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
DBMS	database management system	sistem za upravljanje podatkovnih baz
SVM	support vector machine	metoda podpornih vektorjev

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

Vsako leto veliko ljudi obišče kakšen glasbeni festival. Pridobivanje informacij in samo načrtovanje obiska pa lahko kar hitro postane nadležno, in posledično lahko hitro pokvari uporabniško izkušnjo, že pred samim obiskom festivala. Najprej se uporabnik odloči za enega izmed festivalov. Ponavadi to stori na podlagi različnih dejavnikov, kot so cena, lokacija in datum poteka, veliko pa pomenijo tudi predhodne izkušnje ostalih. Problem pa je, da je potrebno vse te informacije iskati po različnih spletnih straneh saj rešitve, s katero bi lahko pridobili vse informacije na enem mestu ni. Poleg tega bi bilo smiselno implementirati rešitev, kjer si lahko uporabniki medesebojno pomagajo, s tem da delijo predhodne izkušnje, novosti in mnenja v obliki foruma ali pa morda celo dobijo nove prijatelje, v živo razpravljajo in mogoče nekoč celo skupaj obiščejo kakšen festival s pomočjo klepetov v živo.

1.2 Cilji

Cilj diplomske naloge je torej razviti spletno aplikacijo, ki bo uporabnikom olajšala iskanje glasbenih festivalov in udeležbo na njih, ustvariti skupnost, v kateri lahko uporabniki izmenjujejo izkušnje, nasvete in novice, ter vzpodbu-

diti interakcijo med obiskovalci festivala v živo, ter s tem izboljšati festivalsko izkušnjo.

1.3 Struktura diplomske naloge

Poglavje 2

Obstoječe rešitve

V uporabi 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 [14]. 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 [8]. 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 realno-časovne 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 [25]. 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.
  - Realno-časovna 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 nad glasbenimi festivali z možnostjo iskanja in filtriranja.
- Dodajanje festivalov in dodelitev moderatorskih pravic.
- Izpis podrobnosti glasbenega festivala, vključno z datumi, žanri in povezavo do uradne spletne strani.
- Pogled na zemljevid z lokacijo glasbenega festivala, ki se lahko razširi v navodila za pot.
- Iskanje prenočišč v okolici glasbenega festivala glede na podane parametre.

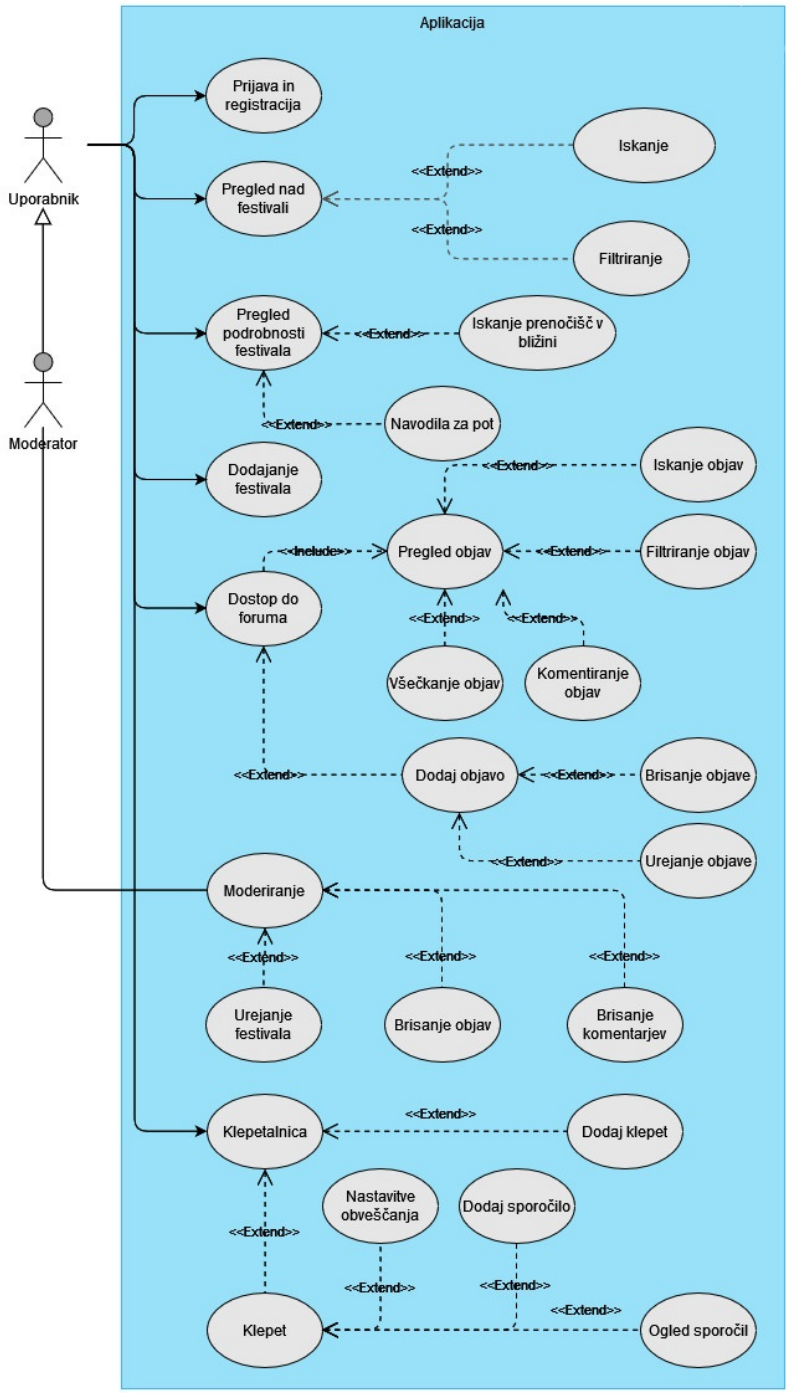
- Branje objav z možnostjo filtriranja in iskanja na forumu za določen festival.
- Objavljanje na forumih.
- Všečkanje in komentiranje objav.
- Dodajanje javnih klepetov za festivale.
- Pošiljanje sporočil v klepetih.
- Naročanje na obvestila za klepete ob novih sporočilih.

Moderatorji pa lahko povrh vsega še:

- Urejajo podatke za svoje festivale.
- Brišejo neprimerne objave.
- Brišejo neprimerne komentarje.

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 v aplikaciji počneta. Podrobneje pa bomo delovanje aplikacije 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. Za čelni del smo uporabili Nuxt [10], ki temelji na ogrodju Vue.js [21]. Za zaledni del smo izbrali Django [4] s knjižnico Django REST Framework [7], ki močno olajša razvoj API vmesnikov in upravljanje s podatkovno bazo. Za podatkovno bazo smo uporabili PostgreSQL [15]. Realnočasovno komunikacijo smo omogočili s pomočjo Pusher Channels [16]. Zemljevidi in navodila za pot so osnovani na platformi Mapbox [12]. Podatkovno bazo ter Django strežnik smo poganjali v Dockerju [5]. Vse te tehnologije in orodja ter njihov namen in uporabo v naši rešitvi smo podrobneje predstavili v naslednjem poglavju.

Poleg omenjenih glavnih tehnologij smo sicer uporabili še Vuetify.js za oblikovanje spletne strani [22], ter Git [6] in GitHub [1] za nadzor različic, za shranjevanje slik smo se odločili za Firebase Storage [3], za urejanje kode pa smo uporabljali Visual Studio Code [20]. Poleg tega smo uporabili nekaj Python knjižnic, med njimi BeautifulSoup4 [2] za strganje podatkov iz bookinga, ter Djoser [9], 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 SSR (Server-Side Rendering), ki omogoča hitrejše čase nalaganja strani in SEO (Search Engine Optimization). 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) [18].

Ena večjih prednosti 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 DRF

Django je Python ogrodje za razvoj spletnih aplikacij, ustvarjeno z namenom da uporabniku omogoči hiter razvoj, s tem, da vsebuje širok nabor orodij, ki poskrbijo za pogosto uporabljena opravila spletnega razvoja (avtentikacija, administracija, upravljanje z podatkovno bazo...). Temelji na vzorcu Model-View-Template, ki je v osnovi zelo podoben bolj znanemu Model-View-Controller. Model skrbi za podatke in vzdržuje povezavo s podatkovno bazo. View sprejema zahtevke in vrača odgovore, Template pa definira strukturo in postavitev strani, tipično z uporabo DTL (Django Template Language) [13].

Django REST Framework je razširitev za Django, posebej namenjena za gradnjo RESTful API-jev. V naši aplikaciji smo DRF v uporabili predvsem zaradi enostavnega dela s podatkovno bazo, enostavne serializacije podatkov (pretvorba kompleksnih podatkovnih tipov kot so Django modeli in poizvedbe v format JSON ali XMl) 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 [11].

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 majhno zakasnitvijo, brez da bi odjemalec poslal zahtevo za prejem podatkov [23].

Pusher Channels delujejo na modelu objavi/naroči. Aplikacije se lahko naročajo na kanale v sistemu. Ko pride do sprememb, pa sistem objavi spremembo v ta isti kanal. Vse aplikacije naročene na ta kanal, so potem obveščene [16].

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 [17]:

- **Mapbox Maps**, ki omogoča prikaz zemljevidov po meri z lokacijami. V naši aplikaciji prikazuje vse zemljevide.

- **Mapbox Directions API**, ki omogoča iskanje poti. V naši aplikaciji služi izrisovanju poti od vnešene lokacije do festivala.
- **Geocoding API**, ki spreminja koordinate v naslove in obratno. V naši aplikaciji je uporabljen, da koordinate 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. S tem ustvarimo lahko izvedljivo datoteko, ki se lahko enako izvaja na kateri koli infrastrukturi [24]. Prednosti uporabe Dockerja je veliko, v našem primeru pa smo ga uporabili predvsem zaradi izolacije okolja in lahke prenosljivosti. V Dockerju smo poganjali Django aplikacijo ter podatkovno bazo.

Poglavje 5

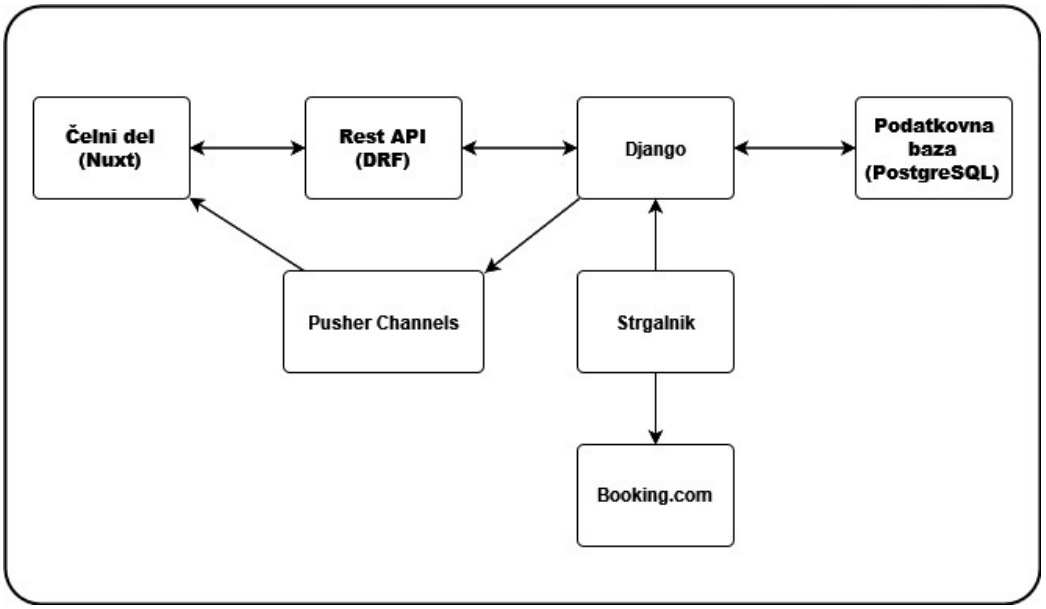
Razvoj rešitve

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 na podatkovno bazo, implementacijo nekaterih glavnih funkcij aplikacije ter uporabniški vmesnik.

5.1 Arhitektura aplikacije

Spletna aplikacija je sestavljena iz čelnega dela, kateri podatke pridobiva preko APIja implementiranega v Django Rest Frameworku. V zalednem delu Django ORM skrbi za upravljanje s podatkovnimi modeli in poizvedbami. Poleg tega skrbi za strganje podatkov o prenočiščih iz spletne strani Booking.com. Prav tako pa Django ob vsakem dodanem sporočilu v klepetih pošlje obvestilo s pomočjo Pusher Channels na čelni del aplikacije, da se uporabniku na drugi strani le to takoj prikaže.

Na Sliki **??** je prikazana arhitektura naše rešitve.



Slika 5.1: Arhitektura razvite rešitve.

5.2 Podatkovni model

5.3 Zaledni del

Za implementacijo zalednega dela smo uporabili Django v kombinaciji z Django Rest Framework.

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.

Knjižnica Django Rest Framework 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 Django ORM nam je omogočila lahko povezavo z zalednim delom aplikacije. V datoteki models.py smo definirali modele. Modeli v Djangu predstavljajo strukturo in relacije v podatkovni bazi. Vsak model predstavlja eno tabelo, vsak atribut pa eno polje v podatkovni bazi. Django ORM potem samodejno poskrbi za kreacijo tabel glede na modele, vendar je potrebno najprej pognati ukaz *python manage.py makemigrations*, da se ustvarijo migracije za vse spremembe. Da se spremembe v datoteki z migracijami dejansko uporabijo in posodobijo podatkovno bazo, pa je potrebno pognati še ukaz *python manage.py migrate*.

V spodnji Izvorni kodi **??** je prikazano, 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 lahko polje v podatkovni bazi NULL. Polje *genre* je povezava na tabelo Genre. Z *models.SET_NULL* smo določili, da se ob primeru izbrisa instance žanra v festivalih povezanih s tem žanrom polje *genre* postavi na NULL. V razredu Meta pa smo določili, da so ob poizvedbah festivali po privzetem v vrstnem redu glede na polje *name*.

```
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

Kot smo že omenili v uvodu v poglavje, smo za komunikacijo čelnega dela aplikacije z zalednim implementirali RESTful API s pomočjo Django Rest Frameworka. Na zahteve odjemalca posluša na vratih 8000.

DRF močno pospeši izgradnjo APIjev, saj je za serializacijo podatkov potrebno le definirati razred za vsak model, ter našteti katera polja bomo serializirali. S tem povemo, kako naj bodo polja modela predstavljena v obliki JSON, ki je standardni format za APIje. 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.

V Izvorni kodi **??** lahko vidimo primer serializacije podatkov za model *Festival* s tremi dodatnimi izpeljanimi vrednostmi, ki jih izračunajo metode:

- **is_favourite** - Glede na uporabnika, ki zahteva podatke in izbrani festival, vrne vrednost True ali False glede na to ali ima uporabnik festival dodan med priljubljene.

- **get_is_mod** - Glede na uporabnika, ki zahteva podatke in izbrani festival, vrne vrednost True ali False glede na to 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, DRF omogoča tudi tako imenovane generične poglede (Generic Views). Ti pogledi ponujajo pogosto uporabljene vzorce operacij CRUD na modelih in s tem poenostavijo proces grajenja APIja tako, da se zmanjšajo potrebo po pisanju ponavljajoče kode. Prav tako omogočajo uporabo predefiniranih razredov s pravicami dostopa, iskanje po poljih modela ter paginacijo. V Izvorni kodi **??** lahko vidimo primer generičnega pogleda *ListCreateAPIView*, ki ponuja končne točke za operacije GET in POST. Preko te končne točke, lahko torej dostopamo do seznama vseh festivalov s filtri in opcijo iskanja ter določitve vrstnega reda rezultatov. Lahko pa na tej točki tudi kreiramo nov festival.

```
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.

Za delovanje APIja je potrebno poleg definicije modela, serializacije in pogleda za definirati še URL končne točke. To lahko storimo v datoteki *urls.py*, kjer definiramo URL naslov za vsako točko. V naši aplikaciji smo poleg končnih točk za avtentikacijo implementirali 10 končnih točk in sicer:

- `festivals/`
- `festivals/{int:pk}`
- `hotels/`
- `festivals/{int:pk}/chats/`
- `festivals/{int:pk}/chats/{int:cpk}`
- `notifications/`
- `posts/`
- `posts/{int:pk}`
- `posts/{int:pk}/comments`
- `comments/{int:pk}`

5.3.3 Strgalnik

Ena glavnih funkcij našega sistema je, da uporabniku omogoča prikaz prenočišč v bližini glasbenega festivala. Za implementacijo je bilo potrebno najprej ugotoviti, s kje bo naša aplikacija črpala podatke. Najlažje bi bilo, da bi obstajala storitev, ki bi ponujala brezplačen API, preko katerega bi lahko dostopali do podatkov, ki jih potrebujemo. Izkazalo se je, da nobena izmed storitev, ki bi nam ustrezala, tega ne ponuja. Za dostop do APIja spletnih

strani Booking.com in Airbnb je tako potrebno čez dolgotrajen proces, po katerem šele lahko potem postanemo uradni partner.

Zaradi navedenega problema smo se odločili, da bomo podatke strgali iz spletne strani Booking.com. Strganje podatkov smo izvedli s pomočjo Python knjižnice BeautifulSoup4. Uporabnik na čelni strani pri izbranem festivalu vnese parametre za iskanje. Na podlagi teh parametrov potem na zalednem delu skripta sestavi URL naslov, v taki obliki, kot če bi brskali po Bookingu, ter prebere vsebino strani in vrne rezultat v obliki JSON nazaj na čelni del.

5.4 Implementacija Mapbox zemljevida

Vsi zemljevidi so v celoti implementirani na čelnem delu aplikacije. Preden lahko uporabljamo Mapbox je potrebno ustvariti uporabniški račun, da lahko pridobimo žeton za dostop. Za osnoven zemljevid je potrebnih le nekaj vrstic kode, še pred tem pa je potrebno prenesti modul *nuxt-mapbox*. Nato ustvarimo novo instanco zemljevida, pri tem pa v osnovi dodamo naslednje vrednosti:

- **container** - referenca na HTML element, ki bo vseboval zemljevid
- **style** - izbor izgleda zemljevida
- **center** - koordinate, ki bodo centrirane
- **zoom** - začetna povečava zemljevida

```
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.

Bolj zanimiva pa je bila implementacija iskanja poti do festivala. Za to smo uporabili Mapbox Directions API v kombinaciji z Mapbox Search Box vnosnim poljem. Z uporabo Mapbox Directions API [**mapboxdirections**] smo lahko izračunali optimalne poti za vožnjo, ter prikazali navodila za pot do cilja. Search Box pa omogoča, da uporabnik izbere izhodiščno lokacijo.

5.4.1 Izbira izhodiščne lokacije

Uporabnik lahko izhodiščno lokacijo izbere na 2 načina. Pri prvem načinu uporabnik preko vnosnega polja Mapbox Search Box poišče lokacijo. Ob izbiri, se sproži dogodek "retrieve", preko katerega lahko potem dostopamo do lokacije in izluščimo koordinate.

Druga opcija je, da uporabnik dovoli uporabo lokacije v aplikaciji. Če to stori, lahko le s klikom na gumb pridobi pot od svoje trenutne lokacije do prizorišča.

Klic APIja izgleda tako:

```
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.

V odgovoru dobimo vsa navodila za vsak zavoj in vmesne točke, skupaj z 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. To smo naredili z naslednjo kodo:

```
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.

S tem smo dodali pot na zemljevid, vendar je bilo potrebno potem 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

Pri implementaciji klepetov smo morali zagotoviti, da vsi uporabniki, ki so aktivni v določenem klepetu, prejmejo novo sporočilo takoj. To smo dosegli z uporabo 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 najprej namestili knjižnici **pusher** na zalednem delu ter **pusher-js** na čelnem delu. Potem smo ustvarili Pusher račun, da smo lahko pridobili pooblastila za uporabo Pusherja - *app_id, key in secret*. Na zalednem delu smo potem ustvarili novo datoteko s 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 in shrambe v podatkovno bazo, takoj odda sporočilo vsem prijavljenim v kanalu klepeta. Potrebno je bilo podati ime kanala, ime dogodka (novo sporočilo), ter vsebino samega sporočila.

```
pusher.trigger(
    f'chat-{message.chat.id}',
    '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 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 bilo potem še potrebno je bilo poslušati na nova sporočila. Za to se je bilo potrebno naročiti na kanal za določen klepet, ter poslušati na dogodek 'new-message'.

```
const channel = useNuxtApp().$pusher.subscribe(`chat-${chatId}`)
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 za prejeta sporočila. Razlika je bila le ta, da smo 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.

Problem, ki se je pojavil pri implementaciji foruma, je bil ta, kako omogočiti večnivojsko komentiranje.

Poglavje 6

Uporabniški vmesnik

Poglavje 7

Zaključek

7.1 Nadaljnji razvoj

Zaradi kompleksnosti projekta, je možnosti za nadgradnje projekta veliko.

Literatura

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

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

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

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

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

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

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

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

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

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

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

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

[13] *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).

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

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

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

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

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

[19] *TypeScript: JavaScript With Syntax For Types.* URL: https://www.typescriptlang.org/ (pridobljeno 2. 8. 2024).

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

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

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

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

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

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