JS Applications Exam – CarTube

You are assigned to implement a **Web application** (SPA) using JavaScript. The application should dynamically display content, based on user interaction and support user profiles and CRUD operations, using a REST service.

1. Overview

Implement a front-end app (SPA) for viewing and managing **car listings**. The application allows visitors to browse through different car ads. Users may **register** with a **username** and **password**, which allows them to **create** their own ads. Ad authors can also **edit** or **delete** their own publications at any time.

2. Technical Details

You are provided with the following resources:

- **Project scaffold**: A **package.json** file, containing a list of common dependencies. You may change the included libraries to your preference. The sections **devDependencies** and **scripts** of the file are used by the automated testing suite, altering them may result in incorrect test operation.

  To **initialize** the project, execute the command **`npm install`** via the command-line terminal.

- **HTML and CSS files**: All views (pages) of the application, including **sample** user-generated **content**, are included in the file **index.html**, which links to CSS and other static files. **Each view is in a separate section** of the file, which can be identified by a **unique class name or id** attribute. Your application may use any preferred method (such as a **templating library** or manual visibility settings) to display only the selected view and to **navigate** between views upon user interaction.
- **Local REST service**: A special server, which contains **sample data** and supports **user registration** and **CRUD operations** via REST requests is included with the project. Each section of this document (where applicable) includes details about the necessary **REST endpoints**, to which **requests** must be sent, and the **shape** of the expected **request body**.

  For **more information** on how to use the included server, see **Appendix A: Using the Local REST Service** at the end of this document.

- **Automated tests**: A complete test suite is included, which can be used to test the correctness of your solution. **Your work will be assessed, based on these tests.**

  For **more information** on how to run the tests, see **Appendix B: Running the Test Suite** at the end of this document.

Note: When creating HTML Elements and displaying them on the page, **adhere as close as possible to the provided HTML** samples. Changing the structure of the document may **prevent the tests** from running correctly, which will **adversely affect your assessment grade**. You may **add attributes** (such as **class** and **dataset**) to any HTML Element, as well as **change "href"** attributes on links and add/change the **method** and **action** attributes of HTML Forms, to facilitate the correct operation of a routing library or another method of abstraction. You may also add hidden elements to help you implement certain parts of the application requirements.

3. Application Requirements

Navigation Bar (5 pts)

Navigation links should correctly change the current page (view). **Guests** (un-authenticated visitors) can see the links to the **All Listings** page, as well as the links to the **Login** and **Register** pages. The logged-in user navbar should contain the links to **All Listings** page, the **Create** page and a link for the **Logout** action.

Guest navigation example:

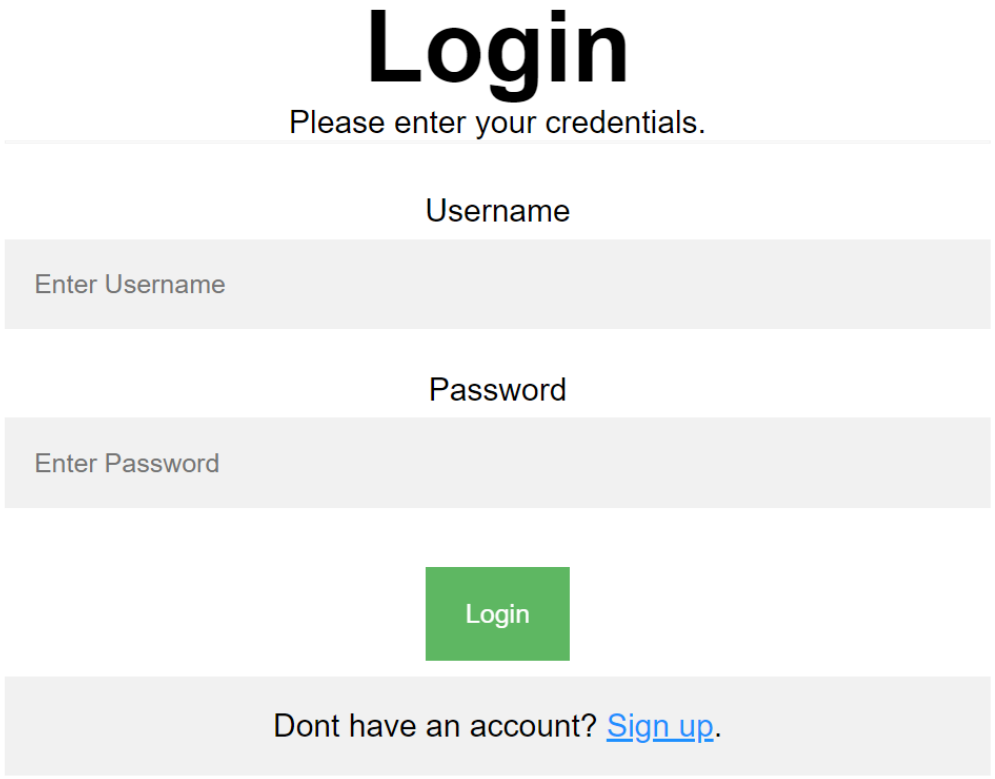
User navigation example:

Login User (5 pts)

The **included REST service** comes with the following **premade** user accounts, which you may use for development:

```
{ "username": "Peter", "password": "123456" }
{ "username": "John", "password": "123456" }
```

The **Login** page contains a form for existing user authentication. By providing a **username** and **password,** the app should login a user in the system if there are no empty **fields**.



Send the following **request** to perform login:

```
Method: POST
URL: /users/login
```

Required **headers** are described in the documentation. The service expects a body with the following shape:

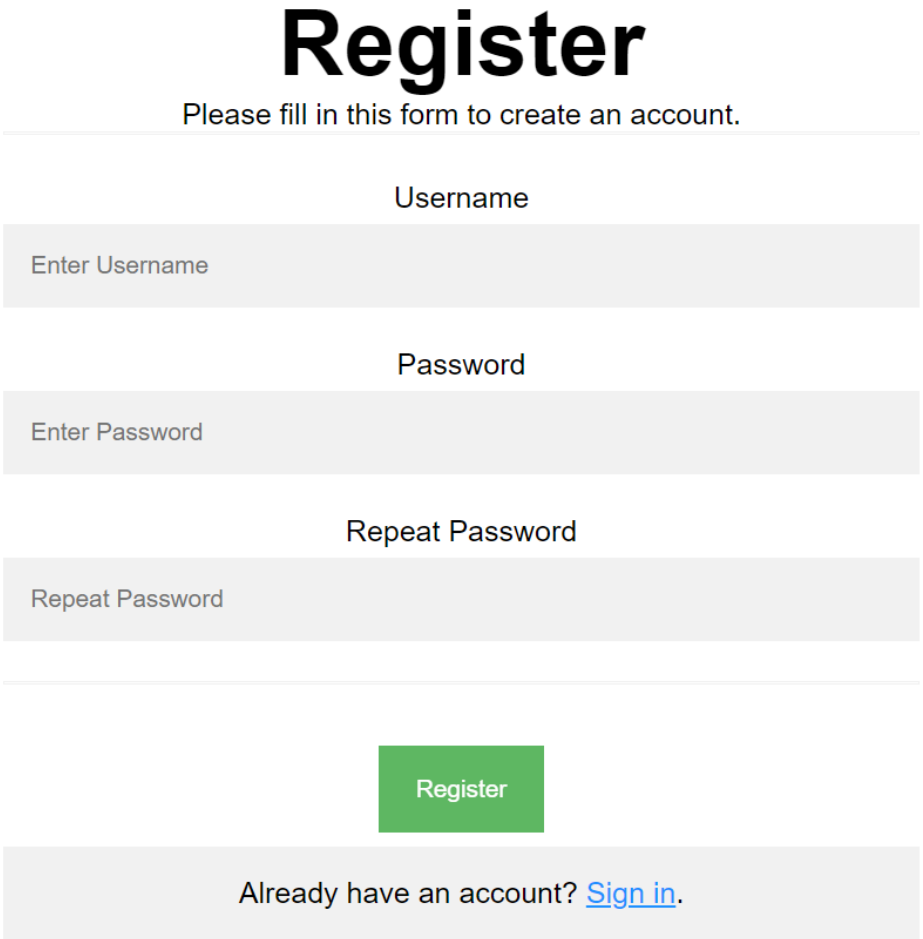
```
{
  username,
  password
}
```

Upon success, the **REST service** will return information about the existing user along with a property **accessToken**, which contains the **session token** for the user – you need to store this information using **sessionStorage** or **localStorage,** in order to be able to perform authenticated requests.

If the login was successful, **redirect** the user to the **All Listings** page. If there is an error, display an appropriate error message, using a system dialog (`window.alert`).

Register User (10 pts)

The **Register** page contains a form for new user registration. By providing a **username** and **password,** the app should register a new user in the system if there are no empty **fields**.



Send the following **request** to perform registration:

```
Method: POST
URL: /users/register
```

Required **headers** are described in the documentation. The service expects a body with the following shape:

```
{
  username,
  password
}
```

Upon success, the **REST service** will return the newly created object with an automatically generated **_id** and a property **accessToken**, which contains the **session token** for the user – you need to store this information using **sessionStorage** or **localStorage,** in order to be able to perform authenticated requests.

If the registration was successful, **redirect** the user to the **All Listings** page. If there is an error, or the **validations** don't pass, display an appropriate error message, using a system dialog (`window.alert`).

Logout (5 pts)

The logout action is available to logged-in users. Send the following **request** to perform logout:

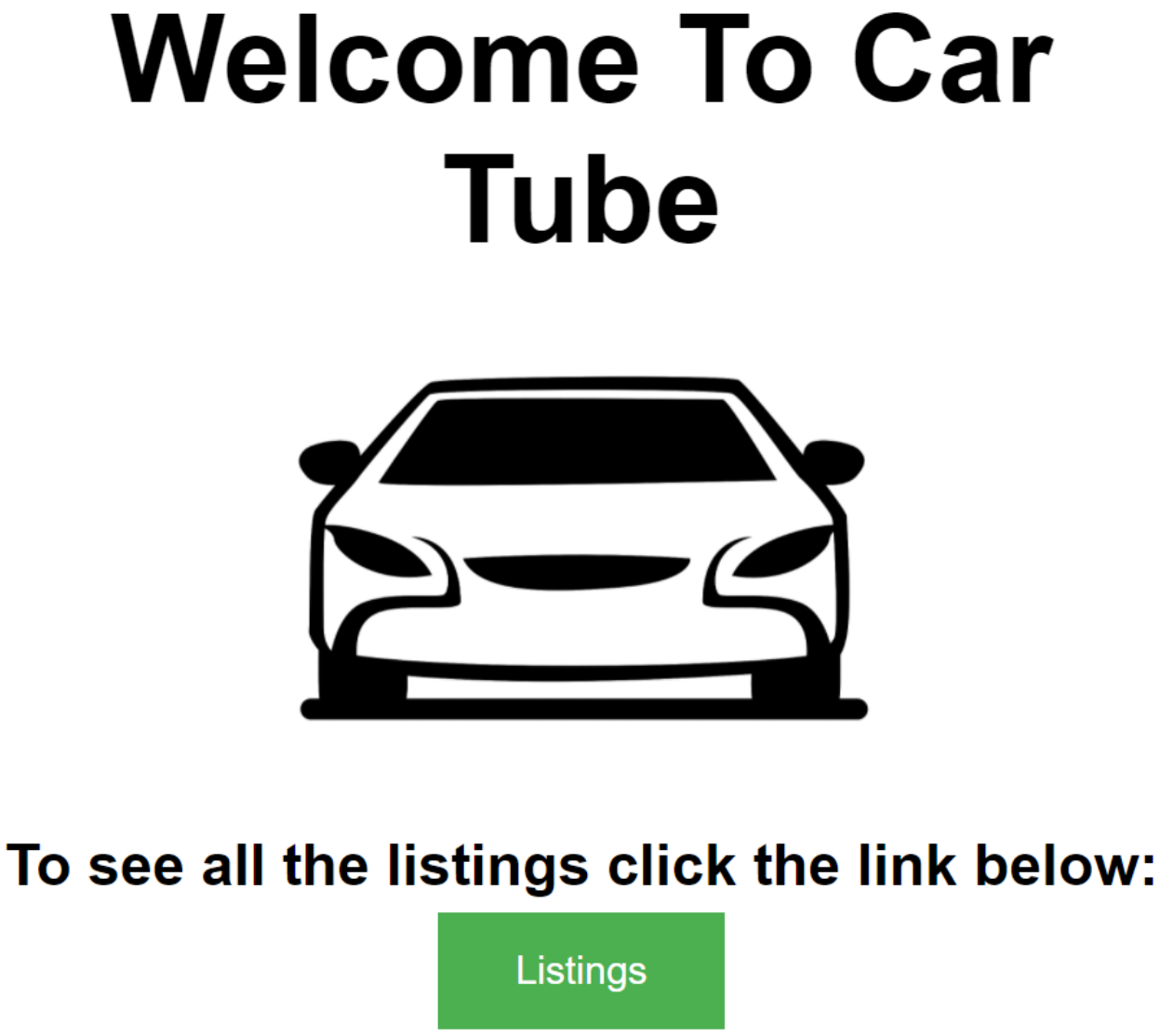
```
Method: GET
URL: /users/logout
```

Required **headers** are described in the documentation. Upon success, the **REST service** will return an **empty response**. Clear any session information you've stored in browser storage.

If the logout was successful, **redirect** the user to the **Home** page.

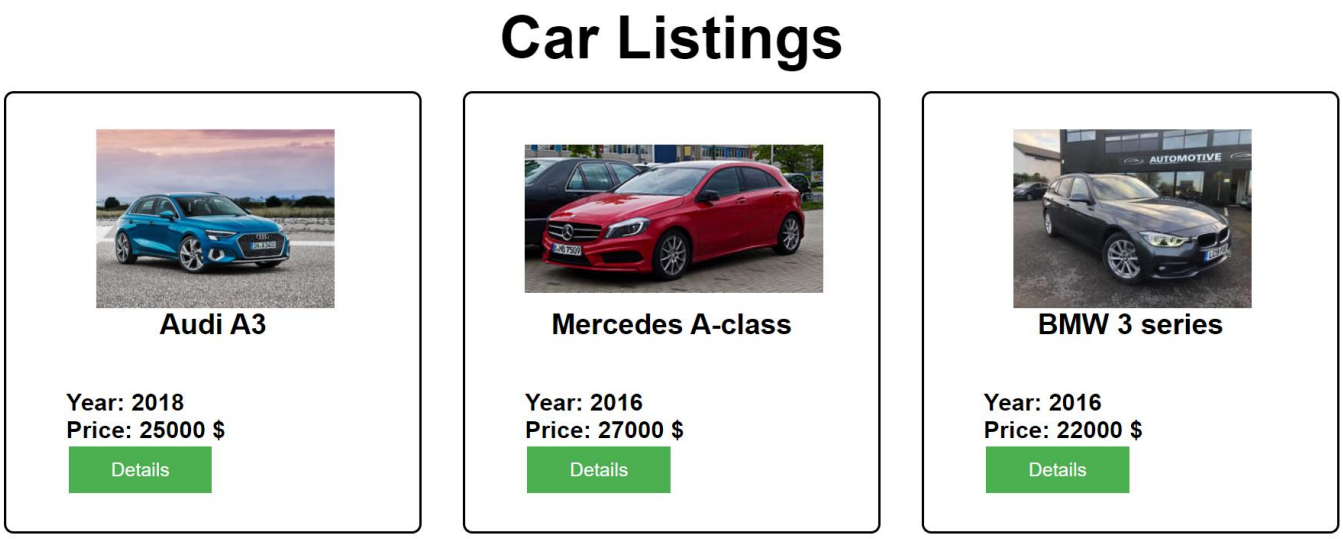
Home Page (5 pts)

All users should be welcomed by the **Home page**, where they could redirect to the **Listings** view.



All Listings Page (10 pts)

This page displays a list of all listings in the system. Clicking on the details button in the cards leads to the details page for the selected listing.



If there are no listings, the following view should be displayed:

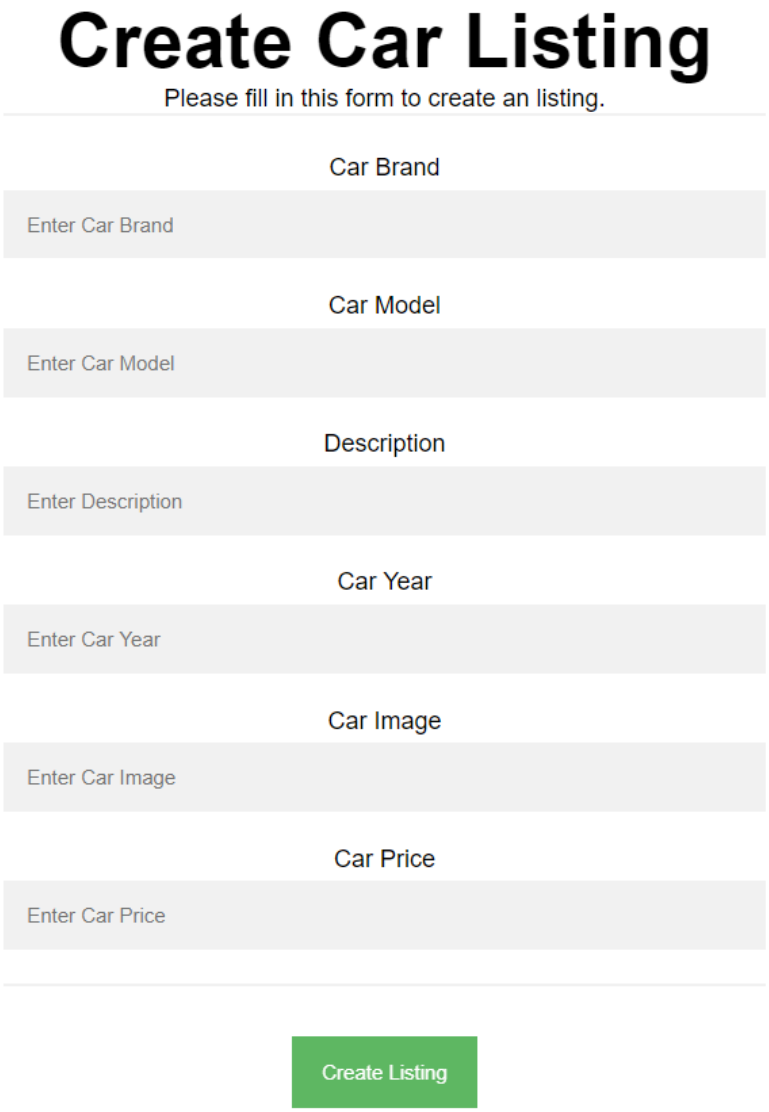
Send the following **request** to read the list of ads:

```
Method: GET
URL: /data/cars?sortBy=_createdOn%20desc
```

Required **headers** are described in the documentation. The service will return an array of listings.

Create Car Listing (15 pts)

The Create page is available to logged-in users. It contains a form for creating new listings. Check if all the fields are filled before you send the request.



To create a listing, send the following **request**:

```
Method: POST
URL: /data/cars
```

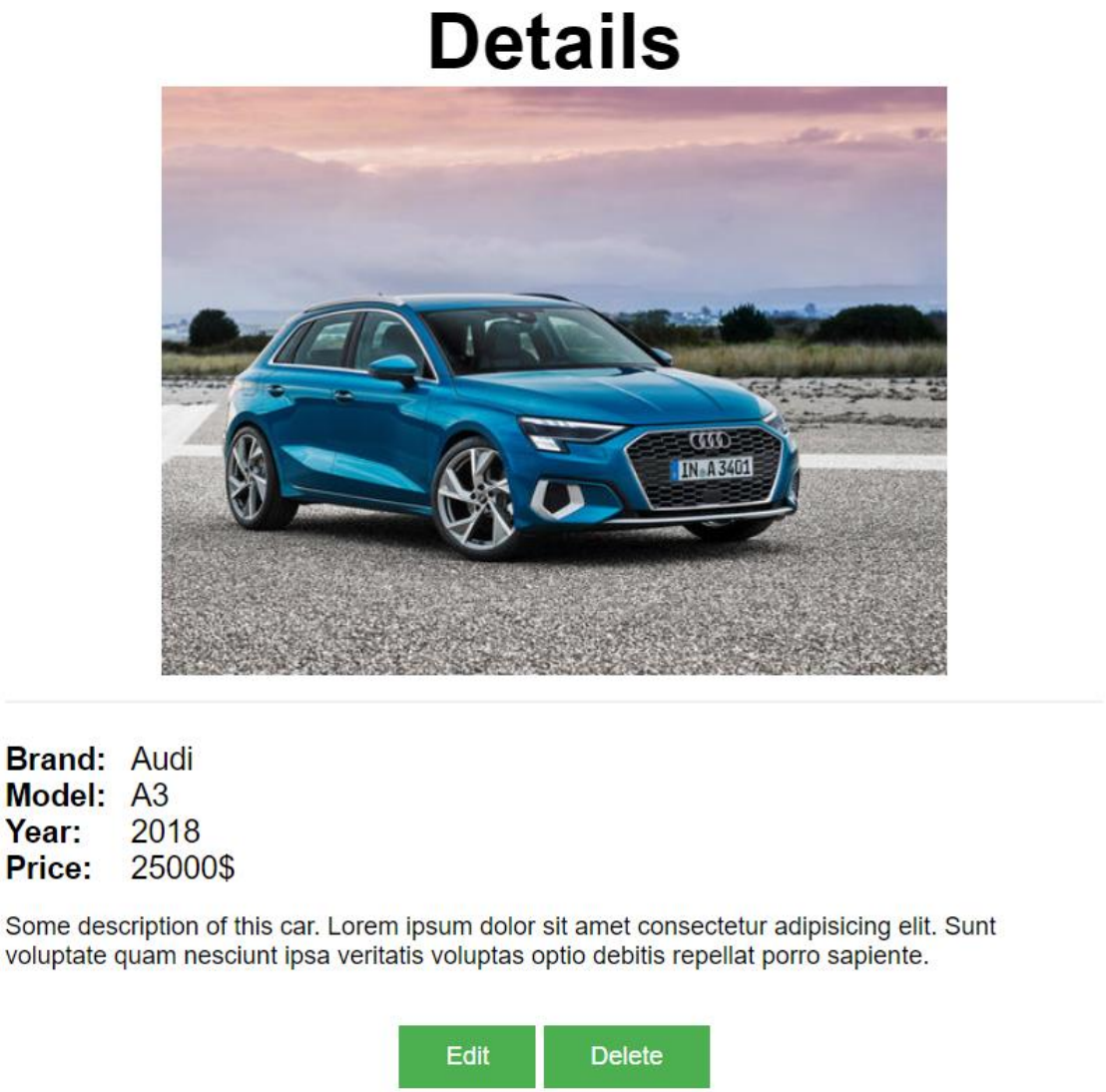
Required **headers** are described in the documentation. The service expects a body with the following shape:

```
{
  brand,
  model,
  description,
  year,
  imageUrl,
  price
}
```

Required **headers** are described in the documentation. The values of **`year`** and **`price`** must be positive numbers. The service will return the newly created record. Upon success, **redirect** the user to the **All Listings** page.

Details (10 pts)

All users should be able to **view details** about listings. Clicking the **Details** link in the **car ad** should **display** the **Details** page. If the currently logged-in user is the creator of the listing, the **Edit** and **Delete** buttons should be displayed.



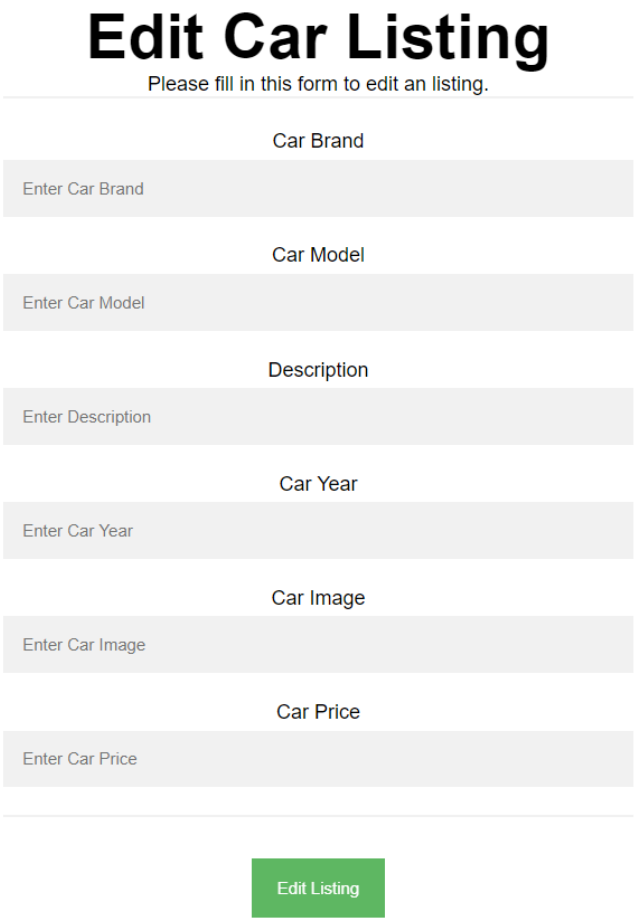
Send the following **request** to read a single listing:

```
Method: GET
URL: /data/cars/:id
```

Where ***:id*** is the **id** of the desired listing. Required **headers** are described in the documentation. The service will return a single object.

Edit Listing (15 pts)

The Edit page is available to logged-in users and it allows author to **edit** their **own** listings. Clicking the **Edit** link of a particular listing on the **Details** page should display the **Edit** page. It contains a form with input fields for all relevant properties. Check if all the fields are filled before you send the request.



To edit an listing, send the following **request**:

```
Method: PUT
URL: /data/cars/:id
```

Where ***:id*** is the **id** of the desired listing.

The service expects a body with the following shape:

```
{
  brand,
  model,
  description,
  year,
  imageUrl,
  price
}
```

Required **headers** are described in the documentation. The values of **`year`** and **`price`** must be positive numbers. The service will return the modified record. Note that **PUT** requests **do not** merge properties and will instead **replace** the entire record. Upon success, **redirect** the user to the **Details** page.

Delete Listing (10 pts)

The delete action is available to **logged-in** users, for listing they have created. When the author clicks on the Delete action on any of their listing, a confirmation dialog should be displayed, and upon confirming this dialog, the listing should be **deleted** from the system.

To delete a listing, send the following **request**:

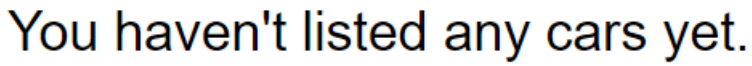
```
Method: DELETE
URL: /data/cars/:id
```

Where ***:id*** is the **id** of the desired listing. Required **headers** are described in the documentation. The service will return an object, containing the deletion time. Upon success, **redirect** the user to the **All Listings** page.

My Listings (10 pts)

This page displays a list of all listings made by the current user.

If there are no listings, the following view should be displayed:



Send the following **request** to read the list of ads:

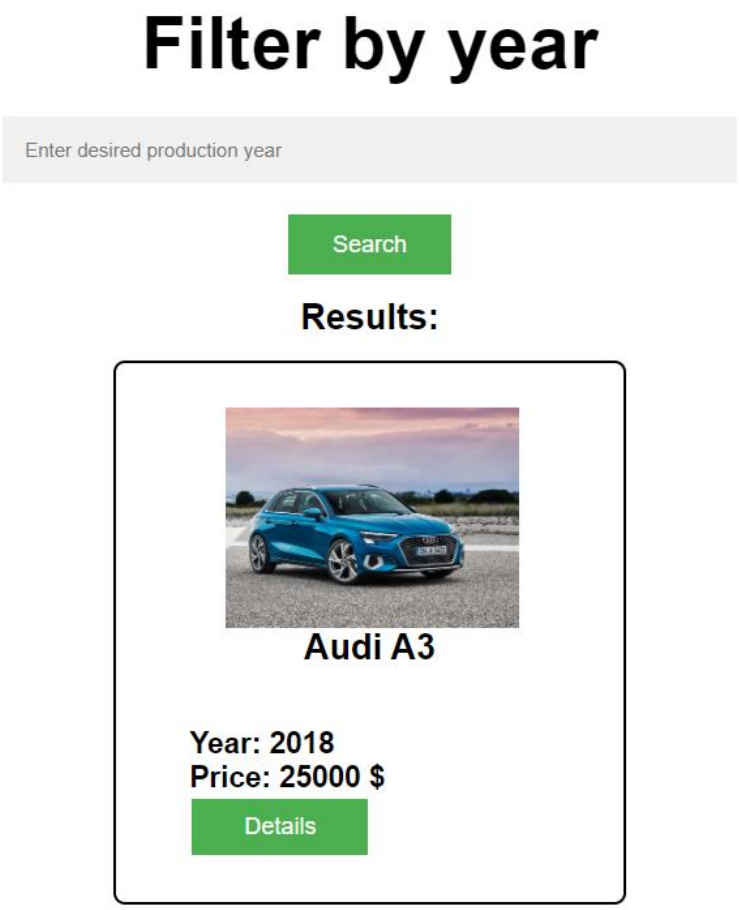
```
Method: GET
URL: /data/cars?where=_ownerId%3D%22{userId}%22&sortBy=_createdOn%20desc
```

Where ***{userId}*** is the id of the currently logged-in user.

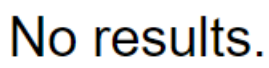
Required **headers** are described in the documentation. The service will return an array of listings.

BONUS: Search (5 pts)

The Search page allows users to filter listings by their production year. It contains an input field and, upon submitting a query, a list of all matching listings.



If there are no results, the following view should be displayed:



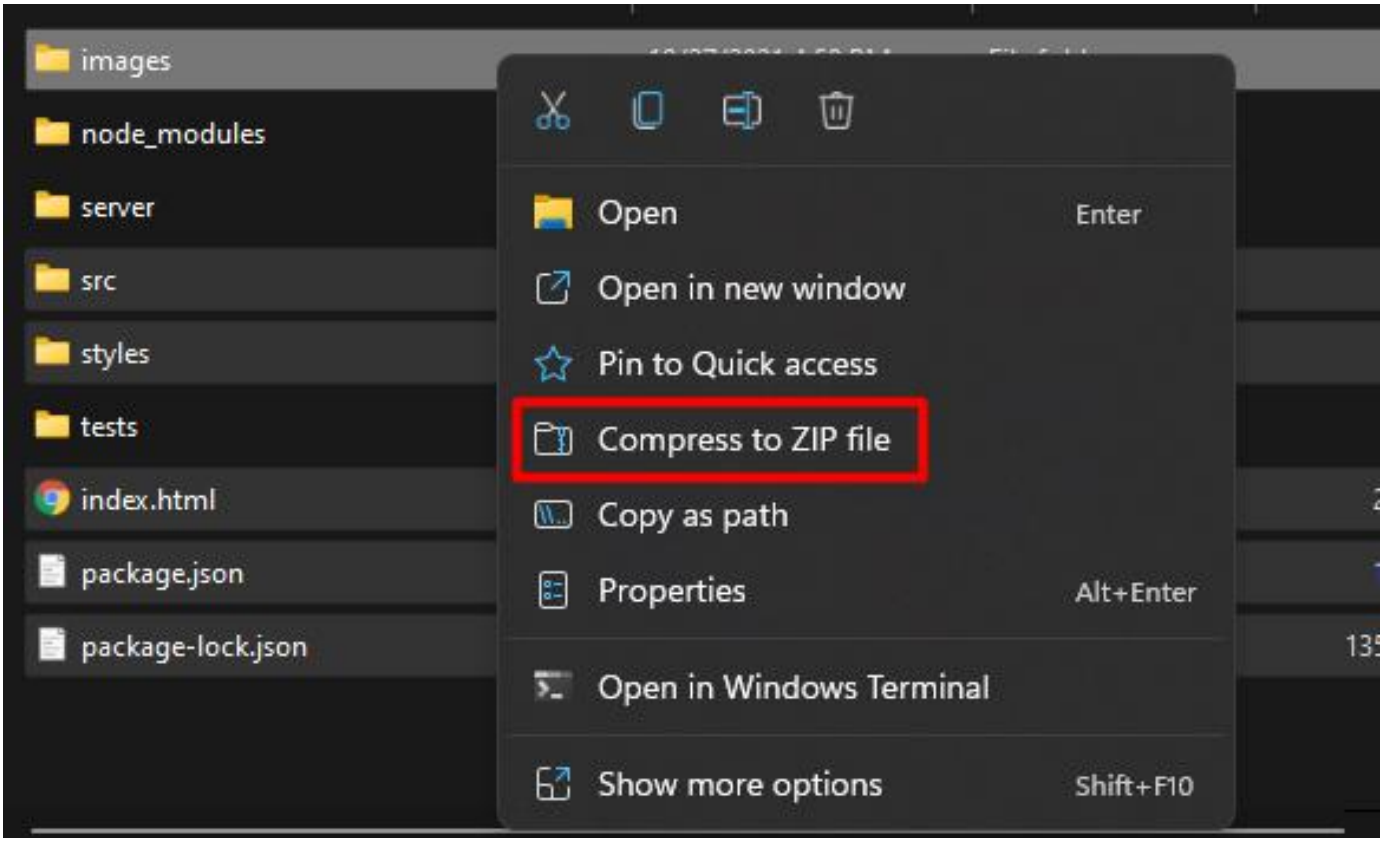
Send the following **request** to read a filtered list of ads by their production year:

```
Method: GET
URL: /data/cars?where=year%3D{query}
```

Where ***{query}*** is the search query that the user has entered in the input field. Required **headers** are described in the documentation. The service will return an array of listings. If there are no matches, display the text **"No matching listings"** instead.

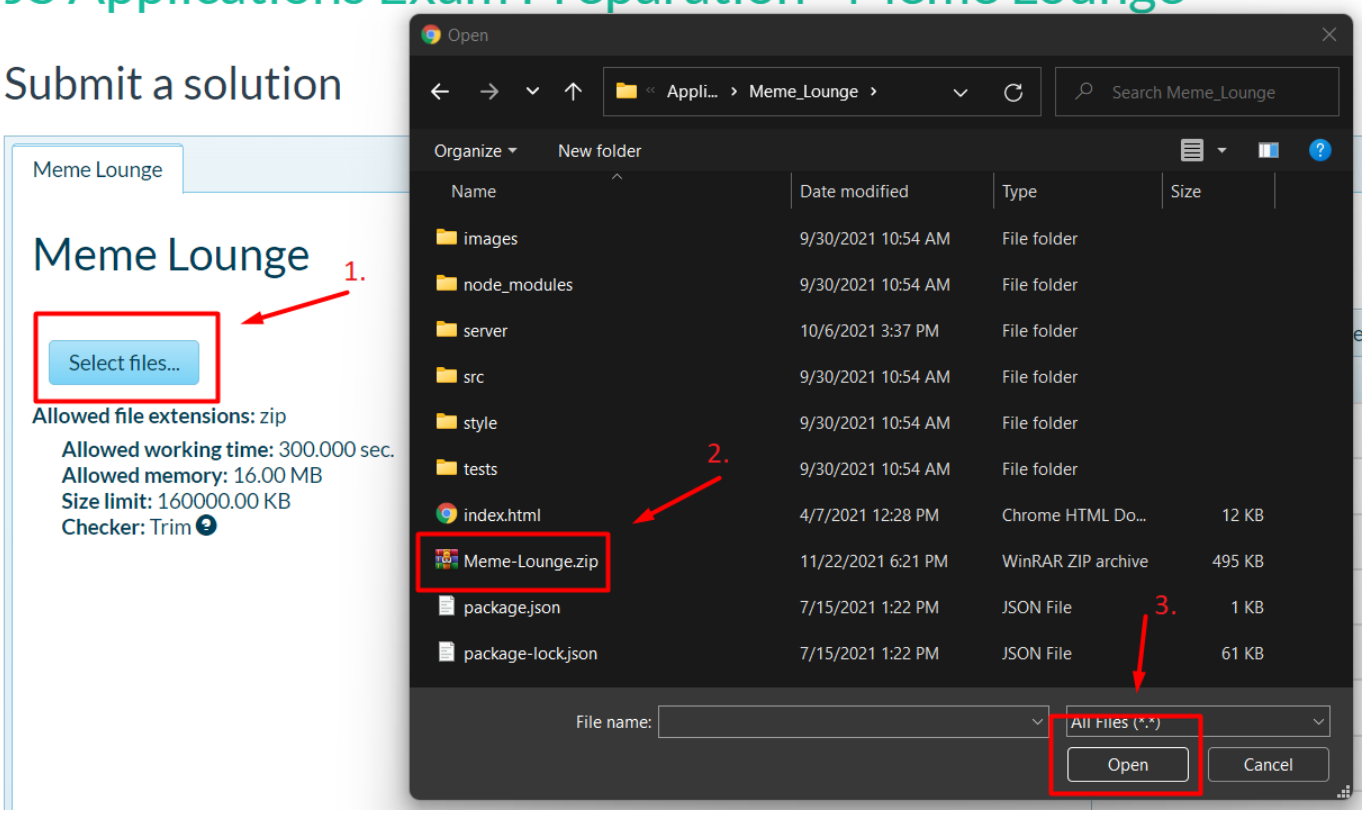
4. Submitting Your Solution

Place in a **ZIP** file your project folder. Exclude the **`node_modules`**, **`server`** and **`tests`** folders. Upload the archive to Judge.



JS Applications Exam Preparation - Meme Lounge

Submit a solution



It will take several minutes for Judge to process your solution!