

Front-End Technologies Basics Exam Preparation I




1. DOM Manipulation

Use the provided skeleton to solve this problem.

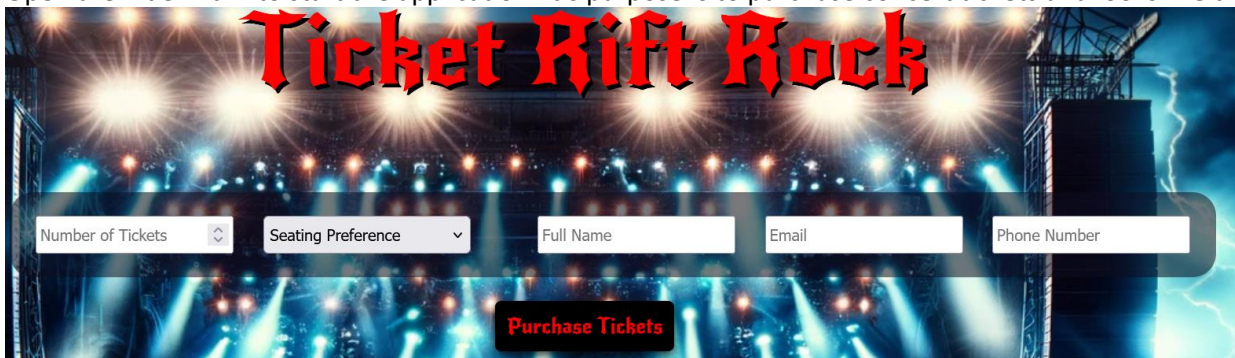
Write the missing functionality of this user interface.

Application structure

You are provided with an HTML application - **Ticket Rift Rock** 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 purchase concert tickets and looks like this:



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

```
<ul id="ticket-preview" style="display: none;">
```

```
<div id="purchase-success" style="display: none;">
```

The ticket preview is supposed to be displayed after the Purchase button is clicked. The purchase success is supposed to be displayed after the purchase is finalized.

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

```
<input type="number" id="num-tickets" placeholder="Number of Tickets" name="numTickets" min="1">
<select id="seating-preference" name="seatingPreference">
  <option value="seating-preference" selected disabled>Seating Preference</option>
  <option value="General">General Admission</option>
  <option value="VIP">VIP</option>
</select>
<input type="text" id="full-name" placeholder="Full Name" name="fullName">
<input type="email" id="email" name="email" placeholder="Email">
<input type="tel" id="phone-number" name="phoneNumber" placeholder="Phone Number">
```

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

```
<p>Count: <span id="purchase-num-tickets"></span></p>
<p>Preference: <span id="purchase-seating-preference"></span></p>
<p>To: <span id="purchase-full-name"></span></p>
<p>Email: <span id="purchase-email"></span></p>
<p>Phone Number: <span id="purchase-phone-number"></span></p>
```

Your Task

Write the missing JavaScript code in the app.js file to make the Ticket Rift Rock work as expected:

1.1. Purchase Tickets button functionality



When the **[Purchase Tickets]** button is clicked, you need to validate that all fields are filled. If this is not the case, do not proceed.

If validation is successful, the following functionality needs to be implemented:

- The information about the purchased tickets is filled in the ticket preview HTML elements – number of tickets, preference, full name, email, phone number
- The ticket preview element must be displayed
- The Purchase Tickets button must be disabled
- The values of the fields in the form must be cleared

1.2. Edit Ticket Info

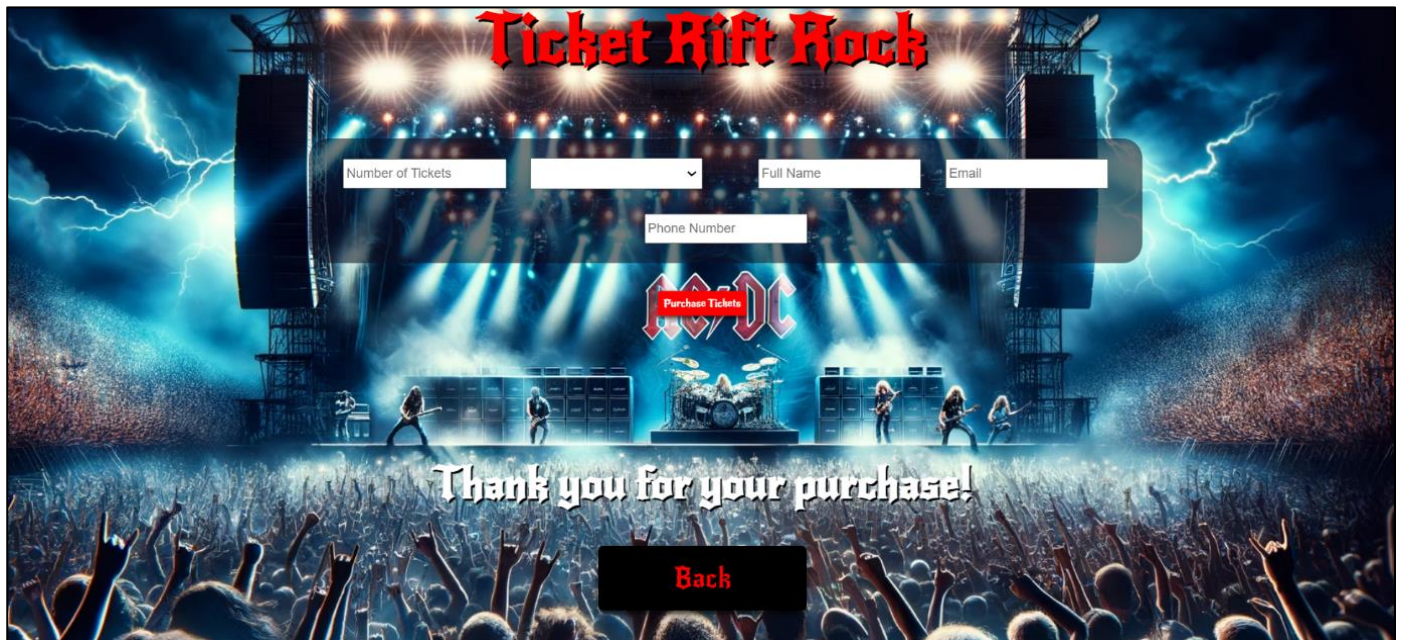
When the **[Edit]** button is clicked, all of the information is loaded in to the input fields from step 1, while the **[Purchase Tickets]** button is enabled again.



The ticket preview element must be hidden.

1.3. Buying

When the **[Buy]** button is clicked, you must hide the ticket preview element and show the purchase success element.



1.4. Back

When the [Back] button is clicked, you must **hide the purchase success element** and enable the [Purchase Tickets] 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 "**My Music Application**". 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.
 - **Catalog page**: See all created albums.
 - **Search Page**: Application functionality for Album searching by name.
 - **Register/Login page**: Option to register a new account or log in.
- 2) For **Logged-In user**:
 - **Create Album Page**: Ability to create new albums.
 - **Catalog**: See all created albums (both their own and others').
 - **Search Page**: Application functionality for Album searching by name.
 - **Logout**: Option to log out of the app.
- 3) **Album details functionality**:
 - **Detail View**: Only logged-in users can see detailed information about an album.
 - **Edit/Delete Buttons**: Only the owner of an album can see and use the Edit and Delete buttons in the detail view to modify or remove the album.
- 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 3 albums, 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 albumName = "";
```

- 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 "My Music Application" 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. Wait for Login button.
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", "Catalog", "Search", "Create Album" 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", "Catalog", "Search", "Login" and "Register" buttons are visible, and "Create Album" and "Logout" buttons are hidden.

2.3.6. Create an Album 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 Album" button.
5. Wait for the create album form to load.
6. Generate random album 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 imgUrl.
9. Locate and fill the input field for price.
10. Locate and fill the input field for releaseDate.
11. Locate and fill the input field for artist.
12. Locate and fill the input field for genre.
13. Locate and fill the input field for description.
14. Press the submit button.
15. Assert that an album with the name you just added is present in the list.
16. Assert that the url is <http://localhost:3000/catalog>.

2.3.7. Edit an Album 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 "Search" button.
5. Locate and fill the input field for search.
6. Locate and click on "Search" button for searching result (you may need to use different locator than "Search" button in navbar – use the class selector .button-list)
7. Locate and click on first album's Detail button with searched album name.
8. Locate and click on Edit button.
9. Wait for the Edit form to load.
10. Locate and fill the name field in the edit form with new value to edit an album.
11. Press the submit button.
12. Assert that the name's value is as expected with edited value.

2.3.8. Delete an Album Testing (CRUD Functionality)

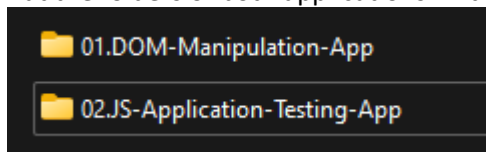
1. Create a test scope.
2. Go to <http://localhost:3000>
3. Log in to the application.
4. Locate and click on "Search" button.
5. Locate and fill the input field for search.
6. Locate and click on "Search" button for searching result.
7. Locate and click on first album's Detail button with searched album name.

8. Press the delete button.
9. Assert that an album with the name you just deleted is NOT present in the list.
10. Assert that the url is <http://localhost:3000/catalog>.

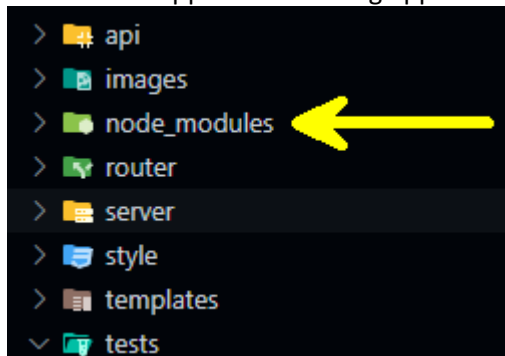
3. How to Submit Your Work

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.