Documentație tehnică pentru RoLocații

I. General

1. Informații generale

Ați pățit vreodată sa mergeți prin România în vacanță și să mancați la restaurantul pe care nici acum nu îl puteți uita? Ei bine, nu l-ați uitat. Dar probabil i-ați uitat numele. Sau locația exactă. Sau probabil chiar și numele localității în care se afla?

Ei bine, RoLocații rezolvă această problemă. Și nu numai pentru restaurante! Toate locațiile din România vă vor fi salvate într-un singur loc: în contul dumneavoastră gratuit de pe RoLocații. Tot acolo veți putea găsi și locația perfectă pentru următoarea dată când nu veți ști unde să ieșiți în oraș. Iar asta datorită recenziilor plăcute lăsate pe Google de către persoanele ce au vizitat locația. Cum se întampla magia aceasta? Totul se bazează pe Google Places API.

2. Tehnologii folosite

Pe front-end am folosit framework-ul/librăria ReactJS. Mai jos găsiți o listă cu toate dependency-urile NPM folosite, așa cum apar in *package.json*.

```
"dependencies": {
    "@fortawesome/fontawesome-svg-core": "^1.2.28",
    "@fortawesome/free-brands-svg-icons": "^5.13.0",
    "@fortawesome/free-solid-svg-icons": "^5.13.0",
    "@fortawesome/react-fontawesome": "^0.1.9",
    "axios": "^0.19.2",
    "email-validator": "^2.0.4",
    "google-maps-react": "^2.0.6",
    "react": "^16.13.1",
    "react-dom": "^16.13.1",
    "react-router": "^5.1.2",
    "react-router-dom": "^5.1.2",
    "react-scripts": "3.4.1"
  }
```

Pe back-end am folosit Node.js cu framework-ul Express, MongoDB servindu-mi drept bază de date. Mai jos găsiți o listă cu toate dependency-urile NPM folosite, așa cum apar in *package.json*.

```
"dependencies": {
    "axios": "^0.19.2",
    "bcrypt": "^4.0.1",
    "cors": "^2.8.5",
    "dotenv": "^8.2.0",
    "email-validator": "^2.0.4",
    "express": "^4.17.1",
    "jsonwebtoken": "^8.5.1",
    "mongoose": "^5.9.11",
    "morgan": "^1.10.0",
    "nodemon": "^2.0.3"
  }
```

Cele două se îmbină facănd API calls de pe front-end la un REST API de pe back.

3. Google Places API

Aplicația mea se folosește de un API pus la dispoziție de Google pentru a funcționa. Inițial doream să mi se afișeze absolut toate locațiile din județul ales, însa curând am aflat ca acest lucru este imposibil folosind API-ul Google. Asta pentru că rezultatele sunt limitate la 60 de locații/request. Atunci, m-am limitat la a găsi numai locații din reședințele de județ. Pe viitor sper să găsesc o metodă de a rezolva asta.

Cam așa funcționează API-ul: se face un request la un URL afișat la ei in documentație și se introduc in parametrii URL-ului mai multe detalii, pentru a obține response-ul dorit. Din păcate nu am avut opțiunea de a seta să caute locații într-un județ sau un oraș întreg, așa ca am abordat o altă rezolvare. Am introdus în parametrii coordonatele reședinței de județ (coordonate pe care le-am scris îintr-un fișier JSON cu detalii de pe Wikipedia) pentru care se face request și va căuta locațiile într-o rază de 50km din acel punct (50km fiind numărul maxim ce se poate alege pentru parametrul razei). Astfel, evident că atunci cand căutam locații din Arad primeam si locații din Timișoara și invers. Așa că, după request mereu filtram response-ul pentru a “scăpa” de locațiile nedorite, si pentru a le afișa numai pe acelea care au în adresă numele reședinței de județ. La început spuneam că Google limitează response-urile la 60 de locații/request, așa ca acele 60 de locații, filtrate pe deasupra, vor ajunge mult mai puține. Rezultat? Nu toate locațiile se vor

afișa. Aceasta este problema cu care m-am confruntat mult timp, și încă nu i-am găsit o soluție. Pe viitor, cel mai probabil voi schimba API-ul. Aplicația mi se pare o idee bună, aducând Google Maps, Booking si multe altele la un singur loc. Păcat insă că am dat de asemenea limitări. Request-urile se fac de pe back-end în urma unui request de pe front-end. Asta pentru a se filtra locațiile direct din back-end și a trimite un response mai curat, cât si pentru a evita erorile legate de “CORS policy” întâlnite.

II. Front-end

1. General

Aplicația web RoLocații are la bază librăria React pe front-end. Asta îmi oferă avantajul de a avea component reutilizabile, pentru un cod curat. Cel mai simplu exemplu ar fi lista cu locațiile. Fiecare locație arată la fel, insă textul diferă. React de asemenea m-a ajutat să creez o SAP (single-page application), avand interacțiuni si aspecte moderne, cu conținut schimbându-se dinamic (folosindu-mă de React Router).

2. Design

În realizarea aplicației am avut în minte un design modern, simplu și ușor de folosit. Arată la fel de bine pe desktop, cât și pe mobil sau tabletă. Foarte intuitiv, oricine s-ar putea descurca în a-și găsi hotelul ideal sau orice altă locație.

3. Cum funcționează harta?

Harta României de pe prima pagină reprezintă “unealta” întregii aplicații ca aceasta să funcționeze. Harta e de fapt un SVG încorporat in HTML. SVG-ul l-am preluat de pe internet (https://simplemaps.com/resources/svg-ro) și l-am modificat, personalizat aplicației mele, astfel că la trecerea cu mouse-ul pe deasupra, fiecare județ în parte se va colora. Text am adăugat pe SVG folosind https://editor.method.ac/. Fiecărui județ (SVG path) în parte i-am aplicat atributul “onclick” (“onClick” in varianta React), astfel că, în momentul click-ului, s-a făcut request spre back-end cu informațiile județului, pentru a primi înapoi un response cu toate locațiile. Fiecare SVG path a venit însoțit de un ID unic. Am scris un fișier JSON cu toate județele si coordonatele lor (a durat ceva timp), alaturi de ID-ul unic. La click s-a făcut corespondența între numele județului și ID, luându-se coordonatele pentru request. Exemplu:


```
{
    "coord": "46.10,21.18",
    "id": "ROU276",
```

```
    "nume": "Arad",
    "resedinta": "Arad",
    "simplu": "Arad"
  },
  {
    "coord": "44.51,24.52",
    "id": "ROU302",
    "nume": "Arges",
    "resedinta": "Pitești",
    "simplu": "Pitesti"
  }
```

De asemenea, în request intră și tipul de locații dorite. Lista cu tipurile valabile am găsit-o aici https://developers.google.com/places/web-service/supported_types. Sunt trecute într-un alt fișier JSON ce este citit pe front-end.

```
{
    "value": "hotel",
    "ro": "Hoteluri"
  },
  {
    "value": "school",
    "ro": "Scoli"
  },
  {
    "value": "hospital",
    "ro": "Spitale"
  }
```

4. Securitate

Modul în care am programat front-end-ul nu lasă utilizatorii să vadă ce nu ar trebui sau, și mai rău, sa aplice atacuri/metode de hacking. Validarea datelor introduse de utilizator are loc atât pe front-end, cât și pe back-end. Pentru a proteja interfața de atacurile XSS, am decis a stoca JWT-ul pentru autentificare in cookie-uri și nu în localStorage.

III. Back-end

1. General

Partea de back-end a site-ului a fost construită folosind Node.js împreuna cu Express pentru a forma împreuna un REST API, cu transfer de date prin request-uri HTTP. Baza de date folosită este baza de date No-SQL, MongoDB, funcționând pe principul documentelor BSON (typed JSON) stocate in colecții. Aplicația folosește arhitectura MVC (mai mult sau mai puțin), “view-ul” aflându-se pe front-end, cu React. Drept abstraction layer pentru MongoDB am folosit Mongoose.

2. Routing

Pentru fiecare componentă majoră a aplicatiei, am aplicat câte o rută specifică. Toate rutele însă apar în urma URL-ului “https://domeniu.com/api/v1", astfel crescând scalabilitatea aplicației când va urma un potențial update major (v2). Am folosit rute pentru autentificare (“/auth/register” de exemplu), pentru locații (“/places”) și pentru a adăuga locații în lista de favorite (“/favourite”).

3. Validare de date, autentificare, securitate și eficiență

Fiecare request ce vine la server este verificat dacă este corect și dacă conține ceea ce e nevoie pentru a prelucra request-ul. Pentru autentificare am scris niște funcții middleware de validare a datelor de input ale utilizatorilor. De exemplu, parola sa conțină minim 6 caractere și să fie egală cu cea din câmpul confirmării parolei. Se verifică, de asemenea, și dacă email-ul introdus este unul valid, respectand structura unui email. Pentru asta am folosit un pachet de pe NPM care se poate gasi într-una din paginile de mai sus.

Autentificarea se bazează pe principiul JSON Web Token, token-ele fiind generate pe server, trimise la client și primite înapoi, pentru a accesa rutele private, printr-un header. Acolo în header intră Bearer token-ul JWT. Verificarea se face printr-o funcție middleware. De asemenea, criptarea parolei pentru transferul în baza de date se face folosind puternicul algoritm one-way, Bcrypt.

Din punct de vedere al securității, pe lânga metodele de validare a datelor, variabilele și datele “sensibile” sunt stocate în afara fișierelor .js, într-un fisier .env, unde se găsesc toate aceste “environment variables”. Asta, bineînțeles, in timpul development-ului. În production, variabilele sunt introduse manual în meniul serviciului de deployment/hosting Heroku. Printe aceste variabile se numară si

NODE_ENV, parola, string-ul de accesare a bazei de date și string-ul secret pentru generarea JWT-urilor.

Pe lânga toate aceste metode de securizare, am mai abordat un concept care nu e bun numai pentru securitate, ci și pentru scalabilitate si un development mai ușor. Variabila NODE_ENV de mai sus reprezintă mediul în care ruleaza aplicația. Production sau development. Dacă se află în development și cumva, vreun dependency sau vreo funcție exprimă un “uncaught exception” sau un “unhandled promise rejection”, aplicația se va inchide automat si restarta, pentru a împiedica utilizatorul de a continua a utiliza aplicația într-un mod eronat. De asemenea, în funcție de mediul în care ruleaza aplicația, în response, în caz de eroare, în development se vor afișa toate detaliile erorii, în mod amănunțit, pe când în caz de production se va afișa doar un mesaj prietenos de tipul “Eroare de server”.

Nu se poate pune problema atacurilor SQL Injection întrucât MongoDB nu se folosește de SQL pentru a funcționa.

Pentru eficiență am abordat programarea asincronă, bazată pe “promisiuni JavaScript”, cu scopul de a mări numărul task-urilor pe care le poate executa singurul thread Node.js simultan, cât am abordat și metode de a scrie cod cât mai curat si scurt posibil.

IV. Implementări viitoare

Următorul lucru pe care îl voi implementa va fi opțiunea de a putea căuta după cuvinte un set de locații din API-ul Google. Bara de căutare momentan apare acolo numai cu scop estetic. O alta implementare dorită de mine va fi opțiunea de a lăsa review-uri în cadrul RoLocații, nu doar în cadrul Google. Se poate observa asta în modelul Mongoose al user-ului. Nu ar fi fost problemă sa le implementez pentru competiție, dacă aflam mai devreme de “Ediția online” ca să apuc sa pregătesc aplicația în mai mult de o săptamână.

Mulțumesc pentru atenție!