# Angular.js Workshop – Car System

Open the server directory, install all its dependencies and run it. It is listening on port 5000. The server uses in-memory database so if you need to restart it for some reason, all the data will be lost and you will need to populate it again.

## Create an Angular application and add authentication

Create an Angular application and prepare the initial project structure. Install Flux (or other similar framework) and prepare its components. Add authentication and make sure the register, login and logout functionalities work correctly. To register a user, you need to send a POST request to the server on ‘**/auth/signup**’ with ‘name’, ‘email’ and ‘password’ data (sent as JSON). To login a user, you need to send a POST request to the server on ‘**/auth/login**’ with ‘name’ and ‘password’ data (sent as JSON). You need to save the user token in your application state. Make sure you validate everything on the client application.

## Add statistics (10 points)

Show the total number of users and cars in the system on the home page. You need to make a GET request to ‘**/stats**’ in order to retrieve the data.

## Add creating of cars (10 points)

Add a form to create cars in the system. Each car has ‘make’ as string, ‘model’ as string, ‘year’ as number, ‘engine’ as string, ‘price’ as number, ‘image’ as string URL and optional ‘mileage’ as number. Make sure you validate everything on the client application. The data must be sent as POST request to the server on ‘**/cars/create/**’. This route is only for authenticated users so you need to send a header with `Authorization` name and value `bearer {*token*}` in order to pass the authentication checks.

## Add listing of cars (10 points)

Add a page where all cars are listed. Since the data from the server comes in pages of 10 cars, your page must have buttons to navigate between the pages. Try to validate the buttons when the page is no longer valid. You need to make a GET request to ‘**/cars/all**’ to receive an array of cars data. Optionally, you can pass a query string parameter ‘page’ to request more data, for example ‘**/cars/all?page=2**’. Link each cars to its details page. Don’t show every piece of information about the car on this page. Leave something for the details page. You may add this functionality on the home page.

## Searching for cars (10 points)

Add an option to search for cars on the listing page. Add a text input and a button. You need to make a GET request to ‘**/cars/all**’ with a query string parameter ‘search’ to retrieve the search results, for example ‘**/cars/all?search=BMW**’. Keep in mind that the ‘page’ and ‘search’ parameters can be combined. For example, the request ‘**/cars/all?page=3&search=BMW**’ will return the third page of the searched term ‘BMW’.

## Add car details (10 points)

Add a page where all car details are shown. You need to make a GET request to ‘**/cars/details/{*id*}**’ to retrieve information about the car with the provided id. This route is only for authenticated users so you need to send a header with `Authorization` name and value `bearer {*token*}` in order to pass the authentication checks. Make sure your Angular application redirects to the login page, if the user tries to open the car details page and she’s not logged in.

## Add option to add likes and reviews (15 points)

On the car details page add an option for the user to put a rating from 1 to 5 and optionally a comment for the current car. Each review has ‘rating’ as number between 1 and 5 and ‘comment’ as string data (sent as JSON). You need to make a POST request to ‘**/cars/details/{*id*}/reviews/create**’ in order to create a review. The id is the car id the user is reviewing. Additionally, add a like button, which makes a POST request to ‘**/cars/details/{*id*}/like**’. You don’t need to send any data, just update the total likes for the current car. The same user cannot like the same car twice. Make sure you validate everything on the client application. Both routes are only for authenticated users so you need to send a header with `Authorization` name and value `bearer {*token*}` in order to pass the authentication checks.

## Add listing of all reviews (10 points)

On the car details page, you need to list all reviews for the current car. You need to make a GET request to ‘**/cars/details/{*id*}/reviews**’ to receive an array of review data. This route is only for authenticated users so you need to send a header with `Authorization` name and value `bearer {*token*}` in order to pass the authentication checks.

## Add profile page (10 points)

Add a profile page where all cars created by the current user are shown. You need to make a GET request to ‘**/cars/mine/**’ to receive an array of cars. This request does not have paging so you do not have to implement it. The route is only for authenticated users so you need to send a header with `Authorization` name and value `bearer {*token*}` in order to pass the authentication checks.

## Add option to delete cars (15 points)

On the profile page, for each car add a delete button to allow the user to remove a previously added car from the system. To delete a car, you need to make a POST request to ‘**/cars/delete/{*id*}**’. This route is only for authenticated users so you need to send a header with `Authorization` name and value `bearer {*token*}` in order to pass the authentication checks.