## Изработка на веб-апликација за споделување на патувања - Carpooling app

Изработил: Кристијан Ристовски, индекс: 196062

### Вовед

Апликациите од овој тип се ретки во нашата држава, но би биле од огромна корист на граѓаните имајќи ја во предвид состојбата на железниците, автобусите и цената за гориво при приватни патувања. Користејќи го ова како инспирација и успешноста на сличен ваков странски продукт, [Blablacar](https://blablacar.com/), како финален проект по предметот Напреден веб дизајн, решив да изработам едноставна апликација која би го решила проблемот со патувањата во нашата држава и со помош на оваа апликација да го покажам стекнатото знаење во полето на frontend веб-дизајн.

### Користени технологии и структура на проектот

Проектот, како скоро секоја модерна веб-апликација, е поделен во два дела, frontend и backend.

Backend делот е на проектот е изработен во **dotNet 8** и е доста едноставен и пригоден на frontend делот за да се прикажат саканите податоци. Како база за податоците ќе користиме локална инстанца на **Microsoft SQL Server** која моделот за табелите ќе го добива директно од backend.

Frontend делот е изработен во Vue 3 и скелетот на целиот проект е изработен со помош на Vite. Како решение за менаџирање со состојба (**state management**) наместо Vuex, во овој проект ќе се користи **Pinia**, како помодерно, побрзо и поедноставено решение, без мутации кои многу го комплицираат целиот проект.

Во следниот дел детално ќе ги анализираме и frontend и backend делот.

### Backend - анализа и опис

Пред да зборуваме за главниот дел од backend делот, треба да анализираме како овој дел од апликацијата се поврзува со база во која ќе ги чуваме нашите податоци. Најпрво, бидејќи работиме во локална околина, на машината мора да се инсталира и покрене инстанца од **Microsoft SQL Server** и истата да се конфигурира. По конфигурацијата, во проектот мора да специфицираме еден **ConnectionString**, прикажан на **Слика 1**, кој му кажува на апликацијата да се поврзе на **localhost\\SQLEXPRESS,** на база **carpooldb** со дополнителни параметри кои овозможуваат правилно поврзување.

|  |
| --- |
|  |
| **Слика 1.** Специфицирање на Connection string |

Следно, ќе го разгледаме моделот со кој опишуваме едно патување. Поради едноставноста на целата апликација, имаме само еден модел кој ја покрива целата сакана функционалност. Комплексноста на моделот може драстично да се зголеми доколку сакаме да имплементираме корисници, посебни модели за возила, дополнителни проверки и пресметки на податоците. На **Слика 2** можеме да ја видиме структурата на моделот и изгледот на моделот во база.

|  |
| --- |
|  |
| **Слика 2.** Модел во код и во база |

По дефинирањето на моделот, мораме да го додадеме во податочниот контекст на апликацијата за таа да може да креира соодветно пресликување од моделот специфициран во код во модел читлив за база на податоци. Додавањето на моделот во податочниот контекст се извршува со кодот прикажан на **Слика 3.**

|  |
| --- |
|  |
| **Слика 3.** Додавање на модел во податочен контекст |

Наредно, ќе го разгледуваме функционалниот дел од backend на апликацијата, односно делот кој нѝ овозможува да земаме и манипулираме податоци од базата. Со контролерите можеме да дефинираме методи кои би овозможиле комуникација со frontend делот и би дозволиле истиот да манипулира со податоците со испраќање на соодветни **requests**. Најпрво го дефинираме контролерот и специфицираме на кој **endpoint** може да се пристапи до него, во овој случај е името на контролерот или endpoint-от ќе биде **/Rides**. Следно го додаваме податочниот контекст во конструкторот на самиот контролер и со тоа ќе пристапуваме до податоците кои ги имаме во базата. Сето ова може да се види на **Слика 4**.

|  |
| --- |
| A screen shot of a computer program  Description automatically generated |
| **Слика 4**. Дефинирање на контролер |

По дефинирањето на контролерот, ќе го разгледуваме **GET** методот прикажан на **Слика 5**. Со овој метод, интуитивно, ги земаме сите досега креирани патувања од базата на податоци, без користење на никаков филтер, односно ќе ги земеме апсолутно сите записи и ќе ги ставиме во листа.

|  |
| --- |
| A screenshot of a computer code  Description automatically generated |
| **Слика 5**. GET метод |

Следно, го разгледуваме **POST** методот за додавање на ново патување, прикажан на **Слика 6**. За овој метод, го изложуваме **/Rides/add** endpoint-от за преку овој endpoint да праќаме податоци од frontend за да ги додадеме во база со помош на дефинираниот метод. По добивањето на податоците за патувањето кое сакаме да го креираме, методот генерира и време на креирање и го додава на патувањето за да имаме преглед кога тоа патување било креирано во база и по успешно додавање во базата, враќаме објект кој го содржи креираното патување.

|  |
| --- |
| A computer screen shot of text  Description automatically generated |
| **Слика 6.** ADD метод |

Наредно ги ќе ги разгледаме **PUT** методите. **PUT** најчесто се користат за ажурирање на веќе постоечки записи во база, па затоа, во оваа апликација, овој метод ќе служи за ажурирање на бројот на достапни места за седење по патување. Имаме **BookSeat** метод кој го користиме кога сакаме да резервираме место. Како параметар во овој метод го земаме идентификациониот број на патувањето и пробуваме да го најдиме. Доколку не постои враќаме грешка дека не е пронајден бараниот запис. Доколку постои го проверуваме бројот на достапни седишта и доколку е нула, фрламе грешка, а во други случаи го намалуваме бројот, со што кажуваме дека седиштето е резервирано. Аналогно на тоа, откажувањето на седиште работи скоро идентично, освен во проверката на достапните седишта. Тука, доколку достапните седишта се еднакви со бројот на седишта во возилото, тоа значи дека сите места се слободни и нема да се дозволи инкрементација на бројот на седишта, односно дополнително ослободување. Двата медоти можат да се видат на **Слика 7** и **Слика 8** соодветно.

|  |  |
| --- | --- |
| A screen shot of a computer program  Description automatically generated | A screen shot of a computer program  Description automatically generated |
| **Слика 7**. BookSeat метод | **Слика 8**. CancelSeat метод |

Последно, во овој контролер ја имаме **DELETE** функционалноста. Со методот **DeleteRide**, како параметар земаме идентификационен број на саканото патување и го бришеме од базата на податоци, доколку тоа постои. Методот може да го видеме на **Слика 9**.

|  |
| --- |
| A screen shot of a computer program  Description automatically generated |
| **Слика 9**. Метод за бришење |

|  |
| --- |
| **A screen shot of a computer program  Description automatically generated** |
| **Слика 10**. Помошен контролер за локации |

Дополнително во програмата имаме дефинирано и помошен контролер кој служи за добивање на опциите потребни во паѓачките менија на некои полиња во frontend делот. Контролерот, прикажан на **Слика 10,** го изложува **/api** endpoint преку кој ќе пристапиме до саканите методи во овој контролер. Исто како и претходно, го додаваме податочниот контекст во инстанцата. Дефинираме **GET** метода на **/api/locations** endpoint со која ги земаме сите почетни и крајни локации и уникатните вредности ги ставаме во една листа, за да добиеме комплетна резултантна листа од сите локации во сите поединечни патувања во што ги имаме во базата на податоци.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |
| **Слика 11**. Изложени endpoints |

Со тоа го опфативме целиот Backend дел од апликацијата, и на **Слика 11** можеме да ги видиме сите изложени endpoints од соодветните контролери од **Swagger** документацијата.

### Frontend - анализа и опис

Frontend делот е далеку по комплексен и ектензивен од претходно опишаниот backend. Големиот дел од бизнис логиката, филтрирања, сортирања и слично, се одвиваат во frontend. Во пракса, ова не се практикува бидејќи за големи проекти, големиот број на податоци што треба да се преработат, најчесто го кочат целиот веб-пребарувач и затоа сета таа потешка логика е префрлена на backend. Но, за овој проект, малиот број на податоци и фокусот на предметот на frontend технологиите, целата логика е префрлена во овој дел. Дополнително, за изгледот на апликацијата, користена е библиотеката **Vuetify** која содржи веќе стилизирани компоненти.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |
| **Слика 12.** Структура на frontend проект |

На **Слика 12** е претставена структурата на проектот и по неа ќе ја вршиме анализата на кодот. Најпрво ќе зборуваме за помошните функции во **services** датотеката и **state management** делот во **store** датотеката.

### Помошни фунцкии - Services

Во сервисите имаме само функции кои комуницираат со backend на апликацијата. Тие функции се сместени во **api** папка во соодветни **JavaScript** датотеки. Во **rides.js** се наоѓаат помошни функции за повикување на **endpoints** од **Rides** контролерот, а во **utils.js** се наоѓаат повиците до **Utils** контролерот. Како што може да се види на сликите 13 и 14 подолу, ја користиме **axios** библиотеката за реализирање на повиците до backend до соодветните endpoints.

|  |  |
| --- | --- |
|  |  |
| **Слика 13.** Содржина на rides.js | **Слика 14.** Содржина на utils.js |
|  |  |

### State management - Pinia stores

|  |
| --- |
| A screen shot of a computer program  Description automatically generated |
| **Слика 14**. Содржина на utilsStore.js |

Во папката **stores**, ги имаме двете складишта на нашата апликација, едната за патувањата, **rideStore.js** и едната за помошните податоци, **utilsStore.js**.

Во **utilsStore.js** ја чуваме состојбата на локациите и сакаме да персистира низ целата апликација. Правиме повик до backend да ги добиеме податоците и ги чуваме во глобално достапна променлива за состојба, па кога и да имаме потреба од пристап до овие податоци, од било која компонента, можеме да пристапиме без повторно да праќаме повик до backend.

Во **rideStore.js** ја чуваме состојбата на патувањата и дополнително правиме сортирање и филтрирање. Во **state** делот ги чуваме параметрите за сортирање, интервалот на максимална и минимална цена на патувањата, иницијалните патувања без никаква манипулација и иницијалниот интервал на максимална и минимална цена и **rides** листата која всушност е конечната листа по сите пресметки. Оваа листа ќе се користи низ апликацијата. Во **actions** делот ги правиме сите пресметки. Методот **getAllRides** го користиме за да ги земиме податоците од backend со користење на функцијата од **api.** Резултатот го ставаме во **initialRides** и овој резултат го користиме за да се навратиме доколку има потреба на иницијалната листа на патувања, а со помош на оваа листа го пресметуваме иницијалниот интервал на цени од патувањата. Правиме плитка копија на иницијалните патувања во **rides** променливата и продолжуваме со сортирање на истата со поставените параметри за сортирање. Имаме и две функции за ажурирање на параметрите за сортирање и интервалот на цени. Методот за сортирање **sort** ги подредува елементите во листата во зависност од тоа што имаме како параметар за сортирање, а **rangeFilter** методот проверува кој елемент од листата е во интервалот на цени кој го имаме моментално.

|  |  |
| --- | --- |
| A screen shot of a computer  Description automatically generated | A computer screen shot of a program code  Description automatically generated |
|  | |
| **Слика 15**. Содржина на rideStore.js | |

### Компоненти - изглед и функционалност

### App.vue

Оваа компонента е главната компонента во која се сместени сите останати компоненти, односно оваа компонента е родител на сите понатаму опишани компоненти. Во **template** делот од компонентата е сместен делот од апликацијата што се покажува на екран, односно HTML, други компоненти и слично. Како што може да се види на **Слика 16** имаме повеќе компоненти кои ќе ги опишуваме поединечно. Во оваа компонента имаме логика за земање на податоците од **store**, прикажување на патувањата, прикажување на компонента за **loading** додека се земаат истите податоци, логика за филтрирање на податоците, односно во оваа компонента се специфицираат параметрите со кои се филтрира во позадина.

|  |
| --- |
| A screenshot of a computer  Description automatically generated |
| **Слика 16**. Изглед на апликацијата (App.vue) |

### RidePosting.vue

Компонентата со која ги прикажуваме патувањата е претставена на **Слика 17**, неа ја рендерираме во **App.vue** со извадокот прикажан на **Слика 18**. Како што можи да се види, на компонентата како **prop** му го праќаме **ride** објектот кој ги содржи сите податоци за тоа патување и компонентата соодветно се справува и ги прикажува тие информации. Слободните седишта се визуелизирани со помош на икони кои слобоните места ги покажуваат со зелена боја, а зафатените како обезбоени икони. Имаме и копчиња за резервирање и откажување место, како и копче за бришење на патување. Сите копчиња ги користат соодветните методи за повикување на методите од backend.

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated | A screen shot of a computer code  Description automatically generated |
| **Слика 17**. Изглед на RidePosting.vue | **Слика 18**. Извадок од App.vue за рендерирање на компонентата |

### SortByComponent.vue

Со оваа компонента ги поставуваме параметрите за сортирање на листата на патувања. По секоја промена на параметрите, компонентата прави соодветен повик до функцијата за ажурирање на параметрите во **rideStore** и понатамошната функционалност му ја остава неа. Повиците до функциите се одвиваат со помош на **watch**, односно Vue ја следи состојбата на одреден параметар во компонентата и доколку се промени, ивршува некоја функција, зададена во телото на watcher-от. Целта на оваа компонента е само тоа, одбирањето на параметрите за сортирање.

|  |
| --- |
|  |
| **Слика 19.** Изглед на SortByComponent.vue |

### RangeFilterComponent.vue

Оваа компонента користи дво-краен лизгач за да го поставиме интервалот на цени со кој сакаме да ги филтрираме патувањата. Во оваа компонента како **props** од главната компонента ги праќаме иницијалните вредности на минимумот и максимумот и ги чуваме за да го ресетираме лизгачот кога посакаме. Секое понатамошно менување на позицијата на лизгачот, покренува метод за ажурирање на параметрите во **ridesStore** за да ги изфилтрира патувањата кои не се во бараниот интервал.

|  |
| --- |
| A screenshot of a video  Description automatically generated |
| **Слика 20**. Изглед на RangeFilterComponent.vue |

### SearchRides.vue

Една од покомплицираните компоненти во овој проект, **SearchRides** ја користиме за пребарување на патувањата по одредени параметри. Првите две полиња за одбирање на почетна и крајна локација, имаат **autocomplete** функционалност која пребарува по листа од локации, а може и да се одбери од паѓачко мени со истите локации низ кои пребарува. Исто така има функционалност за промена на вредностите од едно во друго поле, прикажано подолу на **Слика 22**. Полето за внесување датум прикажува календар во кој може да се одбере некоја дата во **mm/dd/YYYY** формат. Форматот не може да се промени поради тоа што компонентата е сеуште некомплетна и во фаза на тестирање од страна на Vuetify, но ова не прави голем проблем низ апликацијата бидејќи сепак овој датум се конвертира во **JavaScript Date** објект кој има стандарден изглед. Во компонентата без да кликнеме на копче за пребарување, нашите параметри ќе се проследат низ апликацијата. За таа цел користиме **watcher** за да препознае промени во полињата и доколку има некаква промена, повикува **debounce** метод кој всушност претставува функција која се извршува по одредено време. Оваа функционалност ја добиваме од библиотеката **lodash.**

|  |
| --- |
|  |
| **Слика 21**. Изглед на SearchRides.vue |

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated | A screenshot of a calendar  Description automatically generated |
| A black screen with arrows and a black background  Description automatically generated with medium confidence | A screenshot of a computer  Description automatically generated |
| **Слика 22**. Функционалности на SearchRides.vue | |

### NewRidePosting.vue

Како што ни кажува името на компонентата, оваа компонента се користи за додавање на ново патување со помош на popup форма. Со кликање на копчето претставено на **Слика 23,** ја активираме формата и излегува како popup како што е прикажано на **Слика 24.** Формата во неа има правила кои се доделуваат поединечно на секое поле и доколку едно од тие правила е неисполнето, формата нема да дозволи да се пратат податоците до backend. Но, доколку таа е валидна и сите податоци се во склад со поставените правила, во компонентата се повикува методот за праќање на ново патување до backend и истата се затвора, бришејќи ги досега внесените податоци во полињата.

|  |  |
| --- | --- |
| A black background with blue text  Description automatically generated | A screenshot of a computer  Description automatically generated |
| **Слика 23**. Копче со кое ја уклучуваме popup формата | **Слика 24**. Изглед на popup формата |

|  |
| --- |
| A screenshot of a computer  Description automatically generated |
| **Слика 25**. Изглед на формата доколку не е валидна |

### Можности за скалирање и дополнителни функционалности

Како и сите модерни веб-апликации и оваа апликација има огромен потенцијал за скалирање. Покрај тоа што може да се подобри во изглед и да се подобри корисничкото искуство, туку може и да се додадат огромен број на нови функционалности, како најава на корисници, оценување на возачи, дополнителни валидации на формите, поробустен backend, миграција на голем дел од логиката во backend итн. Една функционалност која беше иницијална идеја да се воведе и во овој проект беше користењето на екстерно API, пример Google Maps API, за да се добие листа од градови и места кои потоа ќе може да се изберат како опции и со гео локација да се пресмета времето за патување и слично. Но тоа остана само идеја поради тоа што скоро сите API се со ограничување и клучните функционалности се недостапни без плаќање релативно големи суми, па затоа таа идеја остана нереализирана.