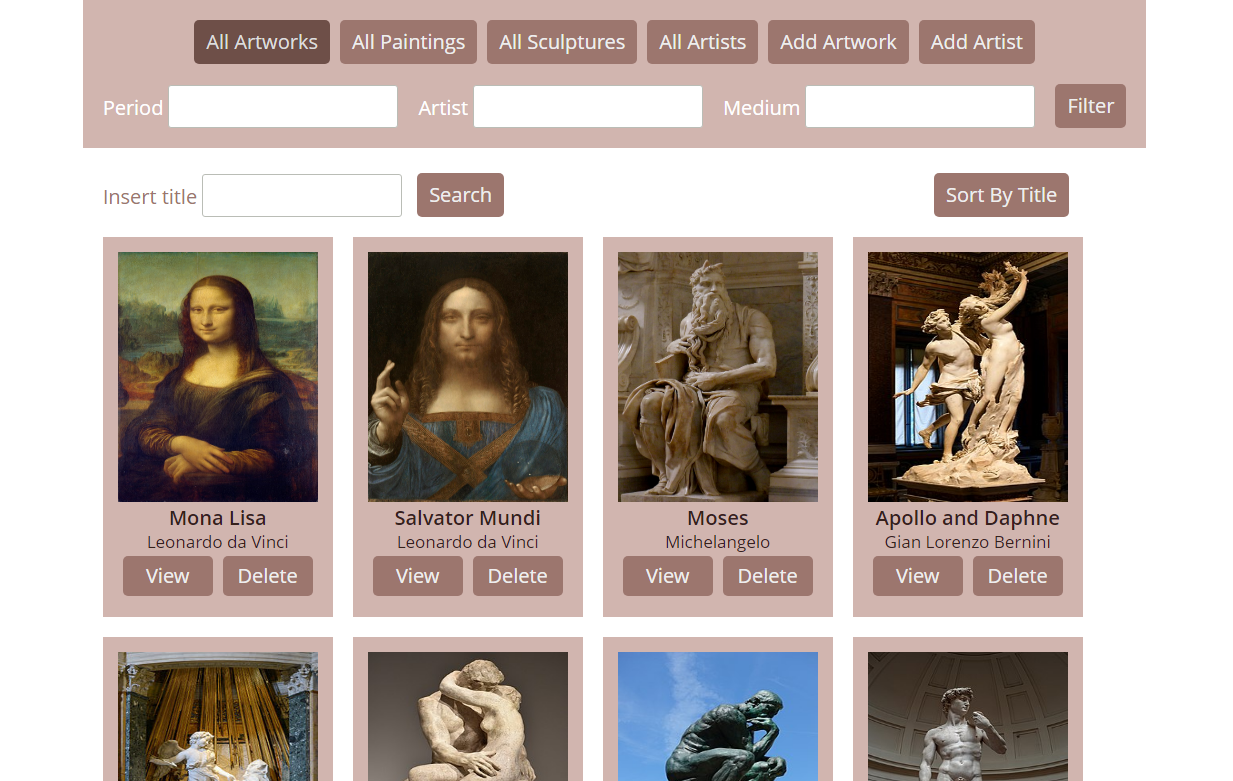
Dokumentacija za projekat iz funkcionalnog programiranja

Autor: Teodora Milanović

Opis aplikacije

Aplikacija omogućava prikaz umjetnika i umjetničkih djela. Korisniku je takođe omogućeno da djela isfiltrira na osnovu njihove forme klikom na dugme ‘All Paintings’ ili ‘All Sculptures’. Realizovan je i filter na osnovu umjetničkih perioda, umjetnika i materijala koji su korišteni za kreiranje tog umjetničkog djela. Postoji i mogućnost dodavanja novog umjetnika, te novog umjetničkog djela, kao i njihovog brisanja. Brisanje se vrši klikom na dugme ‘Delete’, dok se klikom na dugme ‘Add Artist’ i ‘Add Artwork’ prikazuje forma za dodavanje. Klikom na dugme ‘View’ nam se prikazuju dodatne informacije o tom umjetničkom djelu ili umjetniku. Omogućeno je i sortiranje i pretraga na osnovu naziva umjetničkog djela ili umjetnika.

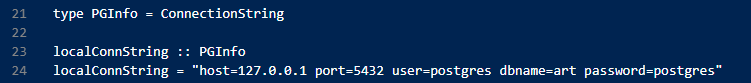


Slika 1 - Prikaz svih umjetničkih djela

Backend

Backend je rađen u Haskell-u. Korištena je PostgreSQL baza podataka, te biblioteka Persistent za povezivanje na bazu i realizaciju ORM-a. Za izradu API-ja je korištena biblioteka Servant, a za realizaciju složenijih upita biblioteka Esqueleto.

Da bismo uspostavili konekciju sa bazom potrebno je definisati potrebne podatke kao što su IP adresa host-a, broj porta, korisničko ime, naziv baze i šifra.

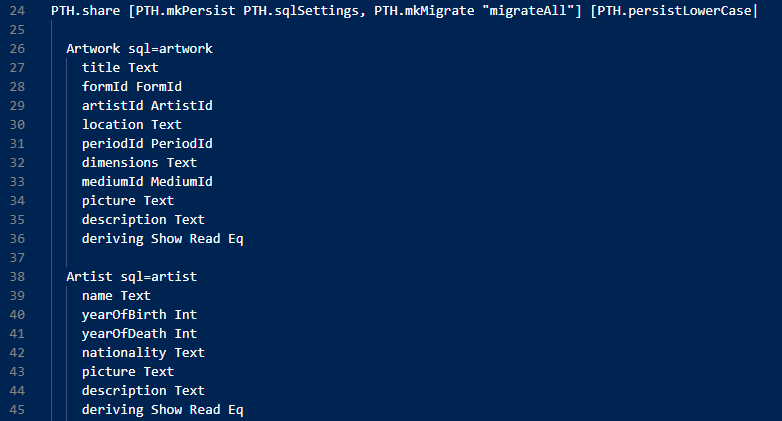


Slika 2 - Potrebni podaci za konekciju sa bazom

Za pokretanje serverske strane nam je bitna funkcija za pokretanje servera i funkcija za migraciju baze koja na osnovu sheme baze podataka, u našem slučaju fajla BasicSchema.hs, mijenja ciljnu bazu tj. bazu pod nazivom art. Zatim je iz komandne linije potrebno izvršiti sljedeće naredbe:

1. stack build
2. stack exec migrate-db
3. stack exec run-server

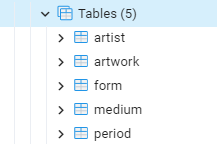
Kao što je spomenuto, za realizaciju ORM-a je korištena biblioteka Persistent, te su u fajlu BasicSchema.hs definisani modeli Artwork, Artist, Period, Form i Medium.





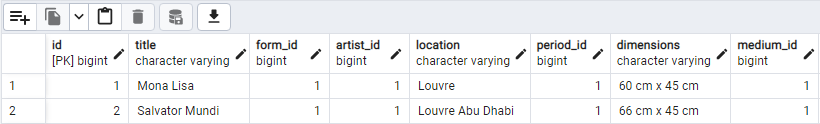
Slika 3 - Modeli

Slijedi da baza podataka sadrži pet tabela - artist, artwork, period, medium i form.



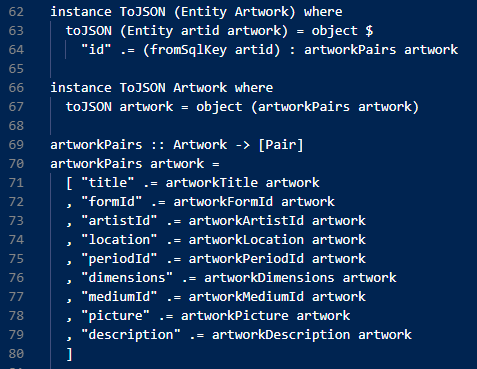
Slika 4 – Prikaz tabela u bazi

Svaka od tabela sadrži primarni ključ pod nazivom id kojim se jedinstveno identifikuju podaci u njoj. Id se generiše automatski i inkrementalno. Pored primarnih ključeva možemo uočiti i strane ključeve koji su prisutni u tabeli artwork. Svako umjetničko djelo pripada određenom umjetničkom periodu, umjetniku, formi, te je za njegovo kreiranje korišten određeni materijal. Samim tim, tabele period, form, artist i medium su povezane sa tabelom artwork putem njihovog jedinstvenog identifikatora.

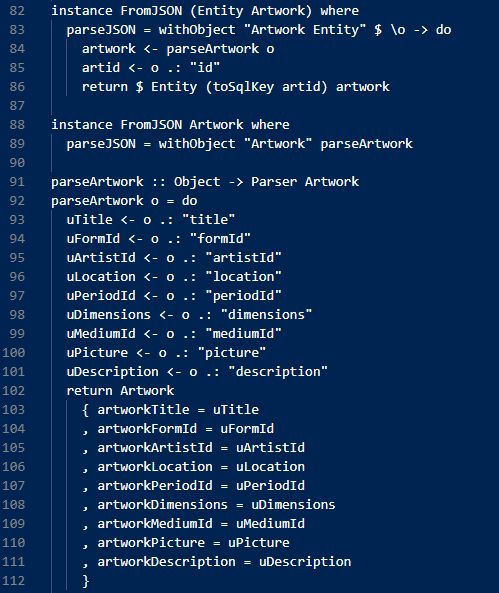


Slika 5 – Izgled tabele artwork u bazi

Podaci se između klijenta i servera prenose u obliku JSON-a, te je potrebno implementirati funkcije za parsiranje podataka u JSON format, te parsiranje podataka iz JSON formata u Haskell tipove. Za rad sa JSON podacima je korištena biblioteka Aeson.

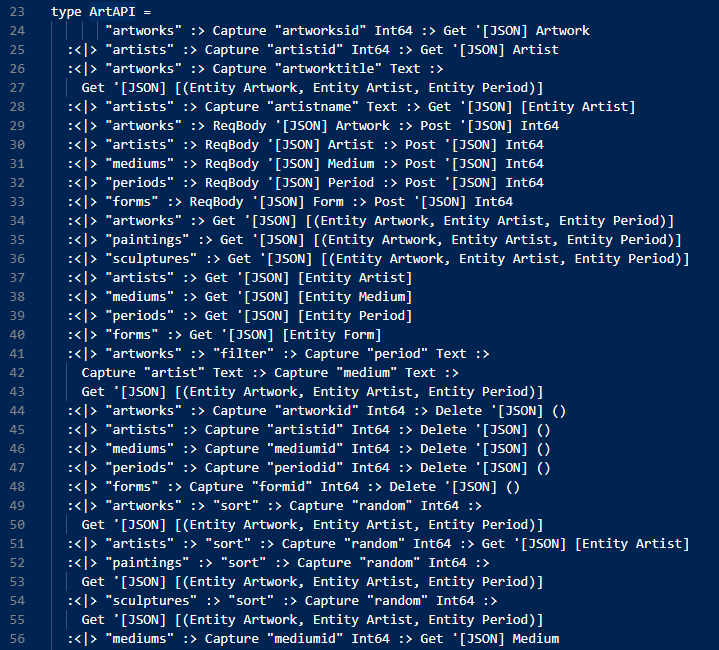


Slika 6 - Parsiranje u JSON format



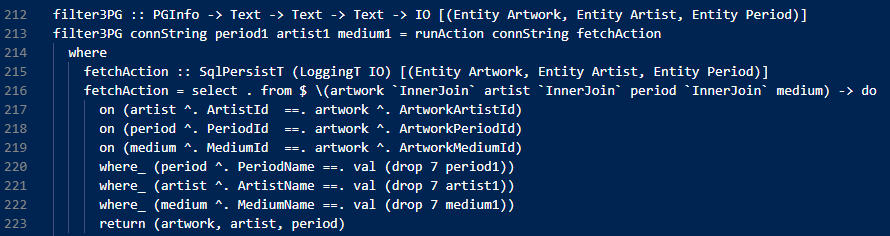
Slika 7 – Parsiranje iz JSON formata u Artwork

U fajlu BasicServer.hs su definisane sve krajnje tačke (eng. endpoints) tako što je definisan naziv krajnje tačke, HTTP metoda koja se koristi, te format u kom će se vraćati podaci, u ovom slučaju je to JSON. Takođe, u slučaju da se podaci šalju u tijelu zahtjeva, definiše se kombinator ReqBody praćen formatom poslatih podataka. U slučaju da se parametri šalju putem URL-a, potrebno je koristiti kombinator Capture praćen tipom podatka datog parametra. HTTP metode koje su korištene su GET za čitanje podataka iz baze, POST za upisivanje podataka u bazu i DELETE za brisanje podataka iz baze. Svakoj od krajnjih tačaka pridružujemo funkciju koja se poziva nakon što se datoj tački pristupi sa klijentske strane.



Slika 8 – Definisanje krajnjih tačaka

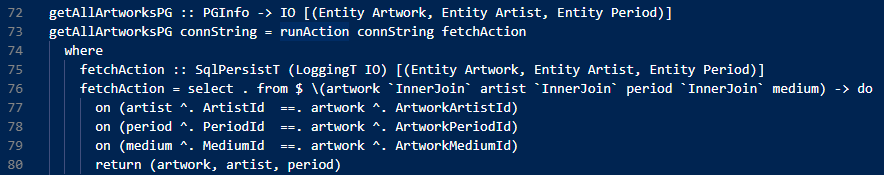
U fajlu Database.hs se nalaze funkcije koje omogućavaju rad sa podacima u bazi podataka. Za realizaciju JOIN upita korištena je biblioteka Esqueleto. U nastavku se nalazi primjer korištenja ove biblioteke.



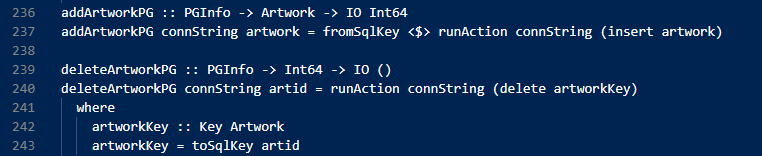
Slika 9 – Primjer višestrukog JOIN upita

Data funkcija služi za filtriranje umjetničkih djela na osnovu perioda, umjetnika i materijala koji je korišten za izradu. Korištenjem višestrukog JOIN upita omogućili smo povezivanje tabele artwork sa tabelama period, medium i artist pomoću stranih ključeva. Filtriranje se vrši uz pomoć višestrukih WHERE klauzula kojima se iz povratnog rezultata izbacuju sva umjetnička djela čiji naziv perioda, naziv umjetnika i naziv materijala za izradu nije jednak vrijednostima proslijeđenim u URL-u.

U nastavku se nalaze primjeri funkcija za dohvatanje, dodavanje i brisanje.



Slika 10 – Funkcija za dohvatanje svih umjetničkih djela iz baze podataka



Slika 11 – Funkcije za dodavanje i brisanje umjetničkog djela

Frontend

Frontend je rađen u Elm-u, te je aplikacija realizovana kao SPA iliti Single-page application. Potrebno je instalirati Elm kao i Node.js jer Elm kompajler transformiše .elm fajlove u JavaScript.

Naredbe za instaliranje potrebnih paketa u komandnoj liniji:

* elm install elm/http
* elm install elm/json
* elm install elm/svg
* elm install mdgriffith/elm-ui
* elm install TSFoster/elm-tuple-extra

Naredba za instaliranje paketa za formatiranje koda i sinhronizaciju prikaza i koda:

* npm install –g elm-test elm-format elm-review elm-live

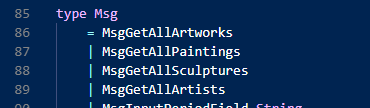
Naredba za pokretanje klijenta na portu 8000:

* npx elm-live src/Main.elm

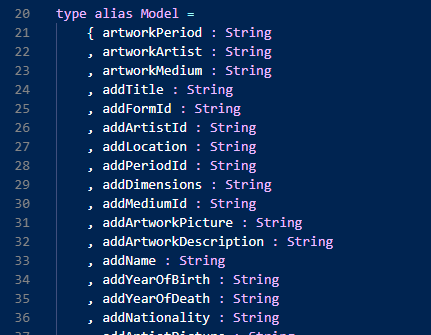
Elm arhitektura se svodi na tri dijela:

1. model
2. view
3. update

Stanje aplikacije se predstavlja strukturom podataka koju nazivamo model. Model sadrži bitne informacije o aplikaciji koje se dinamički mijenjaju, te ga je potrebno inicijalizovati. View funkcija kao argument uzima model, a kao rezultat vraća HTML kod, međutim ne renderuje ga na ekranu. Za to je zaslužan paket pod nazivom elm/virtual-dom koji se automatski instalira. Update funkcija omogućava aplikaciji da bude responzivna. S obzirom da update funkcija za argument uzima i akciju koja treba da se izvrši, potrebno je i definisati poruke (eng. messages) koje predstavljaju te akcije. To se radi deklaracijom novog tipa podataka, u ovom slučaju Msg.

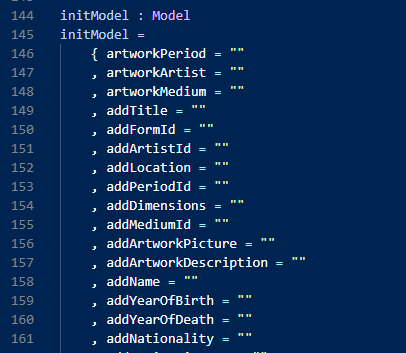


Slika 12 – Dio tipa Msg



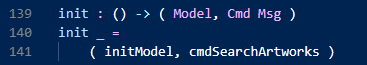
. . .

Slika 13 – Dio modela

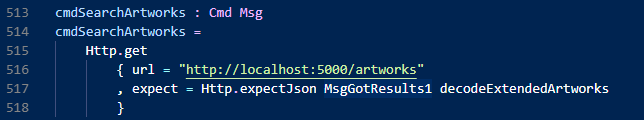


. . .

Slika 14 – Dio inicijalizacije modela

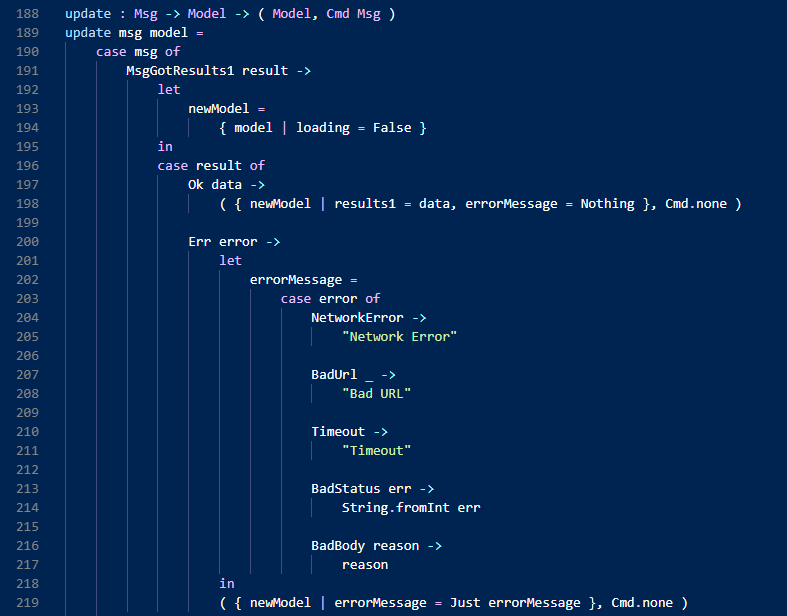


Slika 15 – Init funkcija



Slika 16 – Funkcija za pristup krajnjoj tački /artworks

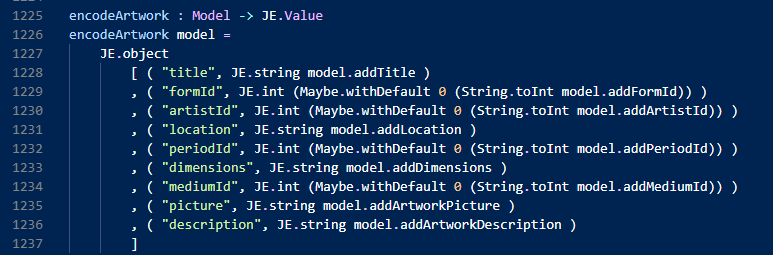
Init funkcija vrši inicijalizaciju aplikacije tako što inicijalizuje model, a zatim pozove funkciju cmdSearchArtworks koja služi za dohvatanje umjetničkih djela iz baze podataka pomoću HTTP metode GET. Na ovaj način mi omogućavamo prikaz svih umjetničkih djela odmah po učitavanju klijenta.



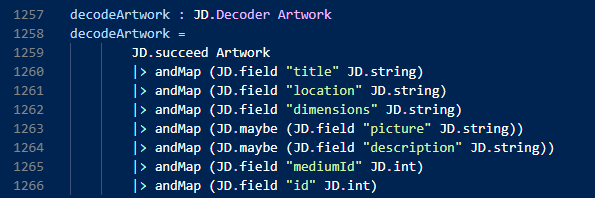
Slika 17 – Dio update funkcije

Nakon što dobijemo podatke sa serverske strane, podaci se pozivom update funkcije smještaju u promjenljivu koja služi za skladištenje umjetničkih djela, u ovom slučaju results1. Samim tim dolazi i do ažuriranja modela, a potom i HTML koda. U slučaju da dođe do određene greške, ista će se prikazati na ekranu.

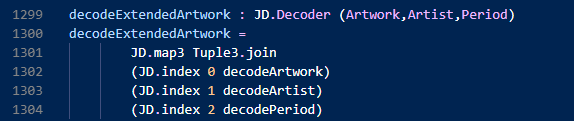
Da bismo uspješno izvršili komunikaciju sa serverom potrebno je i da omogućimo parsiranje podataka u JSON format, te parsiranje podataka iz JSON formata u Elm tipove. Za to se koristi elm/json paket, tačnije Json.Encode i Json.Decode.



Slika 18 – Parsiranje podataka u JSON format



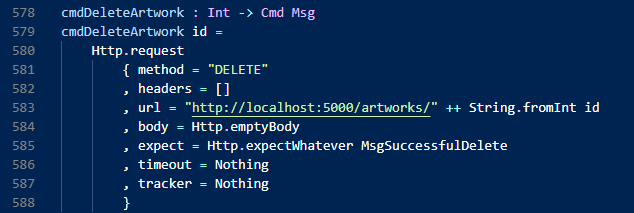
Slika 19 – Parsiranje podataka iz JSON format-a u tip Artwork



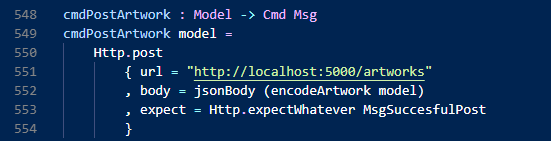
Slika 20 – Parsiranje trojke

S obzirom da su sa serverske strane podaci o umjetničkim djelima stizali u vidu trojki (umjetničko djelo, umjetnik i period), potrebno je svaki od člana trojke zasebno parsirati u odgovarajući Elm tip kao što možemo vidjeti na slici 20.

Pored HTTP metode GET (Slika 16), za komunikaciju sa serverom su korištene i metode DELETE i POST. U slučaju POST metode u tijelu funkcije se šalje podatak koji je parsiran u JSON format. U nastavku se nalaze primjeri ovih zahtjeva.



Slika 21 – Zahtjev za brisanje umjetničkog djela



Slika 22 – Zahtjev za dodavanje umjetničkog djela

Literatura

<https://package.elm-lang.org/>

<https://mmhaskell.com/elm>

<https://mmhaskell.com/real-world>

<https://elmprogramming.com/>