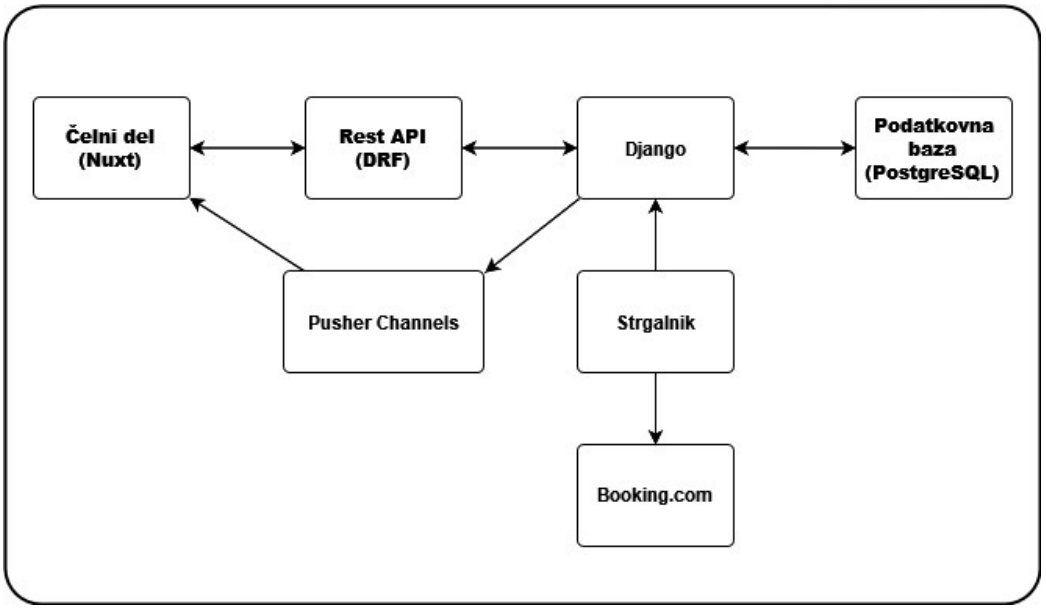
V tem poglavju bomo predstavili celoten proces razvoja naše rešitve, od začetnega načrtovanja arhitekture in podatkovne baze, do končne rešitve. Pri tem se bomo posvetili predvsem implementaciji pomembnejših funkcionalnosti sistema.

## 5.1 Arhitektura aplikacije

Spletna aplikacija je sestavljena iz čelnega dela implementiranega z Nuxt.js, ki pridobiva podatke preko APIja implementiranega v Django REST Frameworku. V zalednem delu imamo Django REST Framework, ki je povezan s Pusher Channels, da se ob vsakem dodanem sporočilu v klepetih takoj pošlje obvestilo na čelni del aplikacije in prikaže uporabniku. Django pa s svojo implementacijo ORM skrbi za upravljanje s podatkovnimi modeli in poizvedbami. Poleg tega Django pridobiva podatke o prenočiščih iz spletne strani Booking.com s pomočjo skripte za strganje podatkov. Na Sliki **??** je prikazana arhitektura naše rešitve.

## 5.2 Podatkovni model

Podatkovni model lahko za lažjo predstavo razdelimo na štiri dele:




Slika 5.1: Arhitektura razvite rešitve.

- **Tabela uporabnika**
  - **CustomUser** vsebuje uporabniško ime, e-pošto, geslo in ostale pomembne podatke uporabnika. Lahko vsebuje povezave na tabele *Comment*, *Message*, *Post*, ter vmesne tabele, ki služijo všečkanju objav in komentarjev.
- **Tabele za glasbene festivale**
  - **Festival** predstavlja glasbeni festival in med drugim vsebuje podatke o imenu, opisu, povezavi do spletne strani, lokaciji in datumih izvedbe. Festival ima lahko tudi določen klepet (*Chat*) in objave (*Post*), ki tvorijo forum.
  - **UserFestival** predstavlja vmesno tabelo med *CustomUser* in *Festival*. Uporabniku omogoča, da postane moderator glasbenega festivala.
  - **UserFavouriteFestival** tudi predstavlja vmesno tabelo med *Cu-*

*stomUser* in *Festival*. Uporabniku omogoča dodajanje glasbenih festivalov med priljubljene.

- **Tabele za forum**
  - **Post** vsebuje vse podatke objav, kot so naslov, vsebina in čas objave. Vsebuje tudi festival *Festival* in uporabnika *CustomUser*.
  - **PostLikedBy** predstavlja vmesno tabelo med uporabnikom in objavo. Tabela je namenjena temu, da lahko uporabnik všečka objave.
  - **PostDislikedBy** podobno kot *PostLikedBy* predstavlja vmesno tabelo med uporabnikom in objavo. Namenjena je temu, da lahko uporabnik objave označi z 'ni mi všeč'.
  - **Comment** predstavlja komentar objave in vsebuje podatke o vsebini, času in morebitnem izbrisu objave. Komentar ima tudi pripadajočo objavo, avtorja in všečke. Da smo lahko implementirali gnezdenje komentarjev smo uporabili polje *parent* - tuji ključ, ki se sklicuje na isto tabelo.
  - **CommentLikedBy** predstavlja vmesno tabelo med uporabnikom in komentarjem. Tabela je namenjena temu, da lahko uporabnik všečka komentarje.
  - **CommentDislikedBy** podobno kot *CommentLikedBy* predstavlja vmesno tabelo med uporabnikom in komentarjem. Namenjena je temu, da lahko uporabnik komentarje označi z 'ni mi všeč'.
- **Tabele za klepete**
  - **Chat** vsebuje podatek o imenu klepeta in festival, kateremu klepet pripada.
  - **Message** vsebuje čas in vsebino sporočila (tekst in slika), ter podatek o avtorju sporočila, pripadajočem klepetu in vsa obvestila povezana s sporočilom.

- **Notification** vsebuje podatke o času obvestila, statusu, ki pove ali je sporočilo bilo prebrano, ter podatke o prejemniku in klepetu v katerem je bilo sporočilo poslano, pa tudi sporočilo, na katerega se obvestilo navezuje.
- **NotifiedUsers** je vmesna tabela med uporabnikom in klepetom. Uporabnikom omogoča, da se lahko prijavijo in odjavijo na obveščanje za izbrani klepet.

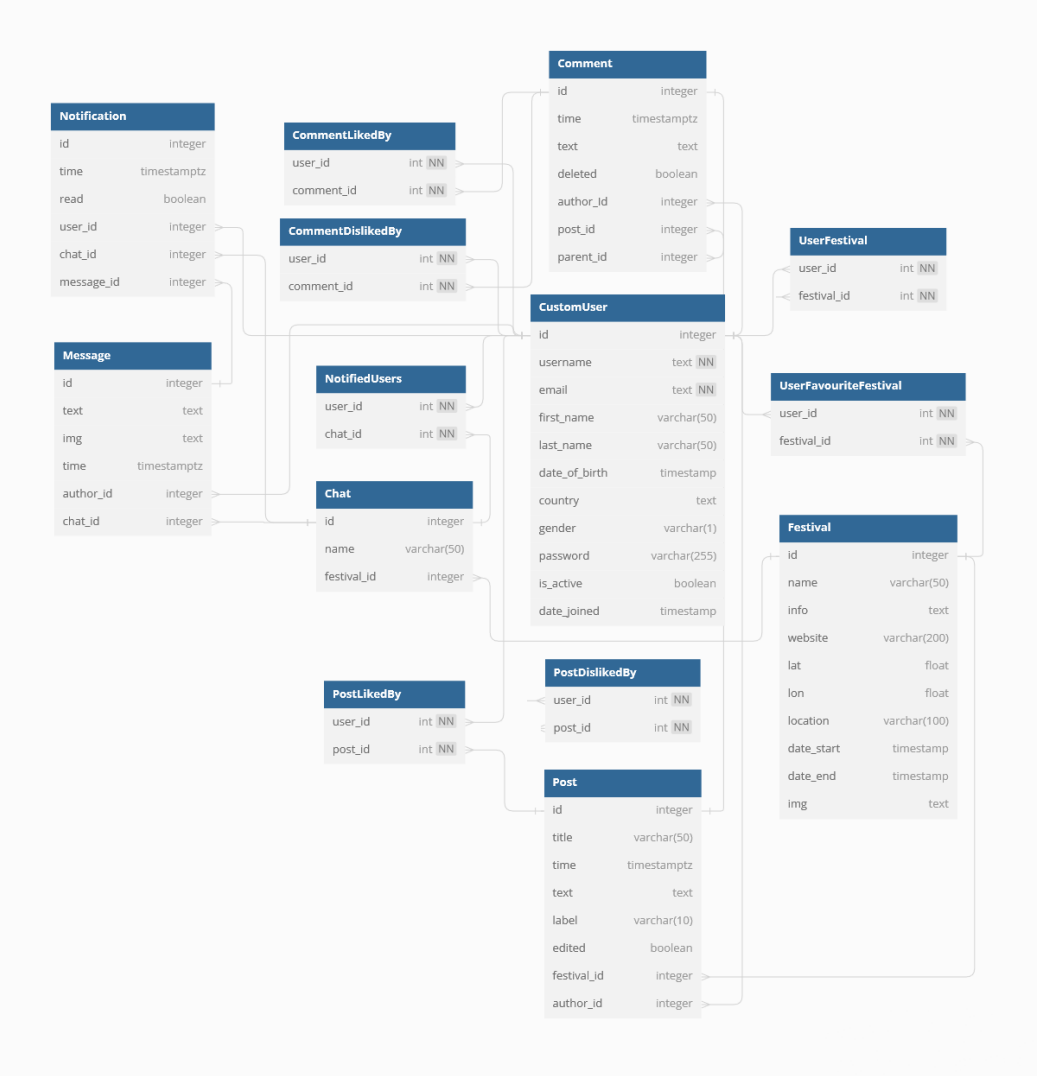
Na sliki **??** je prikazan celoten podatkovni model.

## 5.3 Zaledni del

Za implementacijo zalednega dela smo uporabili Django v kombinaciji z Django REST Frameworkom. Django nam je omogočil poenostavljeno delo s podatkovno bazo, saj nam z vgrajenim ORM sistemom, ni potrebno pisati SQL stavkov, temveč lahko kodiramo v Pythonu. To naredi vsako interakcijo s podatkovno bazo lažjo in bolj intuitivno, istočasno pa ne rabimo skrbeti glede SQL napadov, saj ORM vsako poizvedbo sestavi s poudarkom na varnosti. Uporaba Django REST Frameworka pa nam je z orodji za serializacijo, avtentikacijo in generičnimi pogledi, še dodatno olajšala implementacijo REST APIja.

### 5.3.1 Delo s podatkovno bazo

Uporaba Djanga nam je omogočila, da smo lahko enostavno ustvarili povezavo z zalednim delom aplikacije. Najprej smo v datoteki *models.py* definirali modele, ki v Djangu predstavljajo strukturo in relacije v podatkovni bazi s pomočjo objektov. Vsak Django model predstavlja eno tabelo, vsako polje modela določeno kot atribut razreda, pa en stolpec v tabeli podatkovne baze. Ko smo modele definirali, je Django poskrbel za kreacijo tabel glede na modele, vendar je bilo potrebno najprej pognati ukaz *'python manage.py makemigrations'*, da so se ustvarile migracije. Da smo spremembe v datoteki




Slika 5.2: Podatkovni model razvite rešitve.

z migracijami dejansko uporabili in posodobili podatkovno bazo, pa je bilo potrebno pognati ukaz *'python manage.py migrate'*.

Spodnja izvorna koda prikazuje, kako smo definirali model za *Festival*. Z opcijo *blank* definiramo, da v zahtevku s čelnega dela ni potrebno navesti določenega polja. Z *null* pa definiramo, da je vrednost polja v podatkovni bazi lahko NULL. Polje *genre* služi kot povezava tipa mnogo-proti-ena na tabelo *Genre* (en festival ima lahko več različnih žanrov). Pri tem smo z *models.SET_NULL* določili, da se ob primeru izbrisa instance žanra v festivalih povezanih s tem žanrom polje *genre* postavi na NULL. Privzeta nastavitev je *models.CASCADE*, ki bi v primeru, da je en žanr izbrisan, izbrisala vse festivale povezane s tem žanrom. V razredu *Meta* pa smo definirali, da so rezultati vsake poizvedbe privzeto urejeni po imenu festivala.

```
class Festival(models.Model):
    name = models.CharField(max_length=50)
    info = models.TextField(
        blank=True,
        default='No info provided'
    )
    website = models.URLField(
        blank=True,
        default='No website provided'
    )
    lat = models.FloatField(blank=True, null=True)
    lon = models.FloatField(blank=True, null=True)
    date_start = models.DateField(blank=True, null=True)
    date_end = models.DateField(blank=True, null=True)
    img = models.URLField(blank=True, null=True)

    genre = models.ForeignKey(
        Genre,
        on_delete=models.SET_NULL,
        null=True,
        blank=True
    )
```

```
    class Meta:
        ordering = ['name']
```

Izvorna koda 5.1: Primer kreiranja modela Festival v Djangu.

### 5.3.2 API

Za komunikacijo čelnega dela aplikacije z zalednim delom, smo implementirali RESTful API s pomočjo Django REST Frameworka, ki na zahteve odjemalca posluša na vratih 8000.

Uporaba Django REST Frameworka je močno pospešila izgradnjo APIja, saj smo lahko na enostaven način definirali, katera polja modela bodo serializirana (pretvorjena iz zapletenih podatkovnih struktur v obliko JSON za lažji prenos). Potrebno je bilo le definirati razred za vsak model, ter našteti katera polja bodo serializirana. Poleg osnovnih polj lahko serializiramo tudi izpeljana polja, ki niso neposredno na voljo v modelu. To lahko naredimo tako, da definiramo metodo za izračun vrednosti.

- **is_favourite** - Glede na uporabnika in izbrani festival, vrne vrednost *True* ali *False* odvisno od tega, ali ima uporabnik festival dodan med priljubljene.
- **get_is_mod** - Glede na uporabnika in izbrani festival, vrne vrednost *True* ali *False* odvisno od tega, ali je uporabnik moderator festivala.
- **get_num_favourites** - Glede na izbrani festival, vrne število vseh uporabnikov, ki imajo festival dodan med priljubljene.

```
class FestivalSerializer(serializers.ModelSerializer):
    is_favourite = serializers.SerializerMethodField()
    is_mod = serializers.SerializerMethodField()
    num_favourites = serializers.SerializerMethodField()

    class Meta:
        model = Festival
```

```
        fields = ['id',
                  'name',
                  'info',
                  'website',
                  'lat',
                  'lon',
                  'date_start',
                  'date_end',
                  'img',
                  'genre',

                  'is_mod',
                  'is_favourite',
                  'num_favourites']

        def get_is_favourite(self, obj):
            user = self.context['request'].user
            return user in obj.favourite_by.all()

        def get_is_mod(self, obj):
            user = self.context['request'].user
            return user in obj.mods.all()

        def get_num_favourites(self, obj):
            return len(obj.favourite_by.all())
```

Izvorna koda 5.2: Primer serializacije podatkov za model Festival.

Poleg serializacije podatkov nam Django REST Framework omogoča tudi uporabo tako imenovanih generičnih pogledov (Generic Views). Generični pogledi ponujajo pogosto uporabljene vzorce operacij CRUD in s tem poenostavijo proces grajenja APIja tako, da zmanjšajo potrebo po pisanju ponavljajoče kode. Prav tako omogočajo uporabo vnaprej določenih razredov, ki omogočajo olajšano upravljanje s pravicami dostopa, iskanje in paginacijo. V pogledu je potrebno tudi navesti, kateri razred naj se uporabi za serializacijo podatkov. V spodnji kodi lahko vidimo primer generičnega pogleda *ListCreateAPIView*, ki ponuja končne točke za operacije GET in POST. Preko te

končne točke smo dostopali do seznama vseh festivalov, pri čemer smo lahko uporabili filtriranje in iskanje ter dodajali nove festivale.

```
class FestivalList(generics.ListCreateAPIView):
    permission_classes = (IsAuthenticated,)
    serializer_class = FestivalSerializer
    filter_backends = [SearchFilter]
    search_fields = ['name']

    def perform_create(self, serializer):
        festival = serializer.save()
        festival.mods.add(self.request.user)

    def get_queryset(self):
        order_by = self.request.query_params.get('order')

        queryset = Festival.objects.all()
        user = self.request.user
        if order_by == 'Popularity':
            queryset = queryset.annotate(
                most_popular=Count('favourite_by')
                )
                .all()
                .order_by('-most_popular')
        else:
            queryset = queryset.order_by(
                F('date_start').desc(nulls_last=True)
            )
    ...
```

Izvorna koda 5.3: Primer pogleda za model *Festival* ki omogoča kreiranje iskanje in vračanje festivalov urejenih po popularnosti in datumu.

Po vsem tem smo morali definirati še URL končne točke za naš API. To smo opravili v datoteki *urls.py*, kjer smo za vsako točko definirami URL naslov ter pogled uporabljen za obdelavo zahtev. Poleg končnih točk za avtentikacijo za katere je poskrbela knjižnica Djoser smo implementirali še 10 ostalih končnih točk:

- festivals/
- festivals/<int:pk>
- festivals/<int:pk>/chats/
- festivals/<int:pk>/chats/<int:cpk>
- notifications/
- hotels/
- posts/
- posts/<int:pk>
- posts/<int:pk>/comments
- posts/<int:pk>/comments/<int:pk>

### 5.3.3 Strgalnik

Ena glavnih funkcionalnosti našega sistema je ta, da uporabniku omogoča prikaz prenočišč v bližini glasbenega festivala. Za implementacijo smo morali najprej določiti, od kod bomo pridobivali podatke. Najboljše bi to rešili, če bi obstajala storitev, ki bi ponujala brezplačen API za dostop do potrebnih podatkov, vendar se je izkazalo, da nobena ustrezna storitev tega ne ponuja. Za dostop do APIjev storitev kot sta Booking.com in Airbnb [3] je potrebnen dolgotrajen proces, po katerem dostop še vedno ni zagotovljen. Zato smo se odločili, da bomo podatke strgali iz spletne strani Booking.com s pomočjo Python knjižnice BeautifulSoup4. Ko uporabnik na čelni strani vnese parametre za iskanje, skripta na zalednem delu sestavi URL naslov, v taki obliki, kot če bi brskali po Bookingu, ter prebere vsebino strani in vrne rezultat nazaj odjemalcu.

## 5.4 Implementacija Mapbox zemljevida

Zemljevide smo v celoti implementirali na čelnem delu aplikacije s storitvijo Mapbox. Za dostop do storitve smo ustvarili uporabniški račun, da smo pridobili žeton za dostop ter prenesli modul *nuxt-mapbox*. Nato smo ustvarili novo instanco zemljevida z začetnimi nastavitvami:

- **container** - Referenca na HTML element, ki vsebuje zemljevid.
- **style** - Izbor predloge zemljevida. Izbrali smo predlogo Mapbox Streets, ki se osredotoča na prikaz cestišč.
- **center** - Geografske koordinate, ki bodo v začetku centrirane. V našem primeru te koordinate predstavljajo lokacijo prizorišča.
- **zoom** - Začetna povečava zemljevida. Izbrali smo stopnjo 16, saj omogoča tako pregled nad prizoriščem, kot neposredno okolico.

Za osnovno delovanje Mapbox zemljevida v naši aplikaciji zadošča že nekaj vrstic kode:

```
new mapboxgl.Map({
    container: 'map',
    style: 'mapbox://styles/mapbox/outdoors-v12',
    center: [lon, lat],
    zoom: 16
})

let el = document.createElement('div')
el.className = 'marker img1'
const marker = new mapboxgl.Marker(el)
    .setLngLat([lon, lat])
    .addTo(map)
```

Izvorna koda 5.4: Primer kode za prikaz lokacije na Mapbox zemljevidu.

Zanimivejša je implementacija iskanja poti do festivala. Za to smo potrebovali Mapbox Directions API v kombinaciji z Mapbox Search Box vnosnim poljem. Mapbox Directions API [6] nam je izračunal optimalne poti za

vožnjo, ter nam vrnil podatke v obliki JSON z vsemi pripadajočimi koordinatami, časi, dolžinami in ostalimi metapodatki. Mapbox Search Box pa nam je omogočil, da lahko uporabnik izbere izhodiščno lokacijo preko vnosnega polja v človeku prijaznem zapisu.

### 5.4.1 Izbira izhodiščne lokacije

Uporabnik lahko izhodiščno lokacijo izbere na 2 načina. Prvi način je ta, da uporabnik preko vnosnega polja poišče lokacijo. Ob izbiri lokacije, se sproži dogodek *retrieve*, preko katerega lahko potem dostopamo do lokacije in izluščimo koordinate.

Druga opcija je, da uporabnik dovoli uporabo lokacije v aplikaciji. S to izbiro, je za izris poti potem potreben le klik gumba.

Da smo pridobili podatke pa je bilo potrebno izvesti klic na Mapbox Directions API.

```
const response = $fetch(
    `https://api.mapbox.com/directions/v5/mapbox/driving/
    ${startLocation.lon}%2C${startLocation.lat}%3B
    ${props.lon}%2C${props.lat}?
    alternatives=false
    &geometries=geojson
    &language=en
    &overview=simplified
    &steps=true
    &access_token=${runtimeConfig.public.mapboxToken}`
)
```

Izvorna koda 5.5: URL za klic Mapbox Directions APIja ki vrne navodila za pot v formatu GeoJSON.

Klic APIja vrne odgovor z navodili za vsak zavoj in vmesne točke, skupaj s pripadajočimi koordinatami. Poleg tega dobimo tudi podatke o času trajanja vožnje v sekundah, ter dolžine poti v metrih. Te podatke je potrebno nato še prikazati na zemljevidu.

```
map.on('load', () => {
    map.addSource('route', {
        'type': 'geojson',
        'data': {
            'type': 'Feature',
            'properties': {},
            'geometry': {
                'type': 'LineString',
                'coordinates': props.directions.geoLines
            }
        }
    });
    map.addLayer({
        'id': 'route',
        'type': 'line',
        'source': 'route',
        'layout': {
            'line-join': 'round',
            'line-cap': 'round'
        },
        'paint': {
            'line-color': '#26A69A',
            'line-width': 5,
        }
    })
})
```

Izvorna koda 5.6: Izvorna koda za prikaz poti na zemljevidu.

Ko je pot prikazana na zemljevidu, je potem potrebno ponovno centrirati zemljevid tako, da je vidna celotna pot.

```
const bounds = new mapboxgl.LngLatBounds()
props.directions.geoLines.forEach(([lon, lat]) => {
    bounds.extend([lon, lat])
})

const padding = { top: 50, bottom: 50, left: 50, right: 50 }
map.fitBounds(
```

```
    bounds,
    { padding }
)
```

Izvorna koda 5.7: Prilagoditev središča zemljevida in stopnje povečave da ustreza geografskemu območju poti.

## 5.5 Realnočasovna komunikacija

Pomembna stvar pri implementaciji klepetov je ta, da vsi uporabniki, ki so aktivni v določenem klepetu, takoj prejmejo novo sporočilo. Prav tako morajo uporabniki, prijavljeni na obvestila za klepet nova obvestila prejeti takoj. To smo dosegli z uporabo storitve Pusher Channels, ki sloni na tehnologiji WebSockets. Kako smo to vključili v naš sistem pa bomo opisali v naslednjem poglavju.

Za delovanje Pusher Channels smo na zalednem delu najprej namestili knjižnico *pusher*, na čelnem pa *pusher-js*. Nato smo ustvarili Pusher račun, da smo lahko pridobili pooblastila za uporabo Pusherja - *app_id, key in secret*. Na zalednem delu smo nato ustvarili novo datoteko z Izvorno kodo **??**, ki inicializira novo instanco Pusherja. Podobno smo storili tudi na čelnem delu, kjer pa je bilo potrebno kot parameter podati le *key*.

```
import pusher, os

pusher = pusher.Pusher(
    app_id=os.environ.get('PUSHER_APP_ID'),
    key=os.environ.get('PUSHER_KEY'),
    secret=os.environ.get('PUSHER_SECRET'),
    cluster='eu',
    ssl=False
)
```

Izvorna koda 5.8: Izvorna koda ki kreira novo instanco Pusherja na zalednem delu.

### 5.5.1 Sporočila

Najprej smo implementirali realnočasovno sporočanje. Za začetek smo metodo *perform_create* v pogledu *MessageList* prilagodili tako, da poleg kreiranja novega objekta, takoj odda sporočilo v kanal klepeta na Pusherju. Potrebno je bilo podati ime kanala, ki smo ga poimenovali kar *chat-'ime-klepeta'*, ime dogodka, ki smo ga poimenovali kar *'new-message'*, ter vsebino samega sporočila.

```
data = serializer.data
pusher.trigger(
    f'chat-{message.chat.name}',
    'new-message',
    data
)
```

Izvorna koda 5.9: Pošiljanje novega sporočila vsem odjemalcem naročenim na kanal chat-message.chat.id z uporabo storitve Pusher.

S tem smo pokrili zaledni del, implementirati pa je bilo potrebno še prejemanje teh sporočil na čelnem delu. To smo naredili tako, da smo najprej ustvarili Nuxt vtičnik z instanco Pusherja. Do tega vtičnika smo potem lahko dostopali preko celotne aplikacije in preprečili možnost, da bi brez potrebe kreirali več instanc Pusherja, kot bi jih potrebovali. To nam je tudi omogočilo, da smo z eno instanco Pusherja upravljali tako nova sporočila, kot obvestila.

```
import Pusher from 'pusher-js'

export default defineNuxtPlugin(nuxtApp => {
    const pusher = new Pusher(
        runtimeConfig.public.pusherKey, {
            cluster: 'eu'
        })

    nuxtApp.provide('pusher', pusher)
})
```

Izvorna koda 5.10: Nuxt vtičnik za novo instanco Pusherja.

Vse kar je preostalo je bila prijava na poslušanje novih sporočil. Za to se je bilo potrebno naročiti na kanal za določen klepet, ter poslušati na dogodek *'new-message'*, ter novo sporočilo na strani odjemalca dodati med obstoječa.

```
const channel = useNuxtApp().$pusher.subscribe(
    `chat-${chatName}`
)
channel.bind('new-message', (data) => {
    apiMessages.value.push(data)
})
```

Izvorna koda 5.11: Poslušanje na nova sporočila na čelnem delu.

### 5.5.2 Notifikacije

Podobno kot smo implementirali prejemanje sporočil v realnem času, smo implementirali tudi obvestila. Razlika je bila le ta, da smo na zalednem delu najprej iterirali čez vse klepete za katere ima uporabnik vklopljena obvestila, in se za vsakega posebej naročili na nov kanal.

## 5.6 Večnivojski komentarji

Implementacija foruma, je bila časovno najbolj potratna, zaradi velikega števila funkcionalnosti, ki jih podpira. Za delovanje foruma smo v Djangu potrebovali modele *CustomUser*, *Festival*, *Post* in *Comment*, ter vse potrebne vmesne modele za všečkanje objav in komentarjev.

Največji izziv pri implementaciji foruma je bil, kako omogočiti gnezdenje komentarjev. V sistemu, ki omogoča gnezdenje komentarjev, lahko uporabnik odgovori na predhodnje komentarje, pri čemer se ustvarja hierarhija komentarjev. Prednost tega pristopa je predvsem lažja berljivost, saj se s tem izognemo zmedi kateremu komentarju pripada odgovor.

Za hranjenje hierarhije komentarjev smo na zalednem delu definirali model *Comment* in uporabili samoreferenčni tuji ključ. To storimo tako, kot kadar definiramo relacije mnogo proti ena, le da za referenco uporabimo kar besedo *'self'* [22]. S tem smo dosegli, da ima lahko en komentar enega samega starša in več odgovorov, starš pa je lahko tudi *null*, kar definira komentarje na najvišjem nivoju.

```
parent = models.ForeignKey(
    'self',
    null=True,
    blank=True,
    related_name='child_comments',
    on_delete=models.CASCADE
)
```

Izvorna koda 5.12: Definiranje hierarhije komentarjev v modelu Comment z uporabo samoreferenčnega tujega ključa.

Potem je bilo potrebno poskrbeti še za ustrezno serializacijo podatkov, za kar smo napisali rekurzivno metodo *get_child_comments*, ki za določen komentar zbere vse pripadajoče odgovore in jih sestavi v formatu JSON za prenos do odjemalca.

```
def get_child_comments(self, obj):
    child_comments = obj.child_comments.all()
    return CommentSerializer(
        child_comments,
        context=self.context,
        many=True
    ).data
```

Izvorna koda 5.13: Rekurzivna metoda za serializacijo komentarjev v obliki nadrejen/podrejen.

Potem pa še ustrezen pogled, ki ob klicu na končno točko vrne vse komentarje na najvišjem nivoju skupaj z pripadajočimi odgovori.

```
class CommentList(generics.ListCreateAPIView):
    serializer_class = CommentSerializer
```

```
    def get_queryset(self):
        post = self.kwargs['pk']
        return Comment.objects.filter(
            post=post,
            parent__isnull=True
        )
```

Izvorna koda 5.14: Pogled ki s filtriranjem izbere samo komentarje na najvišjem nivoju.

Nato je bilo potrebno komentarje prikazati še na strani odjemalca, kjer smo ustvarili rekurzivno komponento *Comment*. Za prikaz odgovorov na komentarje ter zamik na vsakem nivoju, smo uporabili spremenljivko *level*, ki smo jo podali kot parameter rekurzije.

```
<div
    v-if="comment.child_comments
    && comment.child_comments.length
    && showReplies"
>
    <Comment
        v-for="(childComment, childIndex)
            in comment.child_comments"
        :key="childComment.id"
        :index="childIndex"
        :comment="childComment"
        :level="level + 1"
    />
</div>
```

Izvorna koda 5.15: Rekurzivna komponenta za prikaz komentarjev na strani odjemalca.

Ostala implementacija je bila dokaj preprosta. Po tem, ko smo omogočili komentiranje, smo dodali še možnosti iskanja in urejanja komentarjev po času, ter po številu všečkov. Implementirali pa smo tudi paginacijo, da smo velike množice podatkov lahko razbili na manjše dele in jih na bolj učinkovit

poslali do odjemalca.

## 5.7 Firebase

Za shranjevanje slik smo uporabili Firebase Storage, ker omogoča varno ter enostavno shranjevanje in branje datotek in lahko integracijo v aplikacijo. Enako kot pri Mapboxu ter Pusherju se je bilo najprej potrebno registrirati, da smo dobili vse potrebne žetone. Potem smo ustvarili nov Nuxt vtičnik, ki hrani instanco Firebasa, da lahko do njega dostopamo v katerem koli delu aplikacije.

Sliko smo v Firebase Storage shranili tako, da smo najprej definirali pot, kjer bo slika shranjena v oblaku, ter referenco do te shrambe. Da ne bi prišlo do nepotrebnih težav, je bilo potrebno poskrbeti, da se nobeno ime slike ne ponovi. To smo rešili tako, da smo vsako sliko poimenovali z imenom slike in trenutnim časom v milisekundah. Potem smo z uporabo metode *uploadBytes* naložili sliko v shrambo. V primeru, ko se slika uspešno shrani, dobimo nazaj URL naslov za prenos datoteke. Ta URL naslov smo potem shranili v našo podatkovno bazo, da smo lahko v aplikaciji dostopali do slike.

```
const uploadImage = async() => {
    const storage = useNuxtApp().$storage
    const fileName = `festival-themes/
        ${Date.now()}-${festival.value.img.name}`
    const imageRef = storageRef(storage, fileName)

    try {
        await uploadBytes(imageRef, festival.value.img)
        const url = await getDownloadURL(imageRef)
        return url
    }
    catch (error) {
        console.log("Error uploading image")
    }
}
```

Izvorna koda 5.16: Nuxt vtičnik za shranjevanje slik z uporabo Firebase Storage.

# Poglavje 6

# Predstavitev aplikacije

V naslednjem poglavju smo predstavili razvito aplikacijo z vidika uporabnika. Opisali smo funkcionalnosti, ki jih uporabniški vmesnik omogoča, pri čemer smo si pomagali s slikami.

## 6.1 Uporabniški vmesnik

Uporabniški vmesnik aplikacije je sestavljen iz štirih glavnih strani. To so domača začetna stran, stran za vsak festival ter pripadajoči forum in klepeti. Poleg teh strani aplikacija vsebuje še strani za dodajanje in urejanje festivalov, objav in klepetov, ter registracijo in prijavo. Vse strani (razen registracije in prijave) vsebujejo še orodno vrstico, preko katere se lahko uporabniki odjavijo iz aplikacije ter dostopajo do obvestil, foruma in domače strani.

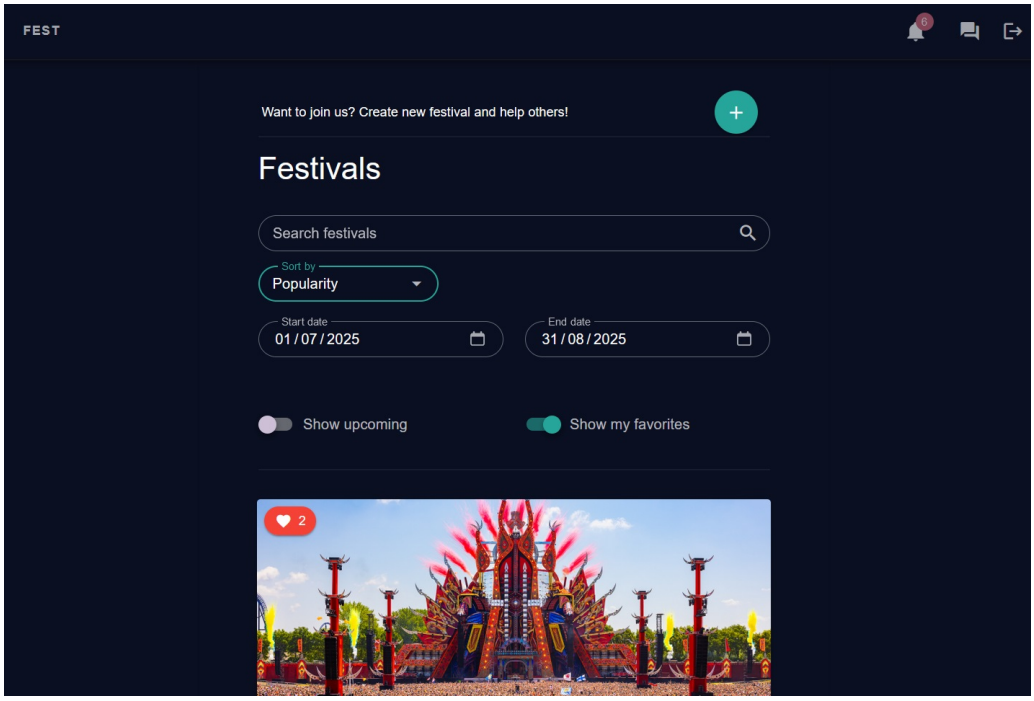
### 6.1.1 Domača stran

Domača stran je namenjena temu, da lahko uporabniki najdejo glasbene festivale in dodajo nove. Za lažje iskanje uporabniku relevantnih festivalov, smo implementirali naslednje možnosti iskanja in filtriranja:

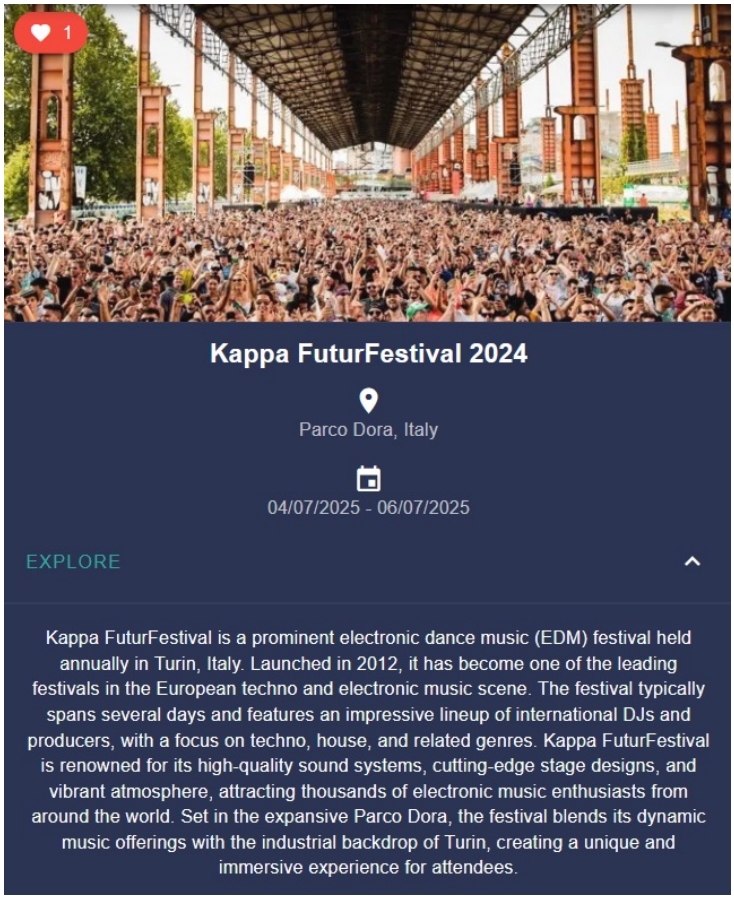
- Iskalno okno, preko katerega lahko iščejo po imenu festivala ter lokaciji.

- Spustni seznam preko katerega lahko uredijo izpis po datumu in popularnosti.
- Omejitev časovnega obdobja v katerem poteka festival.
- Izpis samo prihajajočih dogodkov.
- Izpis samo uporabnikovih priljubljenih festivalov.

Rezultati iskanja so prikazani na karticah, in vsebujejo ime, lokacijo in datume festivala. Če je na voljo, se prikaže tudi slika festivala. V levem zgornjem kotu kartice je za vsak festival prikazano, koliko uporabnikov ima festival dodan med priljubljene. V spodnjem desnem kotu lahko uporabnik razširi kartico, in s tem prikaže še podrobnejši opis. S klikom na gumb 'EXPLORE' v spodnjem levem kotu, pa se odpre stran za izbrani festival.




Slika 6.1: Domača stran spletne aplikacije.




Slika 6.2: Kartica za festival na domači strani.

Poleg iskanja začetna stran omogoča tudi dodajanje novega festivala. S klikom na gumb z oznako +, se odpre novo okno z obrazcem. Tu uporabnik vnese vse potrebne podatke, vključno z lokacijo, katero lahko vnese preko zemljevida.

Festivale lahko v sistem zaenkrat dodajo vsi uporabniki, kar omogoča lažjo pridobitev manjših in manj poznanih festivalov. Uporabnik, ki doda festival avtomatsko postane moderator tega festivala. To mu omogoča, da lahko ureja stran festivala in vse pripadajoče forume. V prihodnosti bi bilo smiselno dodati tudi možnost uradnih partnerjev. Tako bi lahko organizatorji glasbenih festivalov stopili v kontakt z nami, šli čez določen proces in postali

partnerji, s čimer bi aplikacija pridobila točnejše informacije in morebitno dodatno vsebino, ki bi lahko še okrepila uporabniško izkušnjo.

## 6.2 Festivalska stran

Na festivalski strani so prikazane podrobnosti določenega festivala. Uporabnik lahko tu najde informacije o lokaciji (z zemljevidom), žanrih, datumih, uradni spletni strani in ostalem. Od tu dobi tudi povezavo do foruma in klepetov za izbrani festival. Za boljšo uporabniško izkušnjo in kompletno rešitev, pa lahko uporabnik razširi zavihek 'Sleeping options and directions', kjer lahko poišče prenočišča v bližini, ter navodila za pot do festivala.

### 6.2.1 Prenočišča v bližini

V primeru, da uporabnik izbere možnost iskanja prenočišč v bližini glasbenega festivala, se odpre okno preko katerega lahko vnese različne parametre za iskanje. Izbere lahko število ljudi, število sob, datum prijave, datum odjave in najvišjo ceno prenočišča na noč. Določi pa lahko tudi kateri rezultati se prikažejo prvi, pri čemer lahko izbira med najcenejši najprej, najbolje ocenjeni naprej, najbolje ocenjeni z nizko ceno najprej, ter najkrajša pot do festivala najprej. Datumi prijave in odjave so privzeto nastavljeni na prvi in zadnji dan festivala, po tem pa lahko uporabnik poljubno podaljša ali skrajša termin. V primeru da festival že poteka, je uporabnik o tem tudi obveščen, datum prijave pa postane tekoči dan. Tudi v primeru da novi datumi za festival še niso določeni, lahko uporabnik poišče prenočišča, datum prijave postane tekoči dan, datum odjave pa 3 dni za tem.

Rezultati poizvedbe so prikazani na karticah, na katerih za vsako prenočišče najdemo podatke o imenu prenočišča, ceni na noč, kraju, povprečni oceni in številu ocen, ter oddaljenosti od prizorišča festivala. S klikom na gumb 'DETAILS', je uporabnik preusmerjen na stran prenočišča na Bookingu.

### 6.2.2 Navodila za pot

Uporabnik lahko pridobi tudi navodila za pot do glasbenega festivala. Za izbor začetne lokacije ima na voljo dve možnosti. Prva je ta, da vnese lokacijo preko vnosnega polja, pri čemer se mu lokacije ob vnosu teksta avtomatsko predlagajo. Druga možnost pa je, da uporabnik dovoli aplikaciji dostop do lokacije. S to možnostjo se prikaže gumb, s katerim lahko uporabnik zahteva navodila za pot s svoje trenutne lokacije.

Po izboru ene izmed teh možnosti, se zemljevid z lokacijo festivala posodobi v zemljevid s potjo od izbrane lokacije do prizorišča. V levem zgornjem kotu se izpiše čas trajanja vožnje in dolžina v kilometrih, nad zemljevidom pa se izpišejo podrobna navodila za pot.

## 6.3 Forum

Forum je namenjen temu, da uporabniki delijo novice, mnenja in nasvete, lahko pa tudi postavijo vprašanja glede stvari, ki jih zanimajo oz. jim niso jasna. S tem jim omogočimo, da se povežejo in sodelujejo, ter si pri tem pomagajo. Kar pa je najpomembneje je to, da na ta način uporabniki dobijo mnenja ostalih uporabnikov. Takšna mnenja največ štejejo, saj lahko tisti, ki so že obiskali festival podajo največ praktičnih nasvetov.

### 6.3.1 Objave

Vsaka objava je predstavljena na kartici, ki je ločena na 2 dela.

Zgornji levi del kartice vsebuje informacije o festivalu, času objave, lastniku objave, ter tipu objave. Zgornji desni del kartice pa vsebuje meni, ki omogoča brisanje (lastnik in moderator) in urejanje (lastnik) objav.

Spodnji del kartice vsebuje naslov in vsebino objave, pod njima pa najdemo še gumba za všečkanje in nevšečkanje, ter gumb za dostop do komentarjev, ki odpre objavo na novi strani. Vsi gumbi vsebujejo tudi podatke o številu všečkov in nevšečkov, ter številu komentarjev za objavo.

Uporabnik lahko tudi brska po objavah. To mu omogoča iskalno polje preko katerega lahko išče objave po naslovu, vsebini in uporabniku. Na voljo so tudi filtri za določen tip objave, zato da lahko uporabnik izbere samo kategorijo objav, ki ga v trenutku zanima. Rezultate je mogoče razvrstiti po naslednjih kriterijih:

- **Top** - Največje število všečkov
- **Most commented** - Največje število komentarjev
- **Best** - Najboljše razmerje med všečki in nevšečki
- **New** - Najnovejši na začetku
- **Controversial** - Urejeni tako, da so na vrhu prikazane tiste objave, ki imajo razmerje všečkov in nevšečkov približno 1:1

Razvrstitev objav po kontroverznosti je lahko uporabniku v pomoč, saj niso vsa mnenja vseh ljudi vedno enaka. Morda bo nekomu neka informacija koristila, nekomu pa ne.

### 6.3.2 Komentarji

Komentarji so ključni del foruma, saj omogočajo, da vsi uporabniki podajo svoje mnenje na objavo. Brez komentarjev, bi bila komunikacija le enostranska in posledično ne bi bila v veliko pomoč.

Komentarje za objavo dosežemo tako, da kliknemo na izbrano objavo. Po izbiri objave, se objava v isti obliki odpre na novi strani. Pod objavo je tekstovno polje, ki se ob kliku spremeni v besedilno polje, preko katerega lahko uporabnik doda komentar, ter spustni seznam, preko katerega lahko uporabnik razvrsti komentarje po novejših ali pa priljubljenejših najprej. Sledijo vsi komentarji, ki pripadajo objavi.

Komentarji so prikazani podobno kot objave, le da zgornji del kartice vsebuje samo ime uporabnika in čas objave. Spodnji del vsebuje vsebino

komentarja, všečke in nevšečke, ter gumb s katerim lahko uporabnik odgovori na komentar.

Kot smo že omenili so komentarji prikazani večnivojsko. Vsak odgovor na komentar je na zaslonu zamaknjen za nekaj pikslov v desno, kar uporabniku omogoča lažji pregled, še posebej v primerih, ko je pod objavo veliko število komentarjev. Privzeto sta prikazana samo prva dva nivoja komentarjev, sepravi vsi korenski komentarji vključno z odgovori. S pomočjo gumba pod komentarji, ki vsebujejo odgovore pa lahko naknadno uporabnik skrije ali prikaže komentarje.

## 6.4 Klepetalnica

Klepetalnica je namenjena temu, da se uporabniki lahko povežejo med sabo. Vsak festival ima svojo pripadajočo klepetalnico, v kateri lahko uporabniki razpravljajo v živo. Tu imajo uporabniki odprte roke pri uporabi, ker so javne pa se lahko klepetom pridruži vsak. Tako lahko uporabniki med drugim:

- Ustvarijo klepet, v katerem se zberejo uporabniki aplikacije iz neke države.
- Tisti, ki potujejo sami lahko ustvarijo klepet z namenom, da najdejo ljudi za skupni obisk in planirajo srečanja.
- Med potekom festivala lahko razpravljajo o novicah ali pa novostih in spremembah glede na prejšnja leta.
- Z ostalimi delijo informacije o tem, kje lahko najdejo različne stojnice s hrano in pijačo, kje lahko prevzamejo festivalske zapestnice ipd.
- Razpravljajo o nastopajočih ali pa delijo vtise iz prejšnjih dni.

### 6.4.1 Klepet

Do klepetov lahko uporabnik dostopa preko strani izbranega festivala, kjer najprej vidi seznam vseh obstoječih klepetov. Potem lahko izbira med doda-

janjem novega klepeta, ali uporabe že obstoječega.

Vsak klepet je zgrajen iz:

- Naslovne vrstice z imenom klepeta in gumbom za prijavo/odjavo na obvestila.
- Okna kjer so prikazana vsa sporočila.
- Vrstice z vnosnim poljem in gumbom za pošiljanje sporočil.

Okno, kjer so prikazana vsa sporočila v klepetu je sestavljeno tako, da je razvidno komu sporočilo pripada. Uporabnik vidi svoja sporočila poravnana na desno stran klepeta, sporočila ostalih pa so poravnana na levo, skupaj z ikono, ki predstavlja uporabnika. Sporočila so lahko tekstovna ali pa slikovna. Nad vsakim sporočilom poslanim po daljšem obdobju od prejšnjega, izpiše, kdaj je bilo sporočilo poslano. Uporabnik lahko pošlje tekstovno ali slikovno sporočilo, lahko pa tudi oboje hkrati. Ob kliku na tekstovno sporočilo se poleg sporočila prikažeta točen datum in ura ob katerem je bilo sporočilo poslano. Klik na sliko, pa odpre sliko čez celoten zaslon, hkrati pa se na dnu slike izpišeta ime slike ter datum in ura sporočila. Če sporočilo vsebuje tako tekst kot sliko, je prikaz isti kot pri slikovnih sporočilih, le da se pod sliko izpiše še tekst.

Sporočila se prenašajo realnočasovno, se pravi, da so uporabnikom na voljo takoj ko so poslana. Ko je v klepet dodano novo sporočilo, ali pa se uporabnik pomakne v klepetu navzgor, se na dnu klepeta prikaže gumb, ki mu omogoči da se vrne nazaj na dno klepeta.

Za nova sporočila je na dnu zaslona uporabniku na voljo vrstica sestavljena iz gumba za izbiro datoteke, tekstovnega polja in gumba za pošiljanje. Gumb za pošiljanje je onemogočen dokler ni izbrana slika ali pa je vpisan tekst.

### 6.4.2 Obvestila

Pomemben del klepetov so tudi obvestila, ki se prav tako kot pošiljanje sporočil osvežujejo v realnem času. Uporabnik se lahko poljubno prijavi

(in odjavi) na obvestila za klepete katerim želi slediti tako, da v naslovni vrstici klepeta klikne na ikono s sliko zvonca, zraven katerega piše tudi število klepetov z novimi obvestili. Obvestilo o novem sporočilu prejmejo vsi uporabniki razen pošiljatelja. Do novih obvestil lahko dostopamo preko ikone z zvoncem v orodni vrstici aplikacije. Za vsak klepet kjer obstaja novo obvestilo, se izpiše ime klepeta ter čas prejetega obvestila. Prejetih obvestil za en klepet je lahko več, zato se poleg že omenjenega izpiše tudi podatek o številu novih sporočil. S klikom na obvestilo, se odpre klepet, nad zadnjim prebranim sporočilom pa tekst, ki obvešča katero sporočilo je bilo nazadnje prebrano.

# Poglavje 7

# Zaključek

V diplomski nalogi smo predstavili razvoj spletne aplikacije za glasbene festivale. Cilj aplikacije je bil uporabnikom olajšati iskanje informacij povezanih s festivali in ustvariti omrežje, v katerem si lahko uporabniki pomagajo med sabo. To nam je tudi uspelo, saj smo z implementacijo uporabnikom ponudili rešitev, ki jim poleg iskanja glasbenih festivalov omogoča tudi lažje planiranje, ter komunikacijo z ostalimi uporabniki. Pri tem lahko med festivali izbirajo s pomočjo različnih filtrov, za vsak festival pa lahko poiščejo prenočišča v bližini ter navodila za pot. Istočasno jim sistem omogoča, da preko foruma postavljajo vprašanja in delijo nasvete ter novice. Preko klepetov pa lahko z ostalimi komunicirajo v realnem času, se morda dogovorijo za skupni obisk festivala, ali pa se pogovarjajo med samim festivalom ter tako popestrijo festivalsko izkušnjo. Implementirali smo tudi obvestila, s čimer smo uporabniku omogočili, da je na tekočem z vsemi klepeti katerim želi slediti. Pomembno je tudi to da smo razvili uporabniški vmesnik, ki je intuitiven in enostaven za uporabo, njegova odzivnost pa omogoča, da se lepo prikaže na napravah vseh velikosti.

Ideja za aplikacijo izhaja iz mojih slabih izkušenj obiskov glasbenih festivalov, saj sem večkrat naletel na težave, ki jih najverjetneje ne bi nikoli imel, če bi obstajala platforma kjer bi lahko uporabniki delili svoje pretekle izkušnje in nasvete z ostalimi. Poleg tega je vedno potrebno iskati prenočišča

v bližini festivala, za kar je potrebno uporabiti enega izmed spletnih ponudnikov prenočišč ter pri tem iskati najprej lokacijo, ter nato datume, za kar naša rešitev poskrbi sama. Potrebno je le vnesti število potujočih in z enim klikom pridobiti prenočišča.

Ta aplikacija je bila moj prvi večji projekt, ki sem se ga sam lotil v času študija. Kljub začetnim problemom in ne prav dobri seznanjenosti z nekaterimi tehnologijami, je razvoj potekal iz dneva v dan boljše. Odnesel sem veliko novega znanja tako pri načrtovanju, kot razvoju čelnega in zalednega dela, ter uporabe orodij za kontejnerizacijo in upravljanje različic. Prepričan sem, da mi bo znanje ki sem ga pridobil dobro služilo tudi v prihodnosti.

## 7.1 Možnosti nadaljnjega razvoja

Aplikacija, ki smo jo razvili je že precej obsežna, še vedno pa nudi veliko možnosti za nadaljnji razvoj. Nekaj teh možnosti bomo našteli v nadeljevanju.

### 7.1.1 Uradni partnerji

Aplikacija že sedaj vsem uporabnikom omogoča dodajanje novih festivalov, bi pa to lahko nadgradili tako, da bi se povezali z organizatorji festivalov. S tem bi lahko uporabniki med drugim pridobili predhodni dostop do vstopnic in nastopajočih, ali pa interaktivne zemljevide, ki jih nekateri organizatorji ponujajo. Seveda bi lahko organizatorji z ostalimi delili tudi vse pomembne novosti in novice. Tako bi lahko organizatorji festivalov upravljali svoj festival na naši rešitvi, istočasno pa bi jih naša rešitev reklamirala. Težava, ki bi se pri tem lahko pojavila je ta, da organizatorji ne bi želeli sodelovati zaradi morebitnih kritik na svoj račun s strani uporabnikov.

### 7.1.2 Klepeti

Možnosti za nadgradnjo klepetov je kar nekaj. Najbolj očitna možnost, ki se nam ponuja je implementacija potisnih obvestil. Obvestila smo v našem sistemu že implementirali, bi jih pa lahko še nadgradili. Ta bi največ doprinesla tistim, ki bi aplikacijo uporabljali na mobilnih napravah in izboljšala uporabniško izkušnjo predvsem v primerih, ko bi uporabniki klepete uporabljali na prizorišču.

Poleg tega v sistemu trenutno ni možno vedeti, kdaj je bil uporabnik nazadnje aktiven v aplikaciji. S tem, da bi beležili aktivnost uporabnika, bi uporabnikom dalo vpogled v razpoložljivost osebe v klepetu.

### 7.1.3 Zemljevidi

Zemljevidi sedaj omogočajo iskanje poti in navodila za pot. Te bi bilo mogoče nadgraditi tako, da bi uporabniku ponudili več možnih poti do prizorišča med katerimi bi lahko izbiral. Smiselno bi bilo dodati tudi način javnega prevoza, vendar na žalost Mapbox te opcije ne ponuja.

# Literatura


[1] *About GitHub*. URL: https://github.com/about (pridobljeno 2. 8. 2024).

[2] *About the Booking.com Connectivity APIs*. URL: https://developers.booking.com/connectivity/docs (pridobljeno 6. 8. 2024).

[3] *API Terms of Service - Airbnb Help Center*. URL: https://www.airbnb.com/help/article/3418 (pridobljeno 16. 8. 2024).

[4] *beautifulsoup4 · PyPI*. URL: https://pypi.org/project/beautifulsoup4/ (pridobljeno 2. 8. 2024).

[5] *Cloud Storage for Firebase*. URL: https://firebase.google.com/docs/storage (pridobljeno 2. 8. 2024).

[6] *Directions API — API Docs — Mapbox*. URL: https://docs.mapbox.com/api/navigation/directions/ (pridobljeno 6. 8. 2024).

[7] *Django overview — Django*. URL: https://www.djangoproject.com/start/overview/ (pridobljeno 2. 8. 2024).

[8] *Docker: Accelerated Container Application Development*. URL: https://www.docker.com/ (pridobljeno 2. 8. 2024).

[9] *Git*. URL: https://git-scm.com/ (pridobljeno 2. 8. 2024).

[10] *Home - Django REST framework*. URL: https://www.django-rest-framework.org/ (pridobljeno 2. 8. 2024).

[11] *Homepage - Reddit*. URL: https://www.redditinc.com/ (pridobljeno 1. 8. 2024).

[12] *Introduction — djoser 2.2.2 documentation*. URL: https://djoser.readthedocs.io/en/latest/introduction.html (pridobljeno 3. 8. 2024).

[13] *Introduction · Get Started with Nuxt*. URL: https://nuxt.com/docs/getting-started/introduction (pridobljeno 2. 8. 2024).

[14] Salahaldin Juba, Achim Vannahme in Andrey Volkov. *Learning PostgreSQL*. Packt Publishing Ltd, 2015.

[15] *Mapbox — Maps, Navigation, Search, and Data*. URL: https://www.mapbox.com/ (pridobljeno 2. 8. 2024).

[16] *MVC vs MVT Architectural Pattern. Like any other curious person, you must... — by Tejaswi Chaudhari — GDSC UMIT — Medium*. URL: https://medium.com/dsc-umit/mvc-vs-mvt-architectural-pattern-d306a56dce55 (pridobljeno 3. 8. 2024).

[17] *Our purpose*. URL: https://www.purpose.tripadvisor.com/ (pridobljeno 1. 8. 2024).

[18] *PostgreSQL: The world's most advanced open source database*. URL: https://www.postgresql.org/ (pridobljeno 2. 8. 2024).

[19] *Pusher Channels Docs*. URL: https://pusher.com/docs/channels/ (pridobljeno 3. 8. 2024).

[20] *Search, Geocoding and Autofill Services — Mapbox*. URL: https://www.mapbox.com/search-service (pridobljeno 2. 8. 2024).

[21] *Server-Side Rendering (SSR) — Vue.js*. URL: https://vuejs.org/guide/scaling-up/ssr.html (pridobljeno 2. 8. 2024).

[22] *Typically, a self-referential foreign key is used to create the hierarchical structure*. URL: https://books.agiliq.com/projects/django-orm-cookbook/en/latest/self_fk.html (pridobljeno 10. 8. 2024).

[23] *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com/ (pridobljeno 2. 8. 2024).

[24] *Vue.js - The Progressive JavaScript Framework — Vue.js*. URL: https://vuejs.org/ (pridobljeno 2. 8. 2024).

[25] *Vuetify — A Vue Component Framework*. URL: https://vuetifyjs.com/en/ (pridobljeno 2. 8. 2024).

[26] *What are WebSockets? — Pusher*. URL: https://pusher.com/websockets/ (pridobljeno 3. 8. 2024).

[27] *What Is Containerization? — IBM*. URL: https://www.ibm.com/topics/containerization (pridobljeno 4. 8. 2024).

[28] *Woov – Supercharge your Events*. URL: https://woov.com/ (pridobljeno 1. 8. 2024).