

Front-End Technologies Basics Exam Preparation II




1. DOM Manipulation

Use the provided skeleton to solve this problem.

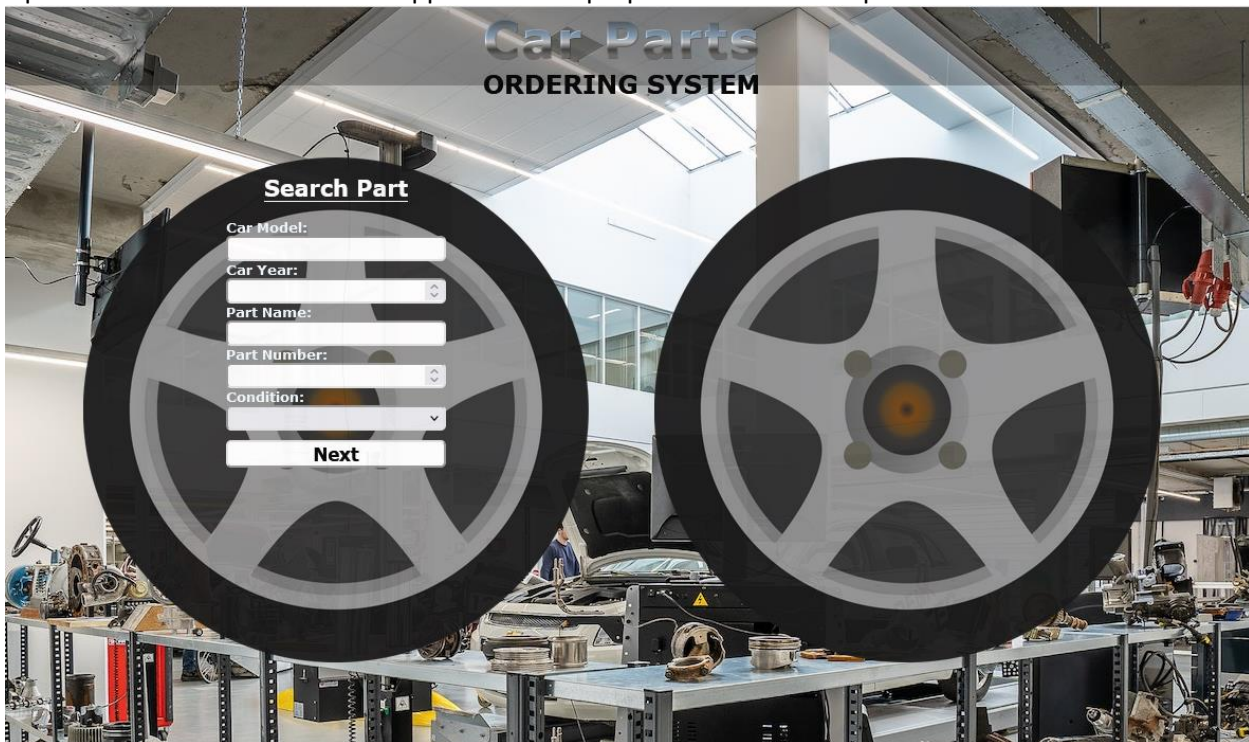
Write the missing functionality of this user interface.

Application structure

You are provided with an HTML application - **Car Parts** with the following structure:

 style	11.7.2024 r. 16:31	File folder	
 app.js	13.11.2024 r. 10:03	JS File	3 KB
 index.html	13.11.2024 r. 9:52	Firefox HTML Docu...	3 KB

Open the index.html to start the application. It's purpose is to order car parts and looks like this:



Look at the html structure in the index.html file. You will notice that there are 2 hidden elements:

```
<div id="part-info" style="display: none;">
```

```
<div id="confirm-order" style="display: none;">
```

The part info is supposed to be displayed after the Next button is clicked. The confirm order is supposed to be displayed after the order is finalized.

Note that all form fields have a unique ids you can use:

```

<label for="car-model">Car Model:</label>
<input type="text" id="car-model" name="car-model">

<label for="car-year">Car Year:</label>
<input type="number" id="car-year" name="car-year" pattern="\d{4}">

<label for="part-name">Part Name:</label>
<input type="text" id="part-name" name="part-name">

<label for="part-number">Part Number:</label>
<input type="number" id="part-number" name="part-number" >

<label for="condition">Condition:</label>
<select name="condition" id="condition">
  <option value=""></option>
  <option value="New">New</option>
  <option value="Used">Used</option>
</select>
<button id="next-btn" type="submit">Next</button>

```

In the Confirm order element there are elements with unique ids you can use to fill in the information required:

```

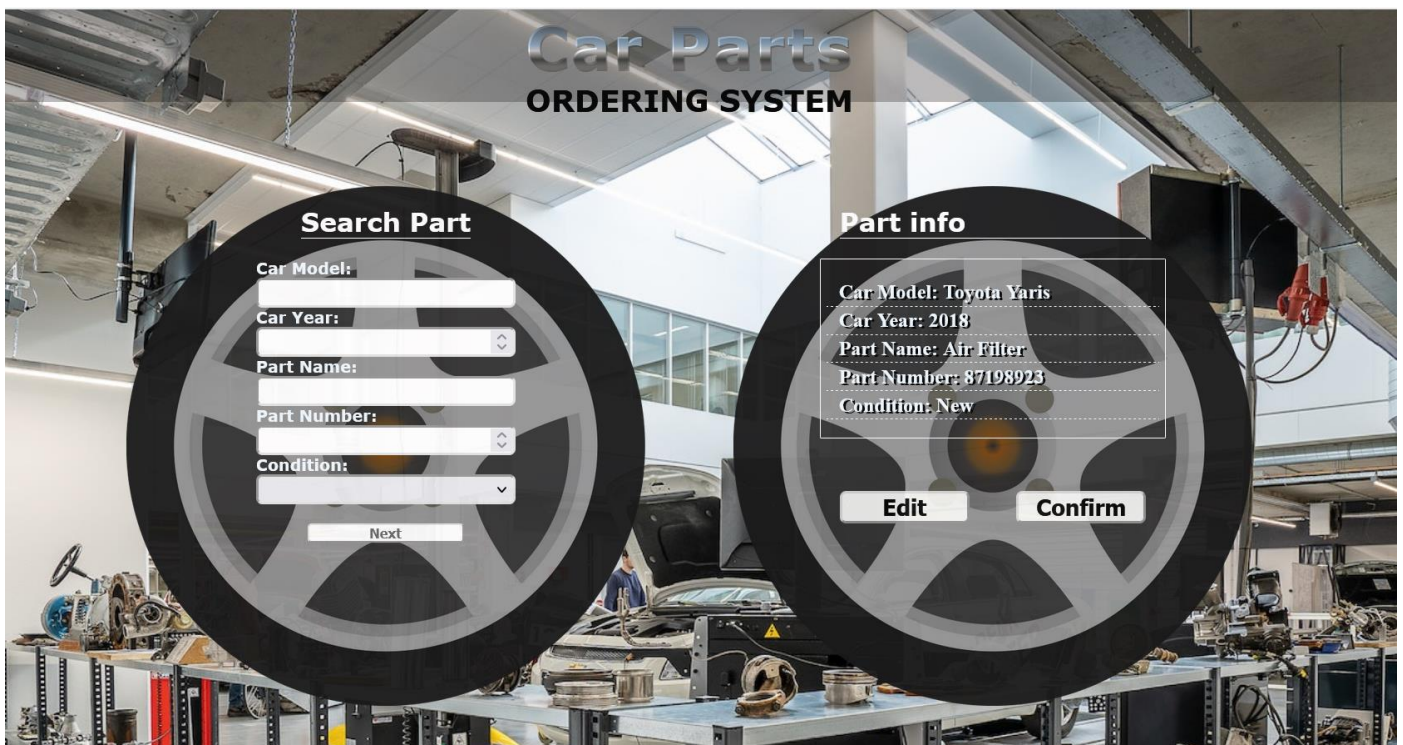
<p>Car Model: <span id="info-car-model"></span></p>
<p>Car Year: <span id="info-car-year"></span></p>
<p>Part Name: <span id="info-part-name"></span></p>
<p>Part Number: <span id="info-part-number"></span></p>
<p>Condition: <span id="info-condition"></span></p>

```

Your Task

Write the missing JavaScript code in the app.js file to make the Car Parts work as expected:

1.1. Next button functionality

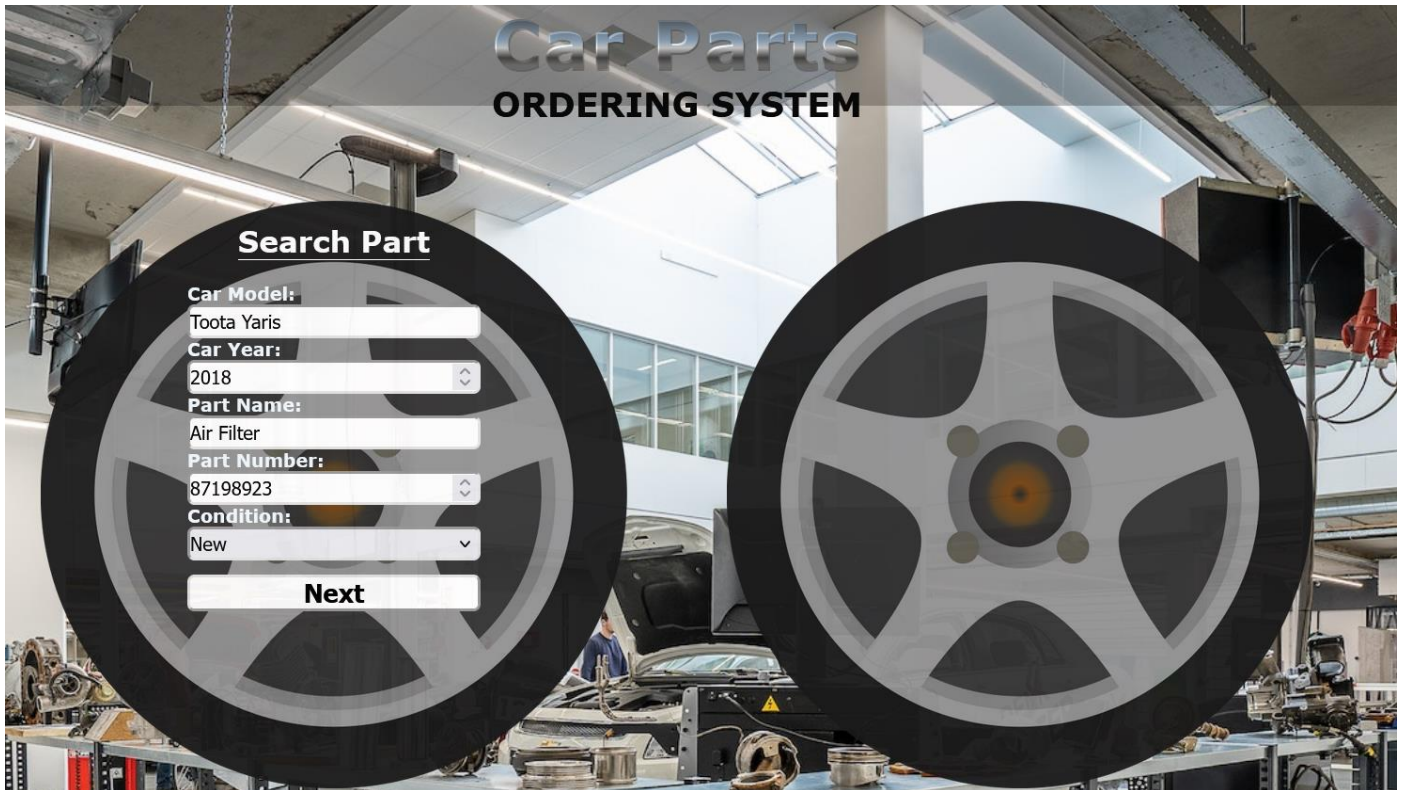


When the [Next] button is clicked, you need to validate that all fields are filled. You must also validate that the Car Year is between 1990 and the current year. If this is not the case, do not proceed. If validation is successful, the following functionality needs to be implemented:

- The information about the ordered part is filled in the part info HTML elements – car model, car year, part name, part number, condition
- The part info element must be displayed
- The Next button must be disabled
- The values of the fields in the form must be cleared

1.2. Edit Part

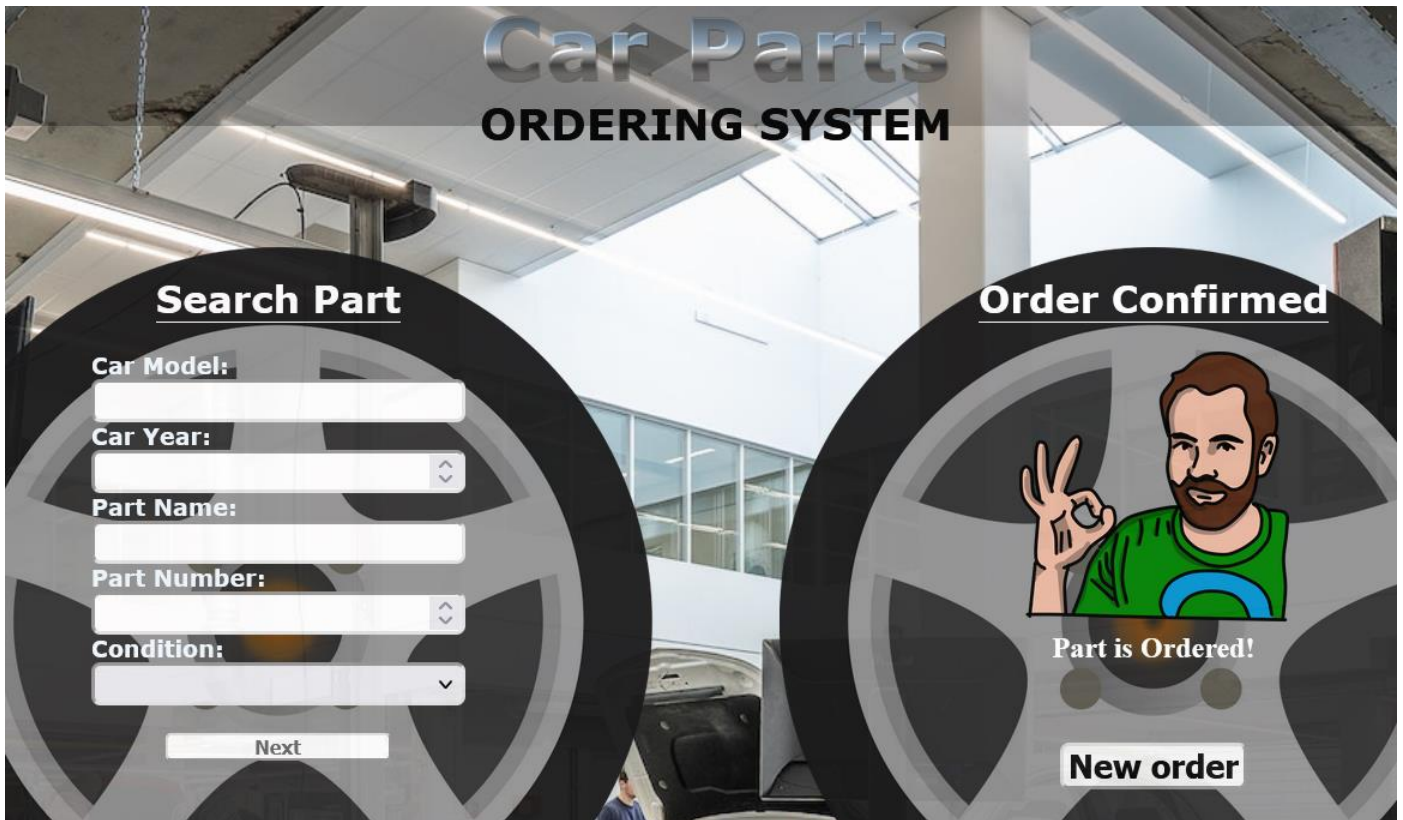
When the **[Edit]** button is clicked, all of the information is loaded in to the input fields from step, while the **[Next]** button is enabled again.



The part info element must be hidden.

1.3. Confirm

When the **[Confirm]** button is clicked, you must hide the part info element and show the confirm order element.



1.4. New order

When the [New order] button is clicked, you must **hide the confirm order element** and enable the [Next] button.

2. JS Application Testing

You have been given a JavaScript application. All the necessary setup is complete, allowing you to start writing your tests. Your goal is to write end-to-end (Front-End) tests using Playwright.

2.1. App Introduction

The **application for testing** is called "Pet Care". It's a user-friendly application offering functionalities and permissions based on user login status:

- 1) For **guest user**:
 - **Home Page**: View a brief introduction to the app.
 - **Dashboard Page**: See all pet postcards.
 - **Register/Login page**: Option to register a new account or log in.
- 2) For **Logged-In user**:
 - **Home Page**: View a brief introduction to the app.
 - **Create Postcard Page**: Ability to create new postcard.
 - **Dashboard Page**: See all created pet postcards (both their own and others').
 - **Logout**: Option to log out of the app.
- 3) **Postcard Details Functionality**:
 - **Detail View**: Only logged-in users can see detailed information about a pet postcard.
 - **Edit/Delete Buttons**: Only the owner of a postcard can see and use the Edit and Delete buttons in the detail view to modify or remove the postcard.
- 4) **Navigation bar**: Provides easy access to application functionalities based on your login status (logged in or guest).

Also, there will be seeded data for 4 pet postcards, that will always be loaded when you start the application.

2.2. Instructions

A folder named "tests" is prepared for you. There is a single file named "e2e.test.js". In it, you must write your Front-End tests with the Playwright framework.

As for execution of the tests, **you need to start the back-end server of the application and the HTTP server.** Everything is configured for you, so you need just to **execute three commands in two Terminals:**

- "npm install" (or "npm init -y") – to install the dependencies for your app.
- "npm run server" – to start the application back-end server.
- "npm start" – to start the HTTP server.
- "npm test" – to run Playwright tests.

Note: You will need to use a **third Terminal window for the execution of Playwright tests.**

2.3. Front-End Testing with Playwright

You are provided with predefined configurations in the e2e.test.js file:

- Needed imports for Playwright:

```
const { test, describe, beforeEach, afterEach, beforeAll, afterAll, expect } = require('@playwright/test');
const { chromium } = require('playwright');
```

- Predefined variables that you can use:

```
const host = 'http://localhost:3000';

let browser;
let context;
let page;

let user = {
  email : "",
  password : "123456",
  confirmPass : "123456",
};

let petName = "";
```

- Before and after test configurations:

```
beforeAll(async () => {
  browser = await chromium.launch();
});

afterAll(async () => {
  await browser.close();
});

beforeEach(async () => {
  context = await browser.newContext();
  page = await context.newPage();
});

afterEach(async () => {
  await page.close();
  await context.close();
});
```

- Test suits:

```
describe("authentication", () => {
  });

describe("navbar", () => {
  });

describe("CRUD", () => {
  });
```

Use them and write the following e2e tests with Playwright for the "Pet Care" application:

2.3.1. Registration with Valid Data (Authentication Functionality)

1. Create a test scope.
2. Go to <http://localhost:3000>
3. Locate and click on the Register button.
4. Wait for the register form to load.
5. Create a unique email value.
6. Locate and fill the input field for email.
7. Locate and fill the input field for password.
8. Locate and fill the input field for confirm password.
9. Press the submit button.
10. Assert that you are redirected to the home page and the Logout button is visible.

Hint: Use the predefined user object to hold and reuse user data.

2.3.2. Login with Valid Data (Authentication Functionality)

1. Create a test scope.
2. Go to <http://localhost:3000>
3. Locate and click on the Login button.
4. Wait for the login form to load.
5. Locate and fill the input field for email.
6. Locate and fill the input field for password.
7. Press the submit button.
8. Assert that you are redirected to the home page and the Logout button is visible.

2.3.3. Logout from the Application (Authentication Functionality)

1. Create a test scope.
2. Go to <http://localhost:3000>
3. Log in to the application.
4. Click the Logout button.
5. Assert the Login button is visible.
6. Assert that the URL is for home page.

2.3.4. Navigation for Logged-In User Testing

1. Create a test scope.
2. Go to <http://localhost:3000>

3. Log in to the application.
4. Assert that "Home", "Dashboard", "Create Postcard" and "Logout" buttons are visible, and "Login" and "Register" buttons are hidden.

2.3.5. Navigation for Guest User Testing

1. Create a test scope.
2. Go to <http://localhost:3000>
3. Assert that "Home", "Dashboard", "Login" and "Register" buttons are visible, and "Create Postcard" and "Logout" buttons are hidden.

2.3.6. Create a Postcard Testing (CRUD Functionality)

1. Create a test scope.
2. Go to <http://localhost:3000>
3. Log in to the application.
4. Locate and click the "Create Postcard" button.
5. Wait for the create postcard form to load.
6. Generate random pet name and save it in predefined variable.
7. Locate and fill the input field for name with random generated value.
8. Locate and fill the input field for age.
9. Locate and fill the input field for breed.
10. Locate and fill the input field for weight.
11. Locate and fill the input field for image.
12. Press the submit button.
13. Assert that a postcard with the name you just added is present in the list.
14. Assert that the url is <http://localhost:3000/catalog>.

2.3.7. Edit a Postcard Testing (CRUD Functionality)

1. Create a test scope.
2. Go to <http://localhost:3000>
3. Log in to the application.
4. Locate and click on "Dashboard" button.
5. Locate and click the Detail button of the pet you created.
6. Locate and click on Edit button.
7. Wait for the Edit form to load.
8. Locate and fill the name field in the edit form with new value to edit an album.
9. Press the submit button.
10. Assert that the name's value is as expected with edited value.

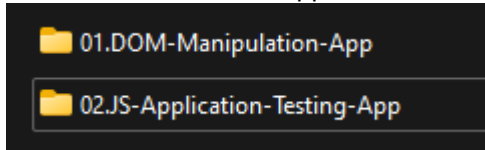
2.3.8. Delete a Postcard Testing (CRUD Functionality)

1. Create a test scope.
2. Go to <http://localhost:3000>
3. Log in to the application.
4. Locate and click on "Dashboard" button.
5. Locate and click the Detail button of the pet we edited.
6. Press the delete button.
7. Assert that a pet with the name you just deleted is NOT present in the list.
8. Assert that the url is <http://localhost:3000/catalog>.

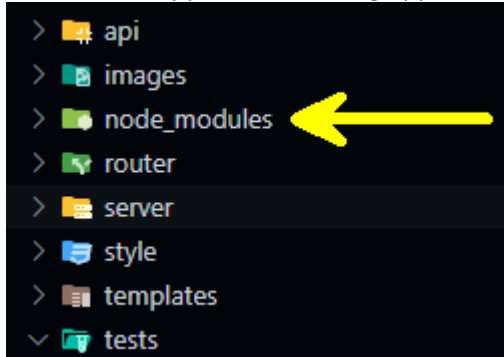
3. How to Submit Your Work

You need to submit your work on the SoftUni website in the Exam Section.

1. Create a folder.
2. Put the folders of both applications in it – the DOM manipulation app and the JS Application Testing app:



3. Go to the JS Application Testing app folder and delete the "node_modules" folder:



4. Archive the folder that contains both applications and your solutions.
5. Upload the archive to the SoftUni website in the course section for your exam.