

Carpooling web application
-project documentation-

Contents

1	Brief project description	4
2	Software development process	5
2.1	Project plan	5
2.2	Milestones.....	5
2.3	Detaljan plan projekta.....	8
2.4	Description of algorithms	13
2.5	Structure of software modules.....	18
2.6	Use case diagram.....	29
2.7	Sequence diagram.....	31
2.9	Component Diagram	32
2.10	Planning diagram.....	33
2.11	Database model	34
3	Software testing	37
3.1	Unity testing.....	37
3.2	Integrtrion testing	41
3.3	Graphics depending on the number of errors in the software by week	42
4	Software delivery	43
4.1	Literature.....	48

1 Brief project description

The theme of the project is the web application for car pooling. The topic is very active and corresponds to a very important area of our lives, as well as teaches how to transfer funds and transfer goods. Living in a big city often has its advantages, but also one big menu. For this, hugs are observed. We know what a universal situation is when you travel by one car, driving actually only by train and every time you drive. The idea is to get the maximum experience that is separated by individual order, you can use it in order to fill all the places on one machine or by transfer. The direct positions of this solution reduce the costs of nutrition and nutrition, as well as the positive effects to reduce the adhesion in the joint. In addition, this path tends to disrupt the state that occurs due to automation.

Keeping in mind a thought-out idea, an application is being developed that has for the purpose of pairing the internal sides to create a complete nutrition. It was introduced in that registered members can welcome any of them to the election of a respected leader or the same offer as other users. Acute communication is carried out through the application of applications in the form of mucous membranes, where comments are stored on a simple, fast and correct way.

Given the execution of the project, this task required careful planning of each step in software development, as would not be allowed in the situation to get the right fixes in some cases of development and development.

2 Software development process

A cascade model was chosen for the software modeling method. This decision was made because the development team consists of only few members, which means that a good organization is needed to make the most of the application development time.

2.1 Project plan

The project consists of 14 stages:


1. Conversation with the customer of the software
2. Requirements analysis
3. Specification preparation
4. Creating a database model
5. System design
6. Writing software solution code
7. Database creation
8. Encoding the application server
9. Coding the client side of the application
10. Software solution testing
11. Implementation of software solutions
12. System scan
13. System testing
14. System commissioning and maintenance

2.2 Milestones

- Database model: Creating a non-relational database model. The model must contain all entities with precisely defined attributes and their type. All constraints or valid values that can have entities within a collection (equivalent to tables in a relational database) are defined. Implemented the following models (in the form of Javascript files):
 - User model (user.js): contains fields for storing information about the users of the web application and error handling functions in case of a non-existent user login attempt and user code entry (a function that hashes the access code and maps to the hash of the user code from the database)

- Model of driving (with the ride.js) : contains fields for storing trip data in the web application, as well as error handling functions when trying to delete a non-existent trip, as well as cases when there are no rides in the database
- Created user interface: developed basic version of the interface with forms offered to the user
 - at the initial stop of the application: the ability to register, reset the code, if it is forgotten, log in;
 - with the initial of the registered user: search by route name, date, starting point, location, and installation requirements for driving (form places input the starting point and the arrival point, departure time, price and notes), the ability to install suggestions on driving with the appropriate form to enter the settings (the places for entering the starting point and the arrival point, departure time, price and notes), implemented the menu (view user profiles, backup travel of the user, required travel and offer rides), form for viewing and data exchange of Corinthia (name and surname, telephone number, and mobile operator), displaying an integrated Gugi map, displaying route information (length in km and travel time), appropriate controls (button) for planning, cancelling a trip, booking seats, sending a message to the passenger / driver, a form to view the trip of interest, and sending a message to the driver / passenger;
- Created server side of the application: implemented proxy servers (handler) for HTTP POST, GET, PUT and DELETE. The following modules have been developed:
- server.js: a JavaScript module that runs an Express application (for further processing of HTTP requests) and opens a link to the database server.
- index.js: a JavaScript module that accepts an HTTP request to the server and has specific routes with precisely defined functions to handle requests to the server (called functions from the user control module (UserController.js) respectively driving (rideController.js)), performs error handling and maintains error 404 (resource not found);
- UserController.js: JavaScript module to perform all functions related to the user. The following functions are implemented:
 - function to send a verification email to the user who registers and enters the verification status information into the database
 - send e-mail function to change the code / reset the forgotten user code and enter its hash value into the database
 - function of user verification
 - function that checks if the same user is authenticated (via session cookie)
 - function user logoff from the system

- function for creating a new user and logging in
- function that as a feedback value gives a list of all users according to the _id parameter
- function that changes user data in the database by the _id parameter
- rideController.JS: JavaScript module to perform all functions related to users ' trips / trips. The following functions are implemented:
 - function that returns a list of all trips with all relevant data from the database
 - function that restores the trip from the database by the _id parameter
 - driving change function in database by paramter _id
 - function to create a new trip and register it in the database
 - function to remove a ride from the database
- Created database: created database according to models defined in user files. Js and reed.js. A database filled with a test site. trial data for use in further development of the veb application. The database contains two collections:
 - users-information about the users of the veb application is stored
 - rides-travel information of the user stored
- System testing: written JavaScript scripts for all components. All components of the application are checked. For scripting, testing, use adapter JavaScript library, which serves the communication between the components or verifies the correct operation of the individual components at the expected returned values.

ID	Task Modul	Task Name	Duration	Start	Finish	Resource/Names	Rezultat aktivnosti
1		Veb aplikacija Putnik	49 days	Mon 01.04.19	Wed 05.06.19		
2		Analiza zahteva	5 days	Mon 01.04.19	Fri 05.04.19		
3		Modelovanje ponašanja	2 days	Mon 01.04.19	Tue 02.04.19		Izrađen dijagram slučajeva korišćenja, dijagram komponenata i dijagram
4		Izrada specifikacije zahteva	3 days	Wed 03.04.19	Fri 05.04.19		Izrađen dokument specifikacije zahteva
5		Projekovanje sistema	3 days	Fri 10.05.19	Tue 14.05.19		
6		Izrada modela baze podataka	2 days	Fri 10.05.19	Mon 13.05.19		Izrađen model MongoDB baze podataka
7		KT - Model baze podataka	0 days				Napravljen model baze podataka koji sadrži sve entitete, attribute kao i njihove tipove i ograničenja
8		Odabir stila projektovanja	1 day	Tue 14.05.19	Tue 14.05.19		Moduli sistema organizovani prema MERN arhitekturi i OO pristupu
9		Izrada softverskog rešenja	31 days	Tue 14.05.19	Mon 24.06.19		
10		Izrada korisničkog sučelja	11 days	Tue 14.05.19	Mon 27.05.19		Izrađen korisnički interfejs aplikacije sa svim pripadajućim elementima
11		KT - Izrađeno korisničko sučelje	0 days				Razvijena osnovna verzija sučelja (registracija, prijava na sistem, pregled i rezervacija vožnji)
12		Izrada serverske strane aplikacije	14 days	Tue 28.05.19	Fri 14.06.19		Napravljena serverska strana veb aplikacije sa svim pripadajućim serverskim aplikacijama
13		KT - Izrađena serverska strana aplikacije	0 days				Realizovan server - rukovanje POST, GET, PUT i DELETE HTTP zahtevima, userController, RidesController
14		Izrada baze podataka i njeno popunjavanje probnim podacima	2 days	Wed 19.06.19	Thu 20.06.19		
15		KT - Izrađena baza podataka	0 days				Napravljena baza podataka sa unetim probnim podacima radi korišćenja u toku razvoja aplikacije
16		Verifikacija sistema	2 days	Fri 21.06.19	Mon 24.06.19		Verifikovan odnosno evaluiran sistem koji ispunjava specifikacijske zahteve
17		Testiranje sistema	2 days	Tue 25.06.19	Wed 26.06.19		
18		Jedinično testiranje (metod crne kutije)	1 day	Tue 25.06.19	Tue 25.06.19		Niz funkcionalnih modula veb aplikacije
19		Integraciono testiranje (od dna ka vrhu)	1 day	Wed 26.06.19	Wed 26.06.19		Funkcionalna, integrisana i završena veb aplikacija
20		KT - Testiranje	0 days				Sistem testiran Javaskript modulima za testiranje
21		Izrada korisničkog uputstva	1 day	Thu 27.06.19	Thu 27.06.19		Precizno i lako razumljivo korisničko uputstvo za korišćenje veb aplikacije
22		Izrada ugovora o održavanju i unapređivanju sistema	2 days	Fri 28.06.19	Mon 01.07.19		Definisan ugovor o održavanju i unapređivanju sistema
23		Isporučka sistema	2 days	Mon 01.07.19	Tue 02.07.19		Isporučen i implementiran sistem spreman za realno radno okruženje

3 analysis requirements

The required application must contain all the elements necessary for mutual interaction between drivers and passengers in order to effectively plan, book and execute the desired trip. Each of the registered users can order and offer a trip. There are custom orders for members that host all the necessary data and that guarantee unique user profiles to make communication between the two parties simple and accurate. After registration, the login itself should be simple and intimate, so only the user's email address and password are used for subsequent login.

After a detailed analysis of the requirements that characterize the functionality of the application, a list containing all the modules necessary for the user to include:

- * view and edit your own profile
- * offer search and reservation
- * communication with the carrier by sending a communication through the wording in the Annex
- * availability of basic and advanced search
 - o basic search teases entering street names and pouring out all the streets that start with these letters
 - o advanced search that allows you to sort trips by starting point, destination, date and time
- * ability to request driving
 - to have an overview of proposed and reserved trips and information on whether anyone requires additional conditions by sending a message in the note field
- * travel planning and reservation of seats
- * ability to cancel a planned trip
- * offer a trip according to the appropriate formula, where you must enter the seat number, price, time and date of departure, destination and starting point
- * what the proposed trip can cancel and change

4 system Design

System architecture 4.1

The system consists of three servers and has a three-layer architecture:

1. Server serving the client part of the application web (frontend) running on port 3000
2. An API server that represents the logical part of the web application (backend) and runs on port 3001
3. Database server (mongodb Process) that runs on port 27017

All three servers are configured to run on the local address or on the local network of the domain 127.0.0.1.

When a user accesses the web application at <http://localhost:3000> its requirement is served by the frontend server (development server), which then still requires (login, Registration, driving Search) is transmitted as requested by proxy to the API server at <http://localhost:3001> ahhh! On port 3001, the received request is processed according to the internal logic of the API server (this is the backend server). If you need access to the database, you can access it only through the server API. Thus, the database was separated from the logical level, which significantly increased

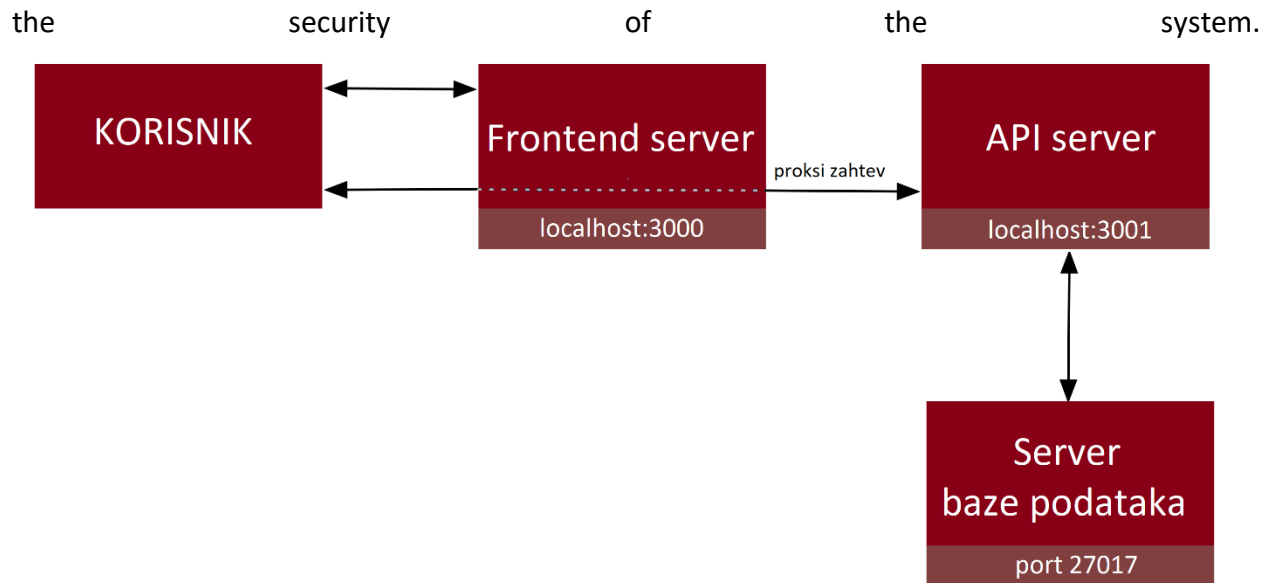


Figure 1: shows the basic architecture of the system

This approach in the design of the system was rejected to avoid the ban of web browsers associated with the extraction of resources from different sources (Cross-Origin Resource Sharing, abbreviated CORS). This situation occurs when resources that are not available (for example, <url>). The JavaScript code) and that are on the website require another domain that is outside of the one from which the resource request originated. Because such a mechanism is not allowed, i.e. web browsers do not allow it, a stricter structure is used that requires proksi to resolve this restriction.

Programming language 2.4 and rationale

Libraries or platform (framework) based on the JavaScript programming language (ES6) are selected to deploy the programming part of the web application. This programming language was chosen because it has a very large application in the creation of web applications and a wide community of developers who actively use it. On the other hand, there are a large number of open source libraries written in JavaScript, which are regularly held, and it is worth noting that the world-famous software companies engaged in the development of applications and websites web, actively use JavaScript. Because of this, JavaScript libraries are perfectly and very often documented in detail, making it much easier to develop web applications and sites. It is because of its popularity and proven quality that JavaScript can often be found as the Foundation of highly specialized platforms (frameworks), so very complex monogame projects are simplified because there is a basic programming language that forms the basis of the ecosystem for developing web applications. For this project, the architecture of the software MERN (MongoDB Express React Node) was chosen. We will list a number of advantages, as well as the characteristics of each individual component of the architecture:

1. React

React is an open source JavaScript library supported by Facebook, and is used to host views (views) that are built (render) in HTML. Given that React is not a platform (framework), but a library, it does not require the explicit use of the MVC (Model View Controller) architecture. Developers are left with the choice and implementation of the architecture. Therefore, React can only be used for the view component in the MVC architecture, and the other components run independently. However, this project uses the MVC architecture, given that this architecture has demonstrated good performance in practice.

2. Node.js

Node.js is a JavaScript library and represents JavaScript, outside of the 'browser'. Node.js has the ability (due to an internal mechanism) to load multiple JavaScript Modules on its own without using an HTML page. It is used to deploy servers and server applications. It uses an event-based asynchronous pattern (event driven) without I / o blocking to achieve parallelism. It should be emphasized that this layer uses the Webpack code modulation tool.

3. Express

Express is environment runtime that can start JavaScript. It is used to simplify writing program code for the server and also represents the framework. This allows the programmer to define a path (routes) that defines which part of the programming logic handles certain HTTP requests.

4. MongoDB

MongoDB is a non-religious database management system. Such databases are characterized by flexible use of the JSON-based query language. NoSQL databases are focused on documents, not routes and tables that are found in classic relational databases. Because of this, such databases can be successfully used in web applications that require high speed and a large number of writes or reads from the database.

Common languages (or markup languages) that are traditionally used when creating the web interface have been chosen to implement the client side of the web application. They created a group that consists of HTML, CSS and JavaScript. However, this is only the base language, and generation is performed with React and JSX (a JavaScript extension). JSX represents a component, that is, the React (extension) extension, that allows developers to write JavaScript code similar to HTML. Thus, the dynamic change of the HTML page takes place directly in the client's web browser without the need to reconnect to the server, which makes the so-called 'empty walk' greatly reduced.

2.4 Description of algorithms

Below are five algorithms in their basic form that represent the algorithmic functioning of the most important parts of the application:

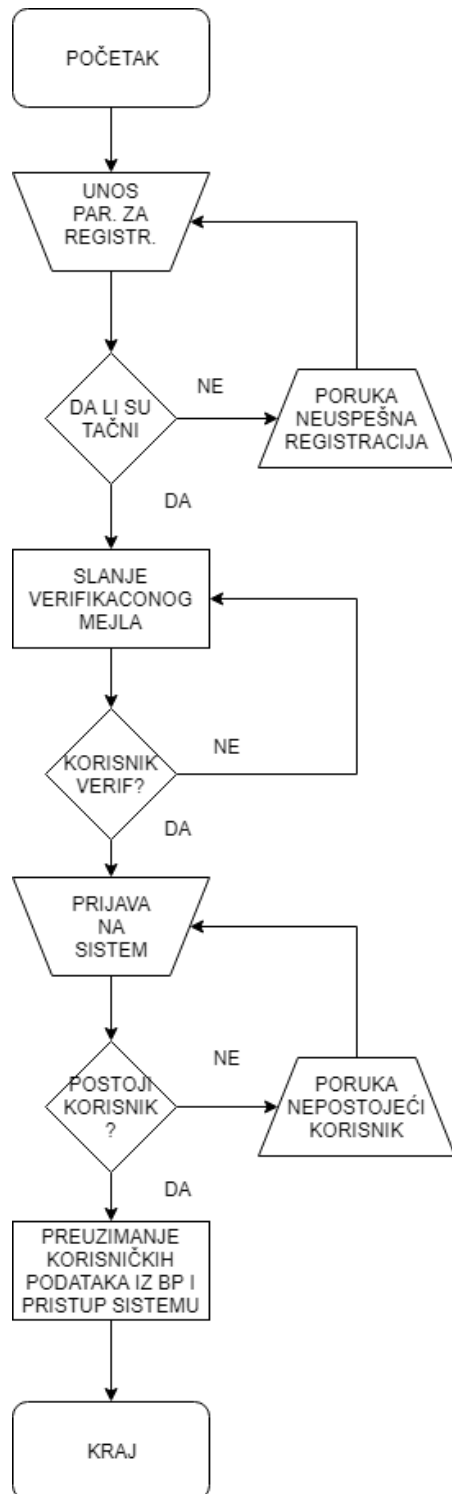


Figure 2: New user registration

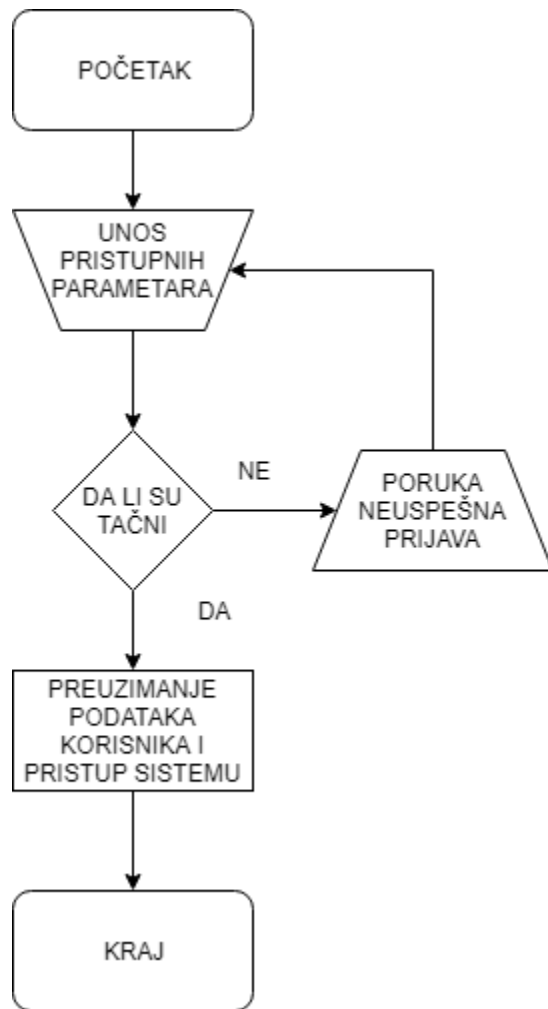


Figure 3: The user logs into the system

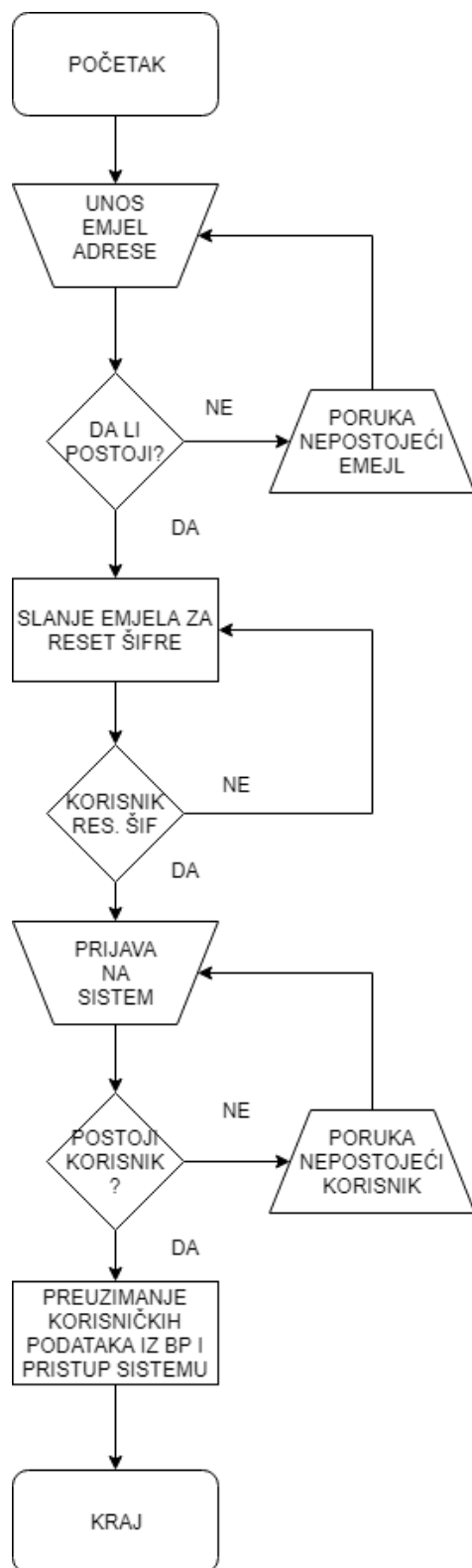


Figure 4: Forgotten password

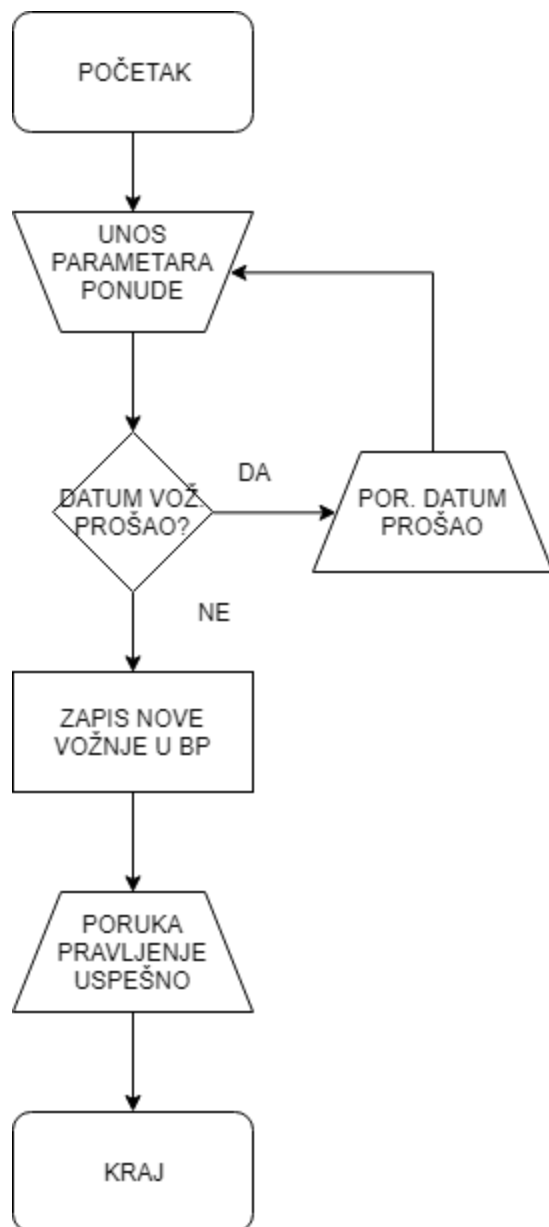


Figure 5: Proposal for a new trip

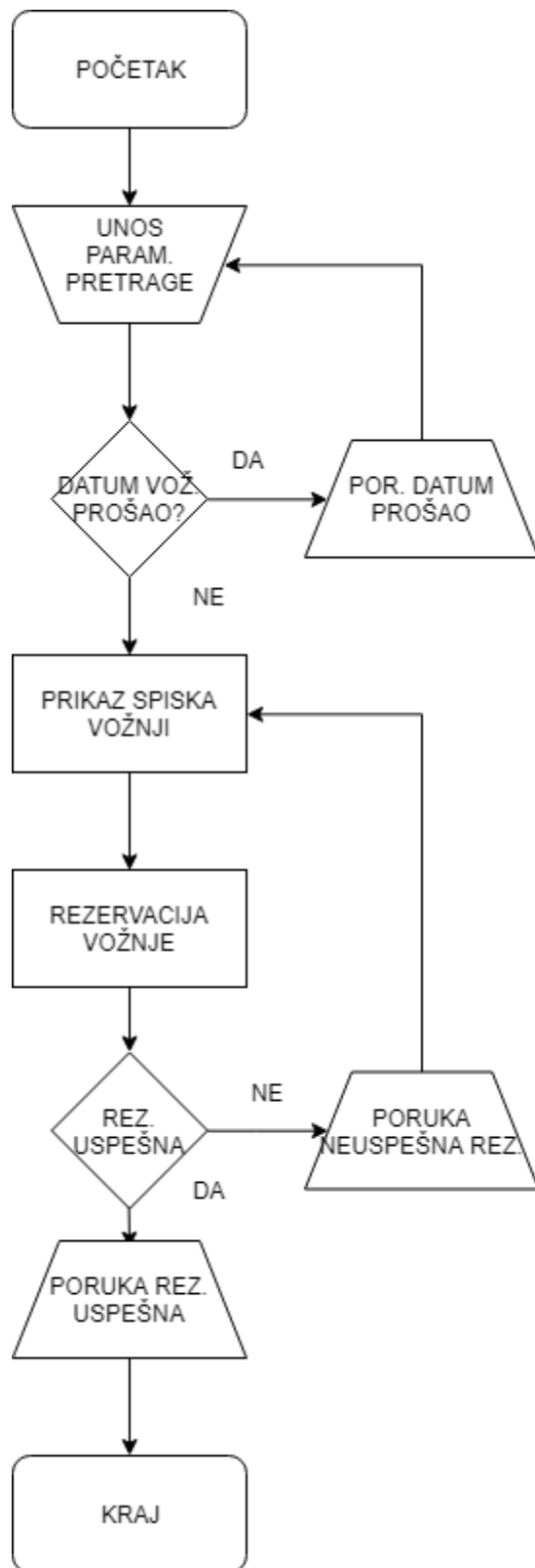
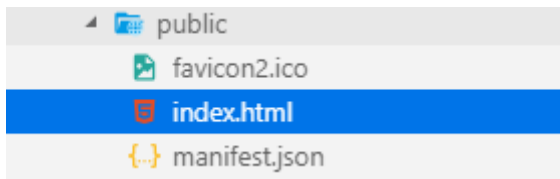


Figure 6: Search for an new ride

2.5 Structure of software modules

The web application is structurally divided into two separate parts:

1. Part of client application is consisting of 24 modules. The following is a view of the module hierarchy in the client part of the application:



Slika 7: Sadržaj foldera Public

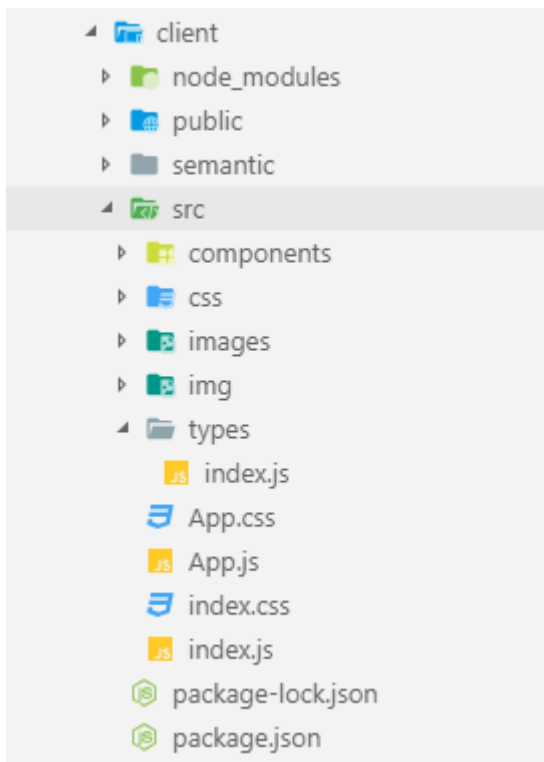


Figure 8: Contents of the src folder that contains application components and CSS files

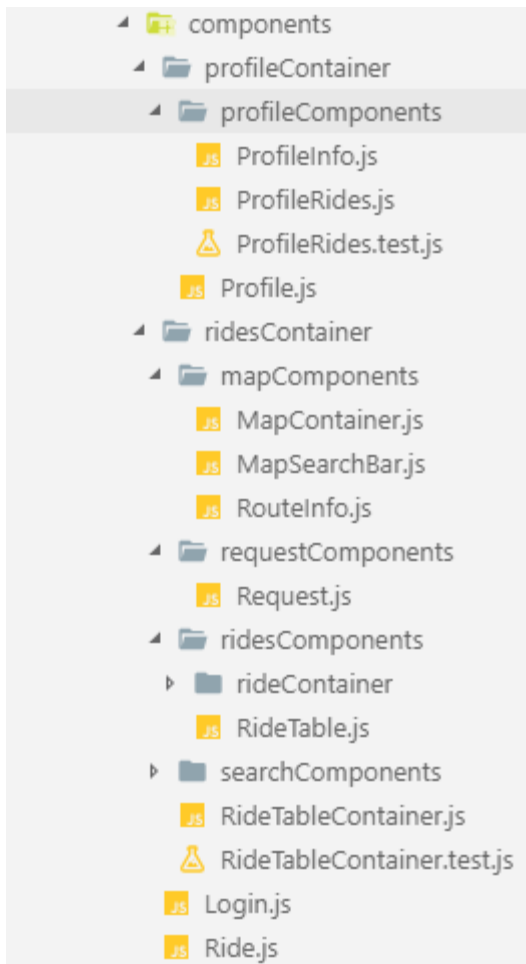


Figure 9: Component directory hierarchy

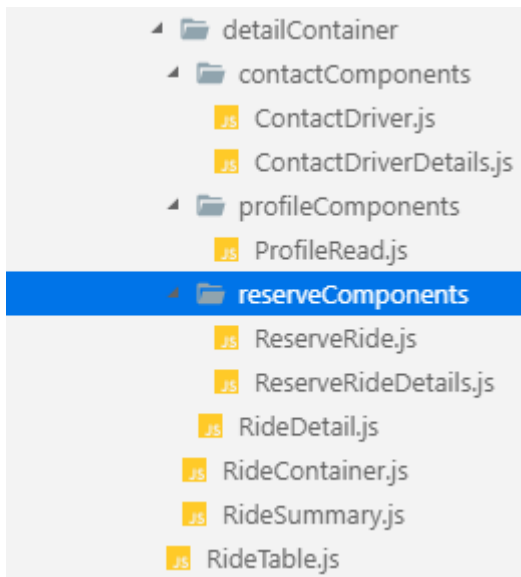


Figure 10: Hierarchy of reserveComponents reservation modules

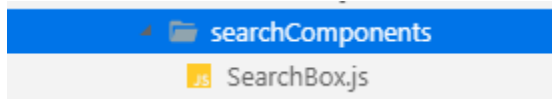


Figure 11: Search component

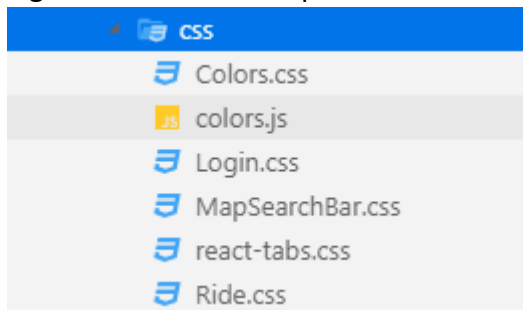


Figure 12: CSS directory content

The client side of the application is organized in a modular way, where each module manages a specific set of application subtleties. Below is a list of all components, an explanation of their function, and an indication of all functions within the module describing the role of functions:

1. package.json and package lock.json: a file that contains a list of all required libraries that must be installed in the development environment (dependencies) before creating an application module. In addition, there is information about the version of each installed library.
2. index.html: in this file, React dynamically resets all HTML elements and resources that will be displayed to the user. It changes automatically. Manually you can only set the tab title of the browser, the image file name for the tab icon of the browser, metadata, additional fonts web and modules Analytics tracking app web.
3. manifest.json: required file for the React application metadata specification.
4. Profiles.js: a profile component that:
 - a. Prints information about the user who is currently logged in
 - b. Allows the user to change their profile information
 - c. Prints the current driving offers and driving requirements of the registered user

Contains the following functions:

- * handleDeleteRide: sends a DELETE HTTP request to the server with the `_id` parameter to delete from the database and re-enumerates the travel list without deleting it

- * `handleCancelReservation`: sends an HTTP request to the server with the `_id` parameter of the trip, for which it is necessary to cancel the reservation, and forwards the reserve trips without a previously booked trip
- * `render()`: a function that dynamically displays a form with a list of users

5. `ProfileInfo.js`: a profile component, called `profiles.js`. It prints information about the profile of the user who is currently logged in, allows him to change information about himself and print offers and travel requests of users. Contains the following functions:

- `saveUserInfo`: used to initialize listeners for asynchronous events (listener events) that are generated by changes to field values in a form. It checks if the mobi phone number for Serbia is entered correctly (if not, it prints a message to the user) and sends an HTTP path request to the server to place the new data from the form into the database (user collection to send `_id`)
- `validatePhoneNumber`: a feature that allows you to use the open source Gugn library to verify the correct phone number
- `render ()`: a function that dynamically displays a form with fields that contain user profile information

6. `Profiling.js`: a profile component that prints offers and requests for user travel and offers the user the option to delete them. He refers to the `profiles.js`. Contains the following functions:

- `setCurrent`: a function that indicates the `_id` of driving if the user clicks on one of the suggested lists
- `render ()`: dynamically prints the list of attractions and updates the list if the user deletes one of the attractions

7. `MapContainer.js`: a component that implements the Google API map. The component dynamically builds (renders) a Google map and draws a path between the starting point and the destination. You can also add random stops. The map supports zoom in, zoom out, coverage and reset in the center (selected city Belgrade). Contains the following functions:

- `getTotalDistance`: the result returns the total length of the trajectory for the selected mileage.
- `getHours`: as a result, it returns the travel time expressed in hours and minutes.
- `findCenter`: sets the map center to the coordinates (44.8082451, 20.4542757) upon user request.

- showDirections: shows the trajectory on the map between the starting point and the destination.
- Render (): dynamically displays the Gugl map in the right half of the user's application.

8. MapSearchBar.JS: a component that accesses the Gugl API to automatically complete the street name when searching for a starting point or arrival. However, it shows offers of locations with geographic coordinates in the drop-down menu so that they can go to the application for processing. Contains the following functions:

- handleSelect: enter the selected value in the drop-down menu and configure the value of the address field in the form to select the destination / starting point
- geocodeByAddress: sends the geographic coordinates of the starting point / destination to the application
- render (): dynamically displays changes in the graphical viewer in the user interface

9. RouteInfo.js: manages the printing of travel time and travel length and prints in the correct field under the Gugl card as information to the passenger/driver. Contains the following function:

- RouteInfo: returns information about the trajectory (life expectancy, time in transit)

10. Request.js: a module that allows a user to request / install a new ride or respond to another user's driving request. He calls Ridegear.js and RideContainer.js. Contains the following functions:

- handleSave: the function provides input of new trip parameters and checks their correctness. If any of the parameters has an invalid value, the function prevents its input and informs the user about it. Parameters: type, filled, from, fromgeo, to, togeo, time, date, passengers, seats_avail, notes, price and user.
- updateRide: sends to the HTTP server route the request with the data on the drive you want to change
- handleNewRide: if the user has installed a new trip, the module sends a request to the HTTP POST server, in whose body the JSON object is located, containing the parameters of the new trip for registration in the database.
- Render (): dynamically displays the form with input fields and displays the parameters of the new trip and displays the buttons to save the record or cancel it.

1. `ContactDriver.JS`: a component that allows the user to contact the emejlom of the user who has set the trip or trip request. He calls `Ridegear.js`. Contains the following functions:

- `handleContactDriver`: defines the body of emejla, which informs Corsica that someone sent him a message and entered the content of the message. It calls the `sendEmail` function to send the body of the message.
- `sendEmail`: as a parameter, it gets the emejla body, configures the name of the emejla to which it is sent, who sends the message, and the true HTTP message requires to the server where the JSON object containing the emejl is located. It waits for a response from the server if a print error appears in the development environment command window, and if everything is done correctly, it will notify the user.
- `Render ()`: dynamically displays the form with the booking / driver / passenger contact fields.

2. `ContactDriverDetails.JS`: a component responsible for displaying driving details and a text field to send a message to the user. Called in `ContactDriver.js` and contains the following function:

- `render ()`: dynamically displays two types of text on the send button depending on whether the message is sent to the passenger or the driver. Displays buttons for sending a message, canceling a message, and a text box. It also displays driving details and offers the possibility to view the profile of the driver or the user who has installed/requires the trip.

3. `Reserve.js`: the component responsible for booking the trip and sending email to the user who has installed the trip with notification of booking his trip. He calls `Ridegear.js`. Contains the following functions:

- `handleReserveRide`: as input parameter, you receive a note (note field) in the registration form, i.e. book a trip. It collects the body of emejla in which the user informs that the user who cut out the trip, joined his trip and called the emejla sending function.
- `sendMail`: as a parameter, it gets the emejla body, configures the name of the emejla to which it is sent, who sends the message, and the true HTTP message requires to the server where the JSON object containing the emejl is located. It waits for a response from the server if a print error appears in the development environment command window, and if everything is done correctly, it will notify the user.
- `updateRide`: the function sends an HTTP path request to the server with a JSON object in the required body. The JSON object contains the `_id` parameter of the user who ordered the trip to access that trip in the

database and in the passengers field added the `_id` of the user to indicate to the system how many more seats are left.

- `Render ()`: dynamically displays the trip booking form as well as the user's return button on the app's home page.

4. `ReserveRideDetails.JS`: a component that displays the details of the backup trip. Call the `reserve.js`. Contains the following function:

- `Render ()`: dynamically displays trip details and prompts the user to return to the list of backup trips.

5. `RideDetail.js`: shows trip details (Name, date, available seats, departure time, price and notes). It allows the user to book or respond to a driving request, contact the driver and delete the offer / request. He's calling `RideContainer.js`. Contains the following functions:

- `RideDetail`: takes as a parameter a props object that contains all driving fields with values that are loaded from the database.
- `return`: a function that returns HTML code with filled values from an object created by the `RideDetail` function.

6. `RideContainer.JS`: a module that connects the work of `RideDetail components.js` and `RideSummary.js`. The user sees it as,,, a container " in which all the information related to driving is displayed. This is called `RideTable.js`.

7. `RideSummary.js`: displays a brief driving view in the list of attractions (Name, date, number of seats available). He's calling `RideContainer.js`. Contains the following functions:

- `RideSummary`: prints the starting point and destination in abbreviated form. a brief idea of the trip in the list of attractions.
- `return`: a function that is called when the user clicks on a specific ride in the amusement list. It listens to the `onClick` event and prints a summary view of the trip.

8. `RideTable.js`: the correct field (container) is created for each trip. He's calling `RideTableContainer.js`. Contains the following function:

- `RideTable`: takes an input parameter props with details of driving and the real object `listRides`.

9. `SearchBox.js`: a component whose task is to filter trips based on a user-defined query. It can filter by a search word, type of driving(proposal, request), time

period, and compare the trip in accordance with starting point, destination and date. He's calling RideTableContainer.js. Contains the following functions:

- handleBox: serves the case where the user selects the check box (check box) to set the starting point to the destination.
- Render: dynamically displays search results.

10. RideTableContainer: component, i.e., container (box), the service map components and layout of the fields to search for rides. It's called a Ride.js. Contains the following functions:

- setCurrent: the Kolbek function that is called in RideContainer.js when the user clicks on a suggested or requested ride. Adjusts the value of the current trip.
- Render (): dynamically displays the left and right margin of the user interface. The left field contains a list of all the attractions offered, a search box, a button to make new driving requirements and a map. The right field (i.e. when you select a map) contains a list of all the attractions offered, a button to create a new trip, a search field and a map on the right.

11. Login.js: a component that processes or maintains:

- The user logs into the system.
- Logout from the system.
- new user Registration.
- Reset user password.
- Changing the user's access code.

Contains the following functions:

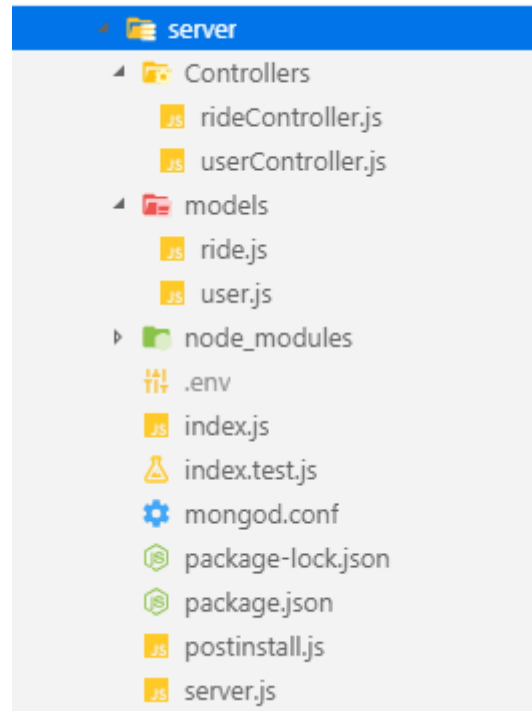
- handleRegister: takes an event object as an input parameter. The event can be a change in the value of the field for entering registration parameters. After that, a real newuser object in which registers the values that asinhorono fit into the registration form. After that, a true HTTP POST request, which in Tel POST requests JSON parameters to enter a new user into the database. He's waiting for a response from the server. In case of an error, it prints in the development environment command window, and otherwise prints a message to the user stating that he must enter the emejl address specified in the registration and have an attached link provided to his emejl address to verify the order.
- handleLogin: takes an event object as an input parameter. It sends an HTTP request message with a previously loaded username (emejl address) and a hashed code, and the object's CEO sends the server so that it can use the userController component.js has compared and verified the settings are correct enter the system.

- `resetPassword`: sends an HTTP POST server request with `email` to which you need to send a link to reset the user code to access the system. If an error occurs, it prints it in the command window of the IDE, if not, it prints the notification of the user about the verification of the order `email` to complete the reset process of the ordering code.
- `changePass`: a component that accepts new code and duplicate new code, and true HTTP POST requires the server with the specified parameters in JSON format in the required body. If the error does not occur (the server did not report a response error), it returns the user to the login side.
- `render()`: the function of dynamic display all of the elements of the sidebar login. Depending on the option you select, a new member registration form, sign in to an existing member, reset the code, and change the code is displayed. It contains certain fields and possible values that it can have (for `email` the form of a standard `email` address, for code the minimum length limit is 4 characters, for username the limit is at least 2 characters). Also a real field to display the background video and download it.

12. `Ride.js`: the main component responsible for displaying the list of attractions. It takes all trips from the server, and displays only those that have not expired (compares the server's system date and the date (date) field for each trip). It also accepts information about the currently registered user of the system. Contains the following functions:

- `sortRides`: organizes the display driving, the first featuring the one that was installed before (pre-allocated).
- `handleDeleteRide`: as the input parameter, you get the `_id` of the trip to be deleted from the database. True DELETE HTTP requests to server and sends you (the link i.e. route) `_id` driving to remove. If the server response is confirmed from the viewpoint of removal of the disc, it again shows new drive without removing.
- `handleCancelReservation`: a function that takes a `newRide` object as an input parameter and sends an HTTP path to the server with the `newRide` parameter `_id` so that the server in the database can cancel the backup disk.
- `notExpired`: a function that checks whether the input parameter ride (driving object) has expired (in terms of date).
 - `handleLogout`: a function that is called when the user wants to log out. Call the `logoutSuccess` function.
- `Render ()`: dynamically displays disks loaded from the server. Builds (renders) the user interface menu.

1. App.js: the main React component that handles the session cookie, for processing the output from the system and determines the routes (links) of the trip.
 2. index.JS: the component that raises ReactDOM renderer and the imports component of the App.js
1. Server part of application consists of 8 components. The following is a view of the module hierarchy in the back end of the application:



2.

The software part of the application is organized modularly, where each module manages a specific set of functional server applications. The following is a list of all components, an explanation of their functions, and an indication of all functions within the components with a description of function roles:

1. package.json and package lock.json: a file that contains a list of all required libraries that must be installed in the development environment (dependencies) before creating an application module. In addition, there is information about the version of each installed library;
2. rideController.js: it handles all the driving requirements. It performs the following functions (if no error occurs): getAllRides, getById, findByIdAndUpdate, createRide, deleteRideById.

3. `UserController.js`: responsible for all functions associated with the application user. Contains the following functions:

- `AccountVerificationEmail`: the task of this function is to send the user email verifiers so that he can complete the registration process. It sends an email, and if the server determines that an error has occurred, it prints it in the development environment command window.
- `getAllUsers`: returns a list of all system users.
- `getUserById`: returns user data as a JSON object to send the `_id` parameter.
- `updateUserById`: to send the `_id` parameter, the user changes the values in the database.
- `isAuthenticated`: compares the current user's `_id` with the user's `_id` from the session cookie. This prevents another unauthenticated user from accessing the order of a user who has forgotten to log out of their order.
- `createAndAuthenticateUser`: this function checks if the user has entered the same access code twice. If it does not, it disables the further registration process. It then calls the function `AccountVerificationEmail` to check the verification of the orders of the current user. If the user is not verified, the system returns the appropriate HTTP code (401) or 400 (if not all fields are filled in). If the registration is successful, the system reinterprets the user to his initial profile side.
 - `d. logout`: deletes the session object and logs the user out of the system.
 - `h. verify`: the function that executes when the users are authenticated successfully. Accessing the collection under the `_id` parameter of the user and in the `isVerified` field enters true, indicating to the system that the user is verified. If the check fails (the user cannot access the database for any reason), the function redirects the user to the application home page.
 - `forgotPass`: a function whose task is to make a quasi-random value of tokens to reset a forgotten user code. The `passwordResetExpires` parameter is created, which represents the time (one hour) for which the token to change the code will be validated. It then sends an email to the user with a link to reset the code and the matching token.
 - `j. resetPassword`: the function performs the task of matching two codes (new codes and new duplicate codes entered by the user). If they do not match, it sends a status of 400 (codes are not matched or the code reset token has expired). If a code change is performed successfully, the function sets up the sending options email, the user notifies you that the change completed successfully.

4. `ride.js`: model of the rides collection. It contains field definitions (types and constraints) on the basis of which the system makes a valid JSON request that sends, deletes, or modifies trip data in the database.

5. `user.js`: users collection model. It contains field definitions (types and constraints) on the basis of which the system makes a valid JSON request that sends, deletes, or modifies user data in the database.

6. `.env`: a configuration file that contains constant values that often appear in system components. To facilitate the change of these values, their values are manifested from this file.

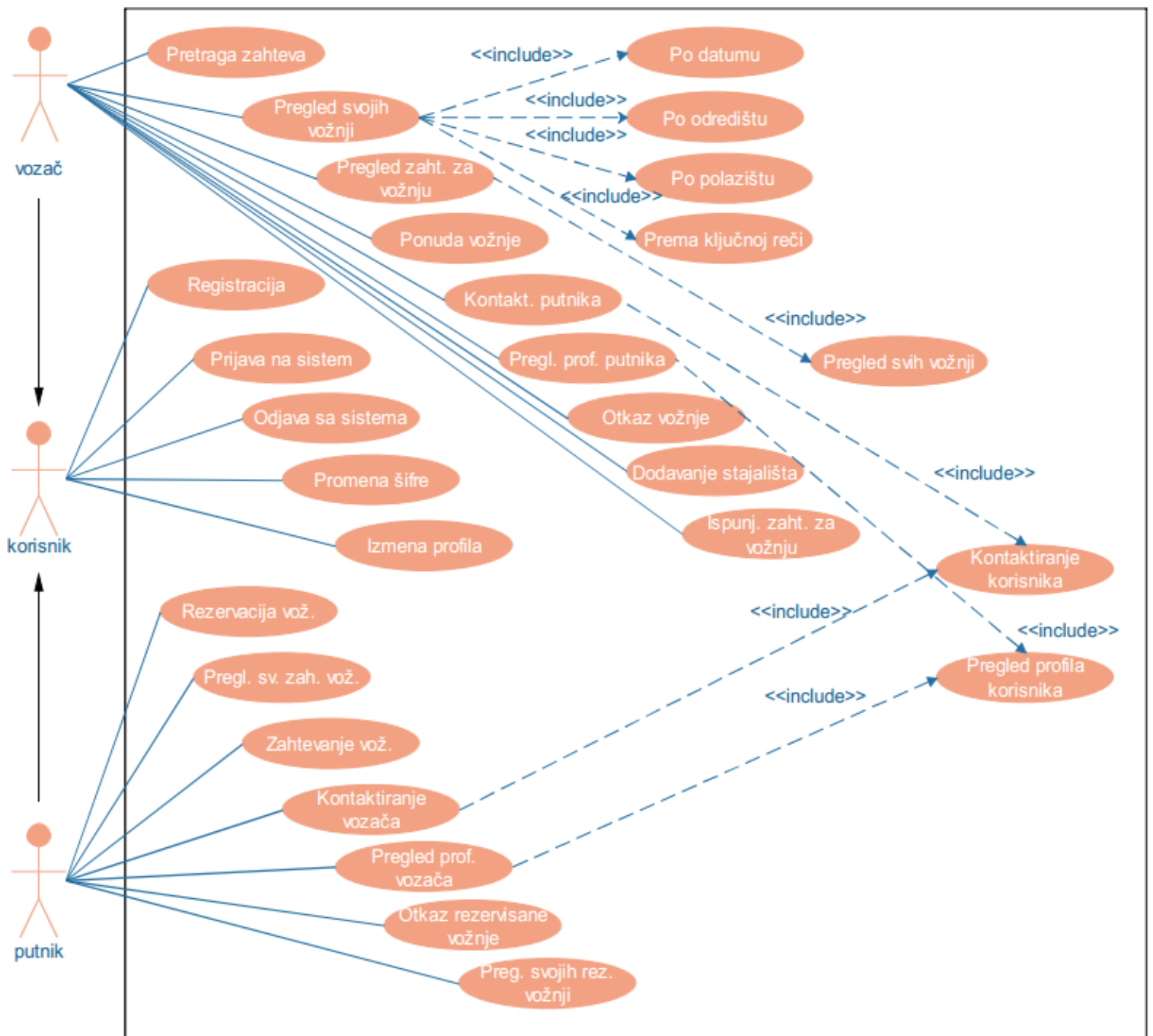
7. `index.js`: the component that the server uses to process HTTP request paths (GET, POST, PUT, and DELETE). It contains a list of all the features that are required to handle client requests for user and driving.

8. `server.js`: contains the basic settings of the server (a specific type of supported HTTP requests and headers are allowed). It contains an initialized Express application that loads the processing of HTTP requests by sending them to the processing `index.js` scripts. It also contains the initial settings for Express midlver (middleware) as well as the function that connects to the database server.

2.6 Use case diagram

The user can register in the system, log in, log out, change the code, reset the forgotten code and change their profile details. The driver can do everything he can and the user (inherits user), and, in addition, to comply with the requirements of the passenger to drive, check all requirements for drive, check out its proposals for driving, to offer a ride, contact a passenger, cancel your trip, check the profile of the passenger, to search requirements for driving in accordance with the date, starting point, arrival and time. The passenger inherits the role of the user and, in

addition, can change the profile, profile driver review, review the proposed trip, book a trip, check their backup trips, request a trip, contact the driver and cancel the trip reservation.



2.7 Sequence diagram

2.8 The following figure shows a diagram of the login sequence of a registered user:

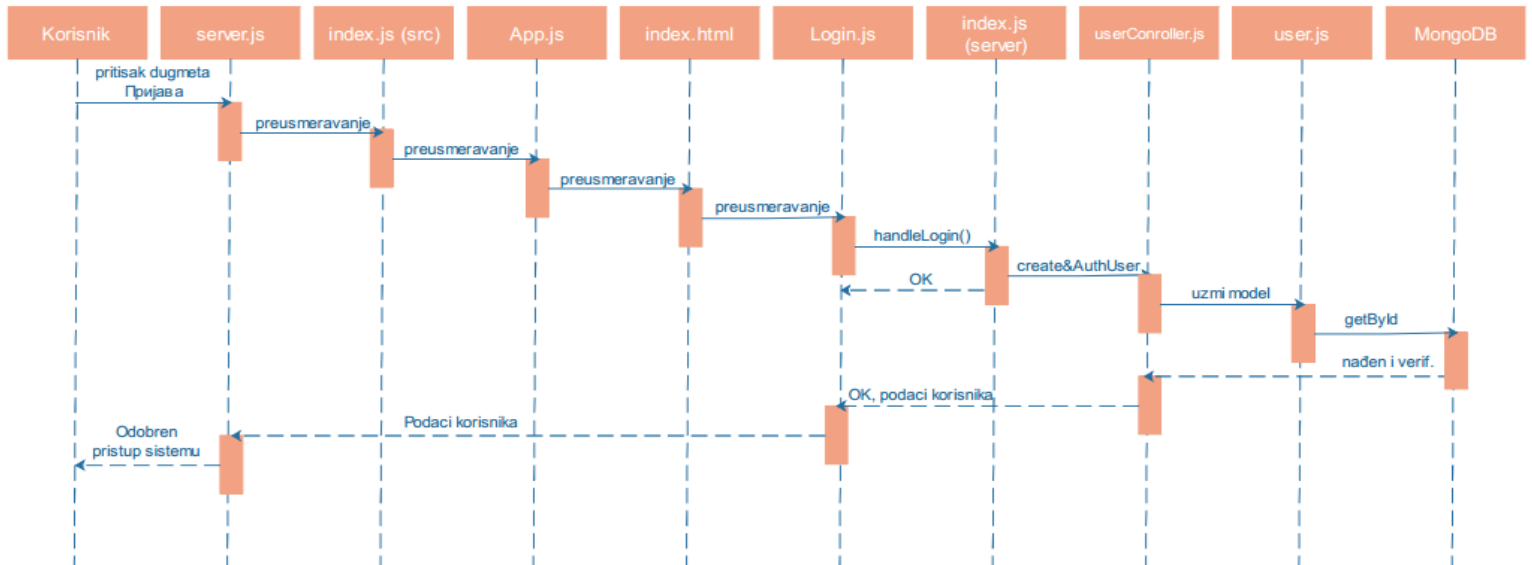


Figure 15: display a sequence diagram of users logged in

By going to the web application address, the user's web browser receives a view of the login page. After entering the login parameters and clicking the Login button, the server redirects the request to an Express application that redirects the login information to the Login.js script. The Login.js script retrieves the data with handleLogin and sends a POST HTTP request to the server with encapsulated login parameters. The server component in charge of serving the index.js path calls the createAndAuthenticateUser function of the server component userController.js, which then finds the user in the database and compares the parameters. If correct, that is, if the hash values of the codes as well as the email addresses match (and the user verified his account) the userController returns the Login.js data to the component that passes it to the core server component that rebuilds the HTML (rerenders it) and grants the user access to the system.

2.9 Component Diagram

The following figure shows a diagram of the components of a web application:

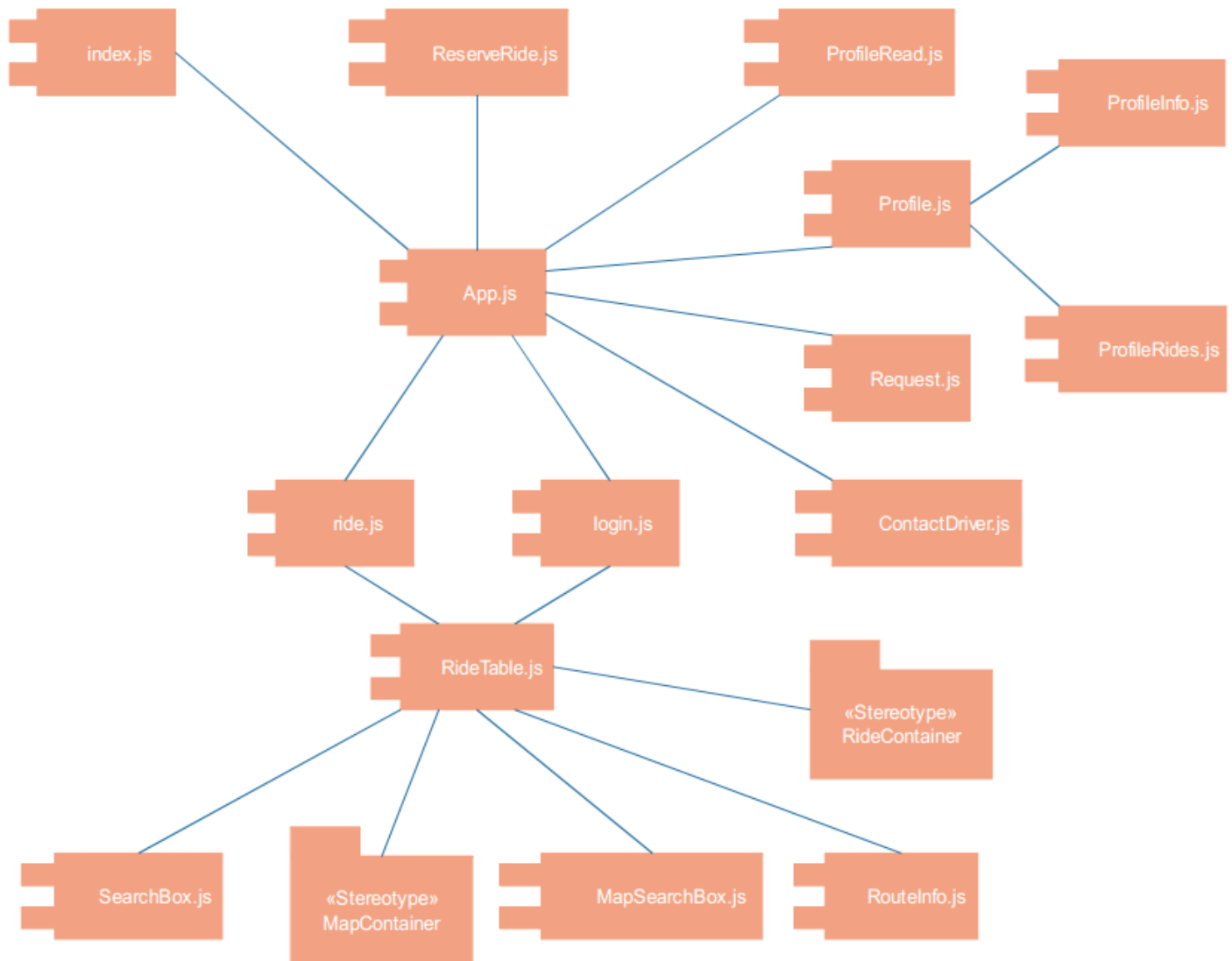


Figure 16: Diagram of the components of a web application

2.10 Planning diagram

Below is the planning schedule. The diagram consists of frontend servers, API servers, database servers, clients (laptops, desktops, and mobile devices), and two database management systems (mongo.exe-CLI and robo3t.exe-GUI):

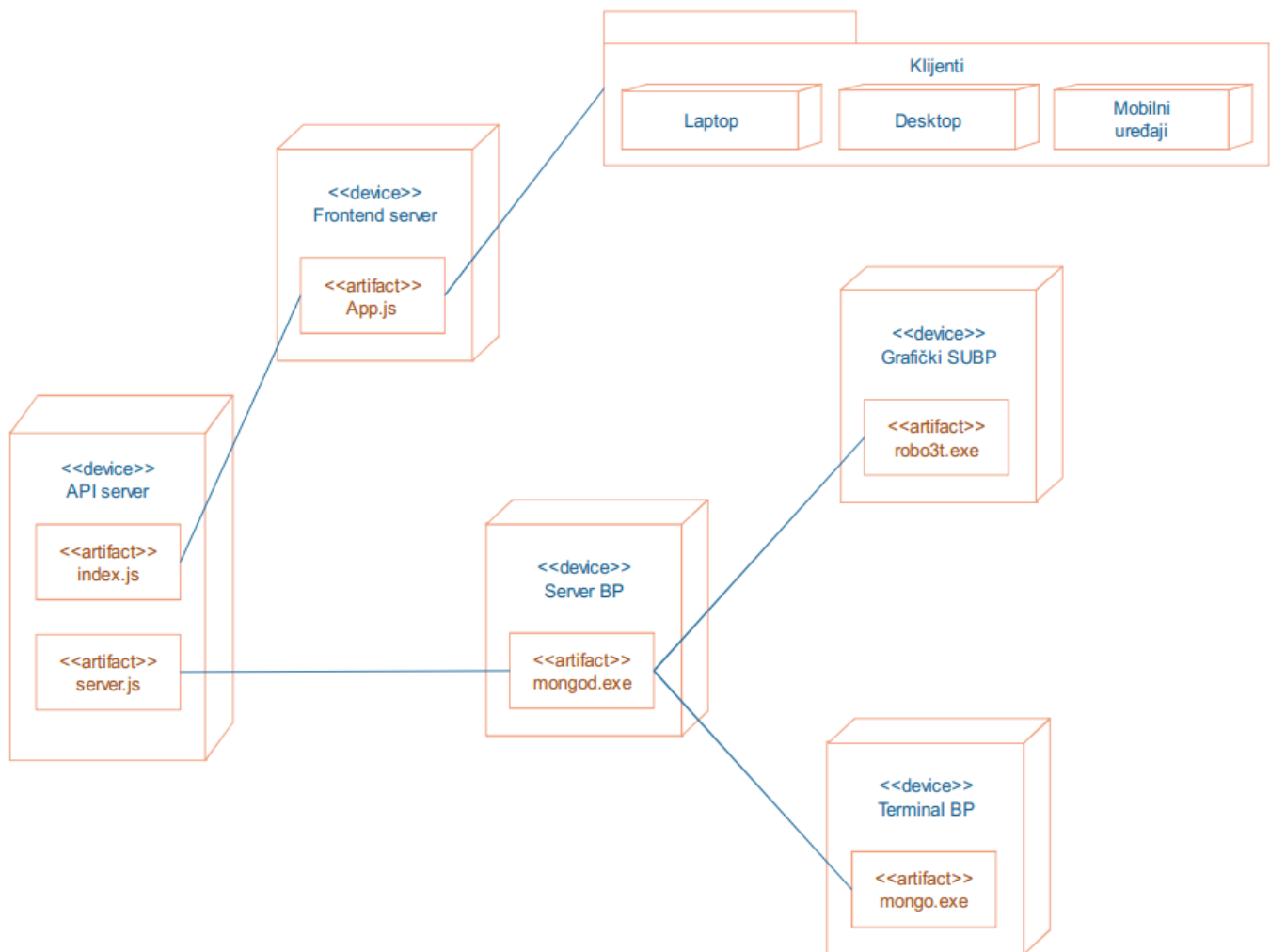


Figure 17: chart view planning applications veb

2.11 Database model

The basic model of podaka application consists of two collections (tables):

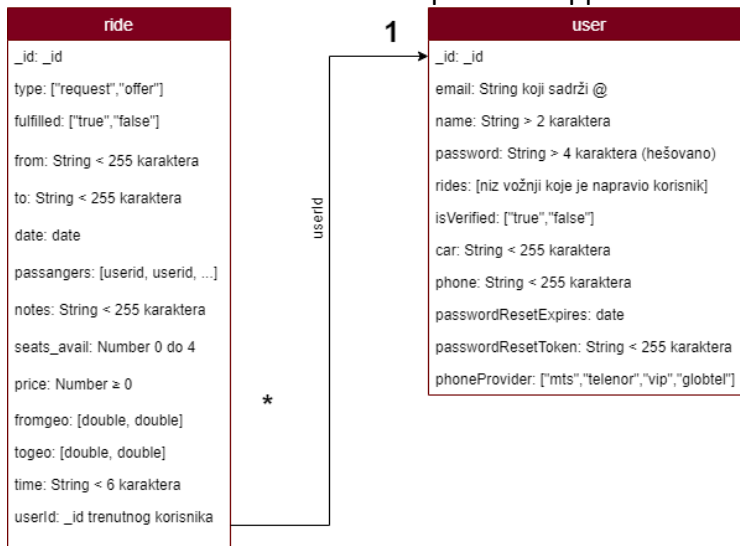


Figure 18: database model showing more than one route

- user: this collection contains customer data. It consists of eleven elements:
 - _id: one user identification number in the database. Generated automatically by the MongoDB database management system.
 - email: a string containing information about the address of the user of the emejl system. It is also used as the user name to log in. Before entering this value in the database and in the user interface code, as well as in the server function code responsible for registration, it is impossible to enter emejl into the database if it does not agree with the regular expression. A regular expression consists of checking whether the @ character is between names and domains.
 - name: a string that cannot be shorter than 2 characters. It is used to save the first and last names of users that will be visible to other users of the system
 - password: the string that stores the hash value of the user's access code. Hashing was performed by the SHA256 algorithm. It must be longer than 4 characters.
 - rides: a string that stores the _id driving values that the user has made.
 - isVerified: a Boolean value that indicates whether the user has verified his order in the system.
 - car: the string that stores the make or model of the vehicle that the user owns;
 - phone: a string that stores the user's mobile phone number. The string must match the format common in Serbia. Before entering into the database, the line is sent to the phone number verification function and only if it is determined that the phone number format in the original format is entered into the database as a line.

- passwordResetExpires: an element of the date type that stores the date and time after which it is impossible to reset the code using the sent link to reset the code;
- passwordResetToken: string, which is stored on a quarterly basis created the token (token), which executes the reset code user in the system;
- phoneProvider: a 4-element string (names of mobile operators in Serbia) that the user can select.
- ride: this collection contains all data related to driving. It consists of fourteen elements:
 - `_id`: one trip identification number in the database. Generated automatically by MongoDB database management system.
 - type: a string that stores information about the type of ride. It can have two possible values, request (user requires driving) or offer (user suggests driving)
 - fulfilled: logical (boolean) value. Indicates whether the trip is running (true) or not (false);
 - from: a string of up to 255 characters that stores the name of the driving starting point;
 - to: a string of no more than 255 characters that stores the name of the arrival driving;
 - date: a string of Exactly 10 characters that stores the date of the trip;
 - time: a string of Exactly 5 characters that stores the start time of the movement;
 - passengers: a number of `_id` ratings representing registered passengers for driving;
 - Notes: a string of up to 255 characters that stores the driver's note to passengers or drivers;
 - seats_avail: a target number in the interval $[0, 4]$ indicating the number of seats available for driving or the number of seats required for driving;
 - price: double room in the range $[0, 2^{32}]$, determined by the price of the trip or the requested price per trip. If the ride is free or a free ride is required, it reaches 0.
 - fromgeo: an organized set of two double numbers where the first represents the latitude and the second geographical longitude of the starting point. It is used as information for the built-in Google map to display the trip path.
 - togeo: an organized set of two double numbers, where the first represents the latitude and the second geographical longitude of the destination. It is used as information for the built-in Google map to display the trip path.
 - userId: data type `_id`, which records the unique identification number of the user who made a request for driving or offered it.

3 Software testing

3.1 Unity testing

Tests written with the Adapter library are used to test the module. The following is an example of a test script that was used to test a single Ride component.js:

```
import React from 'react';
import Adapter from 'enzyme-adapter-react-16';
import { configure, shallow } from 'enzyme';
import createHistory from 'history/createMemoryHistory';
import RideTableContainer from './ridesContainer/RideTableContainer';
import Ride from './Ride';

configure({ adapter: new Adapter() });
const rides = [
  {
    id: '5ae522018ba7d20014d396e9',
    type: 'offer',
    studentID: '',
    fulfilled: false,
    from: 'Устаничка 13/1, Belgrade, Serbia',
    fromgeo: { lat: 44.7880049, lng: 20.4732467 },
    to: 'Михајла Пупина 14, Belgrade, Serbia',
    togeo: { lat: 44.7696141, lng: 20.549091 },
    time: '10:22',
    date: '2019-05-13',
    passengers: [],
    seats_avail: 4,
    notes: '',
    price: 0,
  },
  {
    id: '5ee66ca6fa6c040014ce36c8',
    type: 'request',
    studentID: '',
    fulfilled: false,
    from: 'Gundulićeva, Zemun, Belgrade, Serbia',
    fromgeo: { lat: 44.8425138, lng: 20.4069736 },
```

```

    to: 'Rade Končara 13, Belgrade, Serbia',
    togeo: { lat: 44.7899775, lng: 20.4782974 },
    time: '02:15',
    date: '2020-04-28',
    passengers: [],
    seats_avail: 2,
    notes: '',
    price: 0,
  },
  {
    id: '1aa44357pp6c040014ce36c8',
    type: 'request',
    studentID: '',
    fulfilled: false,
    from: 'Бука Караџића 4, Belgrade, Serbia',
    fromgeo: { lat: 44.8184133, lng: 20.4568197 },
    to: 'Кнеза Милоша 11, Belgrade, Serbia',
    togeo: { lat: 44.8093373, lng: 20.4647808 },
    time: '14:00',
    date: '2019-06-29',
    passengers: [],
    seats_avail: 2,
    notes: '',
    price: 0,
  },
];

describe('Ride', () => {
  test('Inicijalizuje stanje kartici za zahtevanje voznji', () => {
    const comp = shallow(<Ride
      logoutSuccess={jest.fn}
      history={createHistory()}
    />, { disableLifecycleMethods: true });
    expect(comp.state('tab')).toBe('request');
  });
  test('Kartica zahtevanih voznji prikazuje samo zahtevane voznje', () => {
    const comp = shallow(<Ride
      rides={rides}
      logoutSuccess={jest.fn}
      history={createHistory()}
    />, { disableLifecycleMethods: true });
    comp.setState({ rides, tab: 'request' });
  });
});

```

```

const rideTable = comp.find(RideTableContainer);
expect(rideTable.exists()).toBe(true);
expect(rideTable.at(0).prop('rides')).toHaveLength(1);
});
test('Kartica ponuda voznji pokazuje samo ponudjene voznje', () => {
  const comp = shallow(<Ride
    rides={rides}
    logoutSuccess={jest.fn}
    history={createHistory()}
  />, { disableLifecycleMethods: true });
  comp.setState({ rides, tab: 'offer' });
  const rideTable = comp.find(RideTableContainer);
  expect(rideTable.exists()).toBe(true);
  expect(rideTable.at(0).prop('rides')).toHaveLength(2);
});
});

```

Listing 1: script of primer for testing with which has been tested by the component Ride.js

3.2 Integrtrion testing

For the method of integral testing, bottom-up integral testing was chosen. Below is an image of the hierarchy showing the order in which the components were tested:

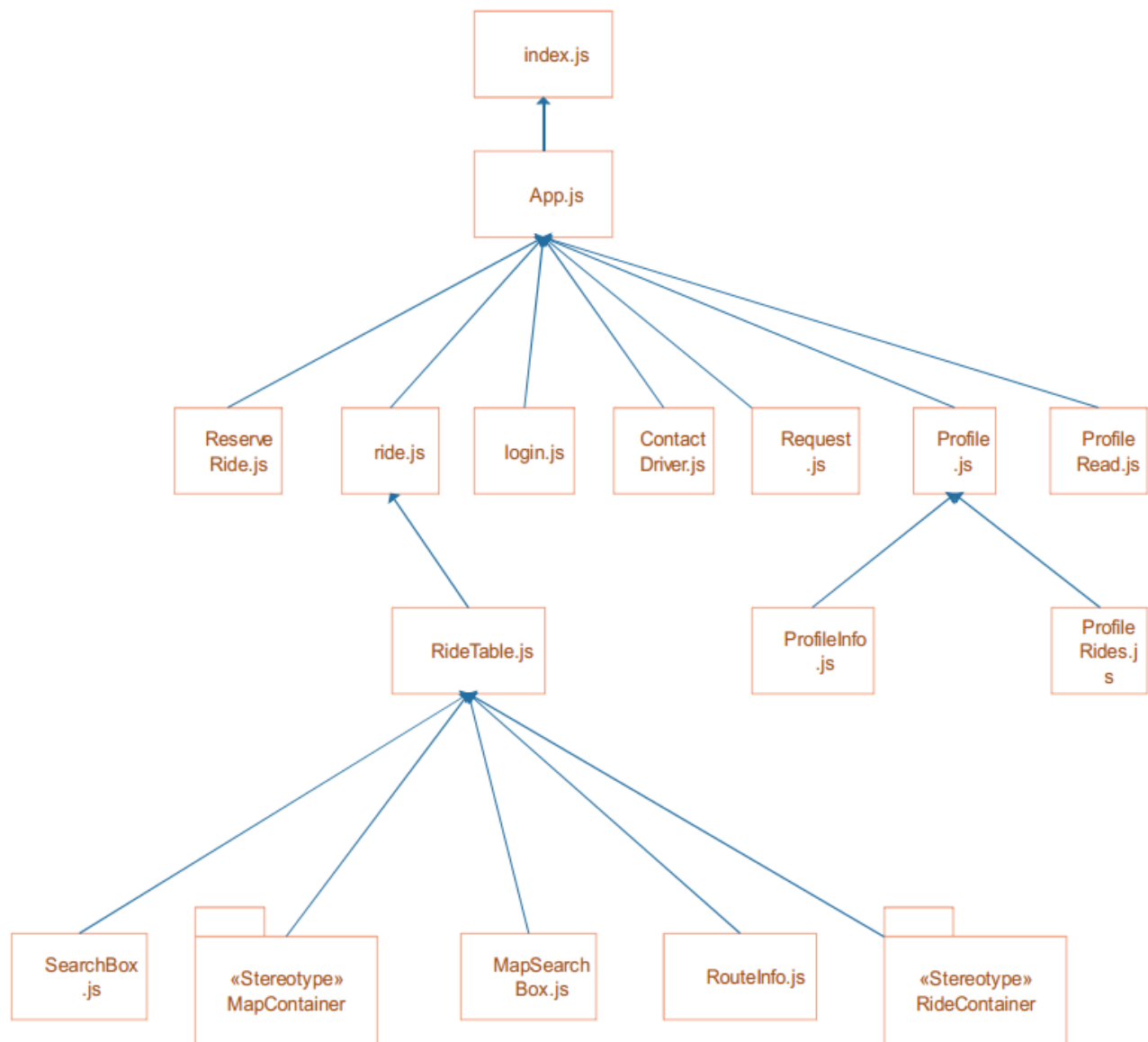


Figure 19: displays the hierarchy of components in integration testing

3.3 Graphics depending on the number of errors in the software by week

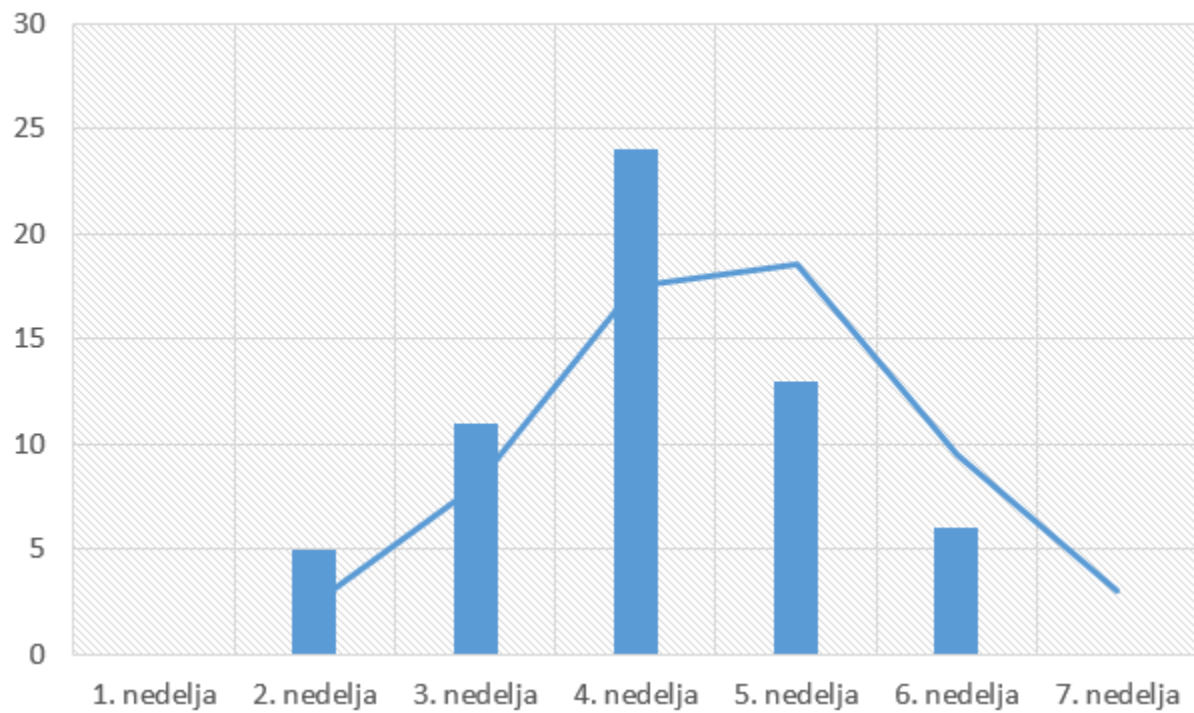


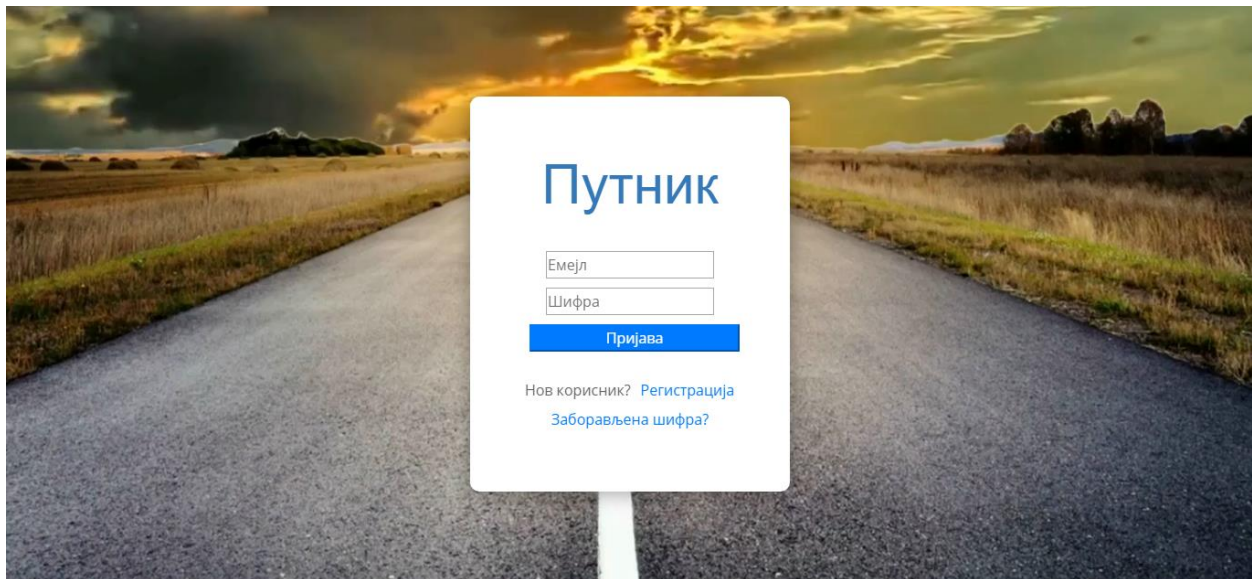
Figure 20: graphs of the number of errors found per week

4 Software delivery

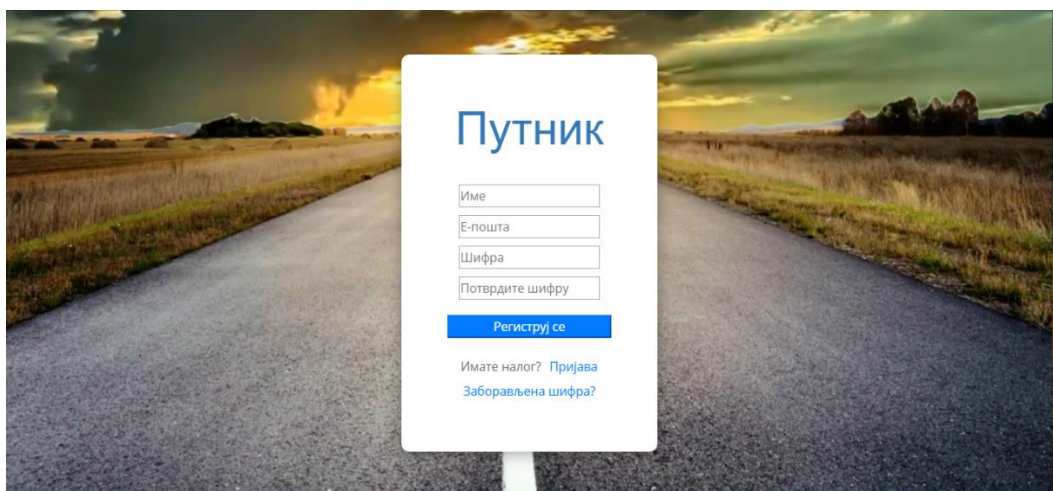
The following is part of the user guide for some of the most commonly used features:

New member registration

After loading the main, initial application window at the bottom of the screen there is a link that says Registration. You need to click on this link and wait for a new window to open.



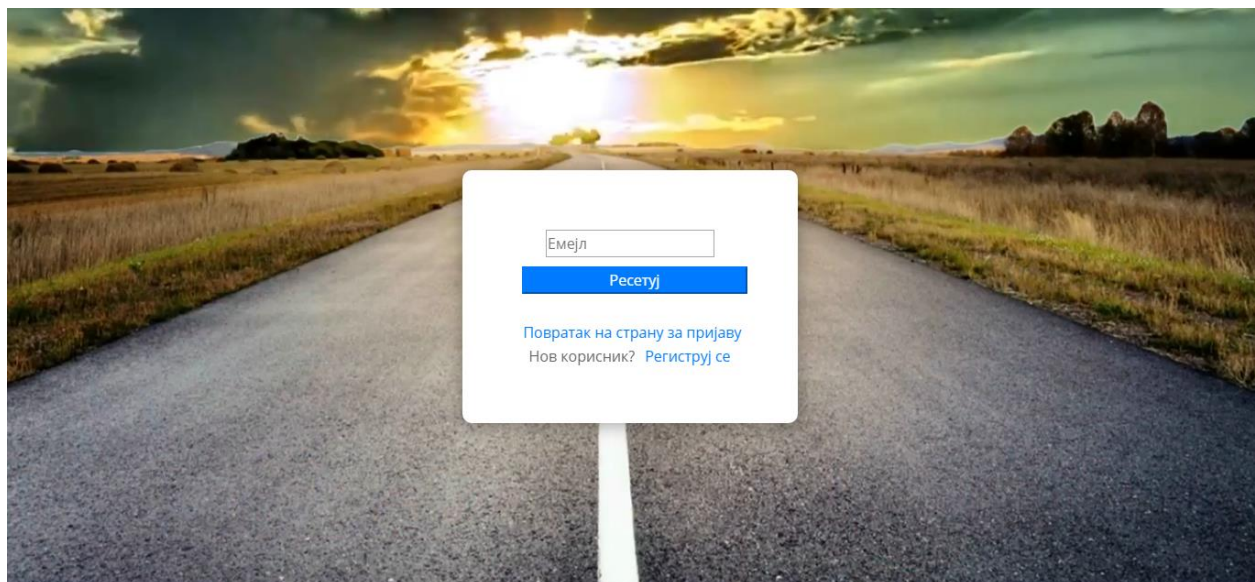
The new window contains the fields to be filled in: name, email, password and password verification. Restrictions related to these fields are indicated by the Code field where the minimum password length can be four characters. In addition, the correct email address form must be entered in the Email field.

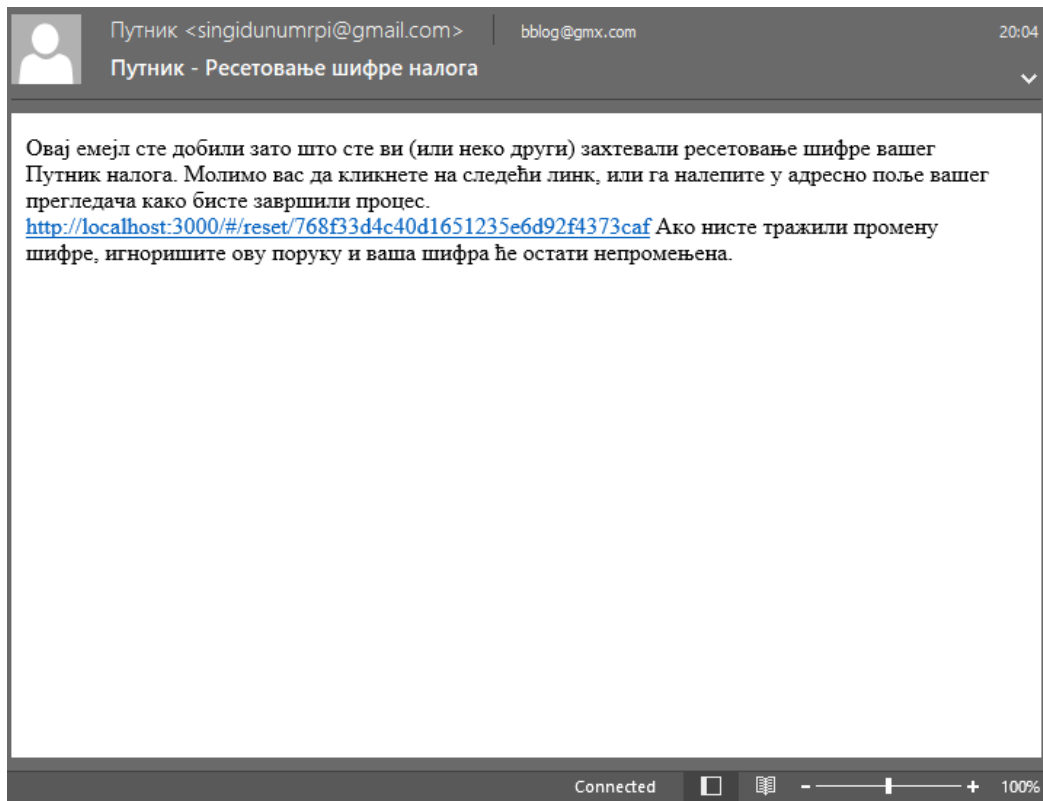


After completing the form correctly, you should click the Register field. A message will appear stating that a message has been sent to the user's email and that the verification link should be clicked. After that, the account is registered and the user can start using the application by logging in with his e-mail and password he entered during registration.

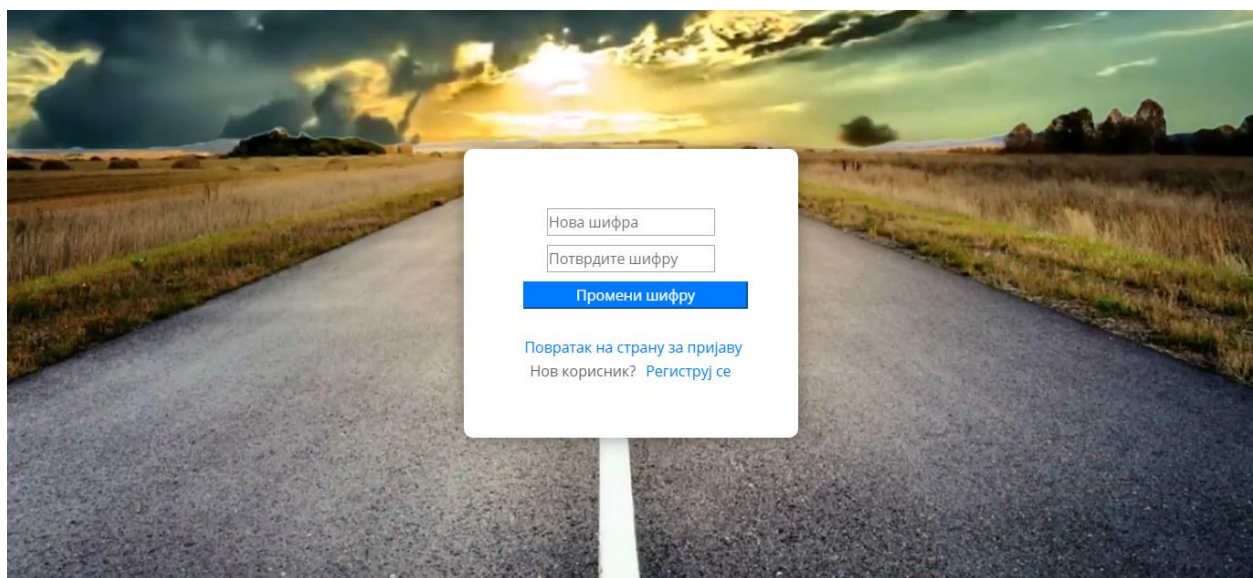
Forgotten password

If a password is forgotten, there is a way to reset the old one and enter a new one. Needed is to click Forgot Password? and enter the email address used when registering. After confirming with the Reset button, the system generates a message which it sends to the given e-mail within which there is a link that should be clicked.





After confirmation, a window opens where you enter a new code and confirm the new code. The process ends by selecting the Change Password button.



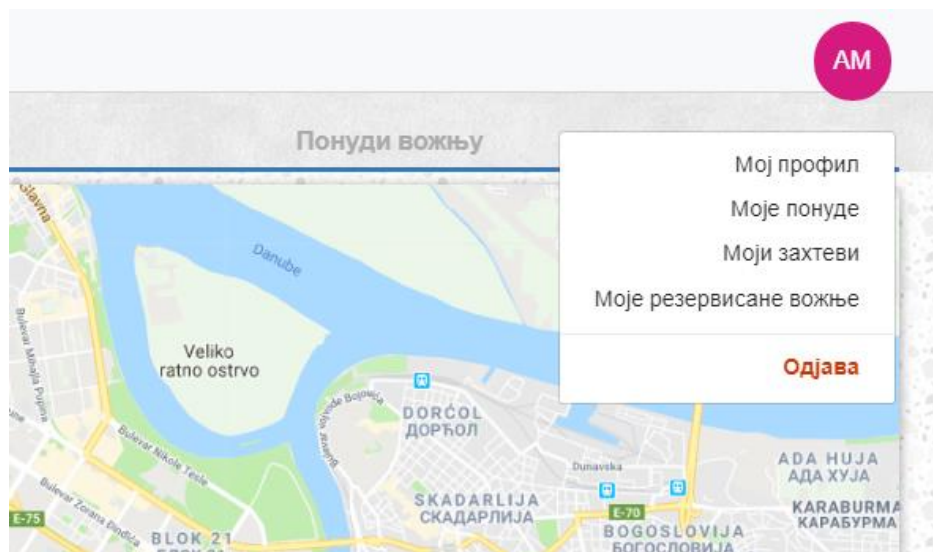
The procedure for offering a ride will be shown in the user guide. With this precedent, registered users get the opportunity to announce the planned ride. After completing all phases, and entering the relevant information, the bids are made visible to all users who can start booking

the offered route. In the main window for the carrier, in the lower left corner, there is a button labeled Offer a ride.

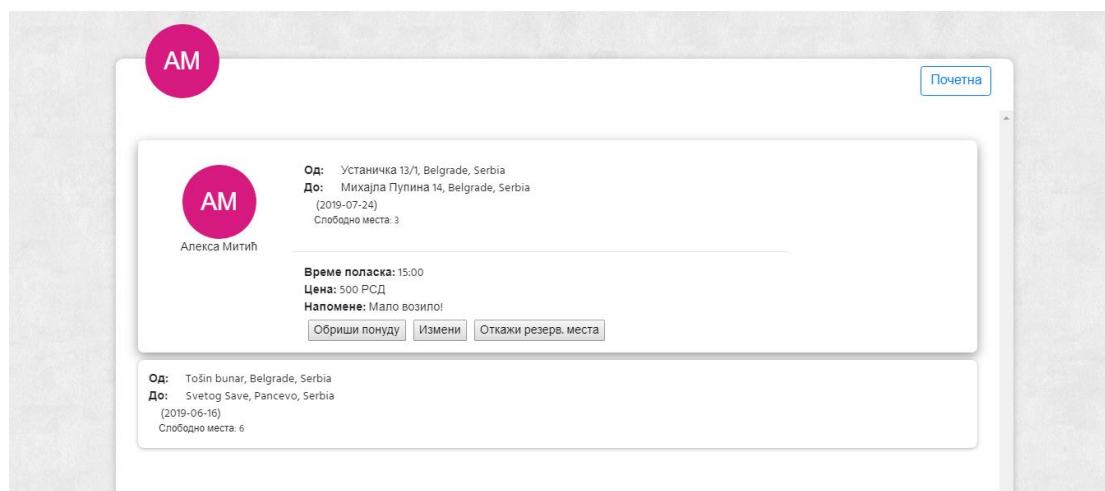
Immediately thereafter, a window opens in which all the necessary information must be entered in order for the ride offered to be valid and to be made available to other users of the application.

Data entry begins with the first blank field titled From: where the initial location is entered. the place and address from which the driver starts driving. The next field should be filled in with the destination, location where the ride ends, which again make up the place and address. In the Enter time field, enter the exact time when starting from the starting location. It is necessary to enter the date of the planned ride, where there is a restriction that prevents the user from entering incorrect data, the system prevents the date from the past to be entered. In accordance

with the size of the vehicle used for planned trip the user enters the number of vacancies that can be reserved by interested members. Specific or characteristic things that describe the ride offered may be included in the note. This may also be some useful information that may further interest users to decide to book a ride. In the price field is entered the requested amount for booking one place in the vehicle. Clicking the Save button completes the process of posting the ride, which at that point becomes valid.



Check the data of the publication of driving the user can perform through the user's order by clicking on the link My offers. Where you can read all the items you entered when you published the trip.



At any time the user can change the shape of the twill and change some of the data by selecting the change taster. In addition, it is possible to undo a full run-through and delete all of the published data by pressing the Delete offer.

4.1 Literature

- Tomašević, V., Application Software Development, Singidunum University, Belgrade, 2019.
- <https://developer.mozilla.org> the MDN documentation web, access 09.05.2019.
- <https://docs.mongodb.com> the documentation of MongoDB, access 10.05.2019.
- <https://mongoosejs.com/docs/api> documentation Mongoose, access to 10.05.2019.
- <https://reactjs.org/docs/getting-started.html>, React.JS documentation, accessed 10.05.2009.
- <https://nodejs.org/docs>, NOD.JS documentation, accessed 10.05.2009.