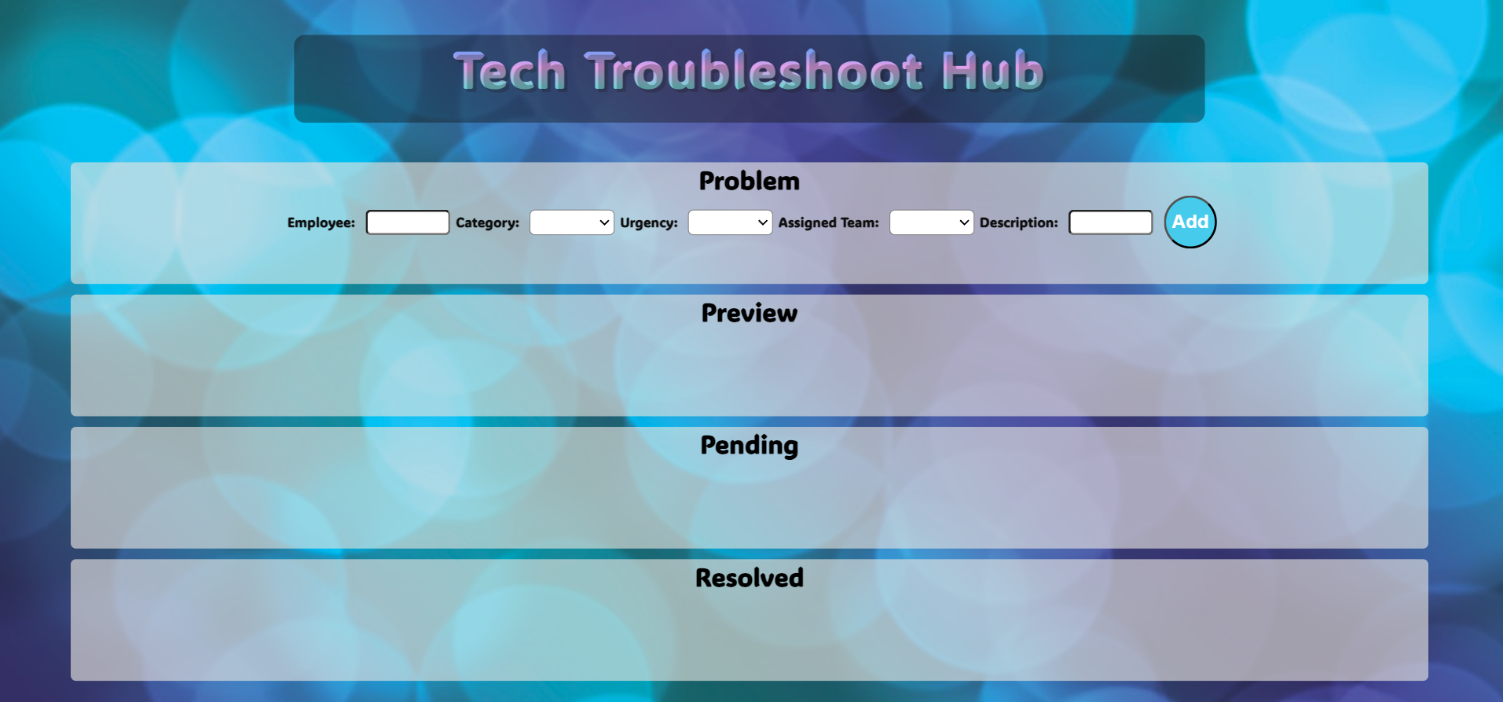
# **Front-End Technologies Basics – Exam Preparation II**

This document defines the exam preparation assignments for the   
["Front-End Test Automation"](https://softuni.bg/modules/141/front-end-test-automation-may-2024/1456) module @ SoftUni

## DOM Manipulation

**Use the provided skeleton to solve this problem.**

**Write the missing functionality** of this user interface. The functionality is divided in the following steps:



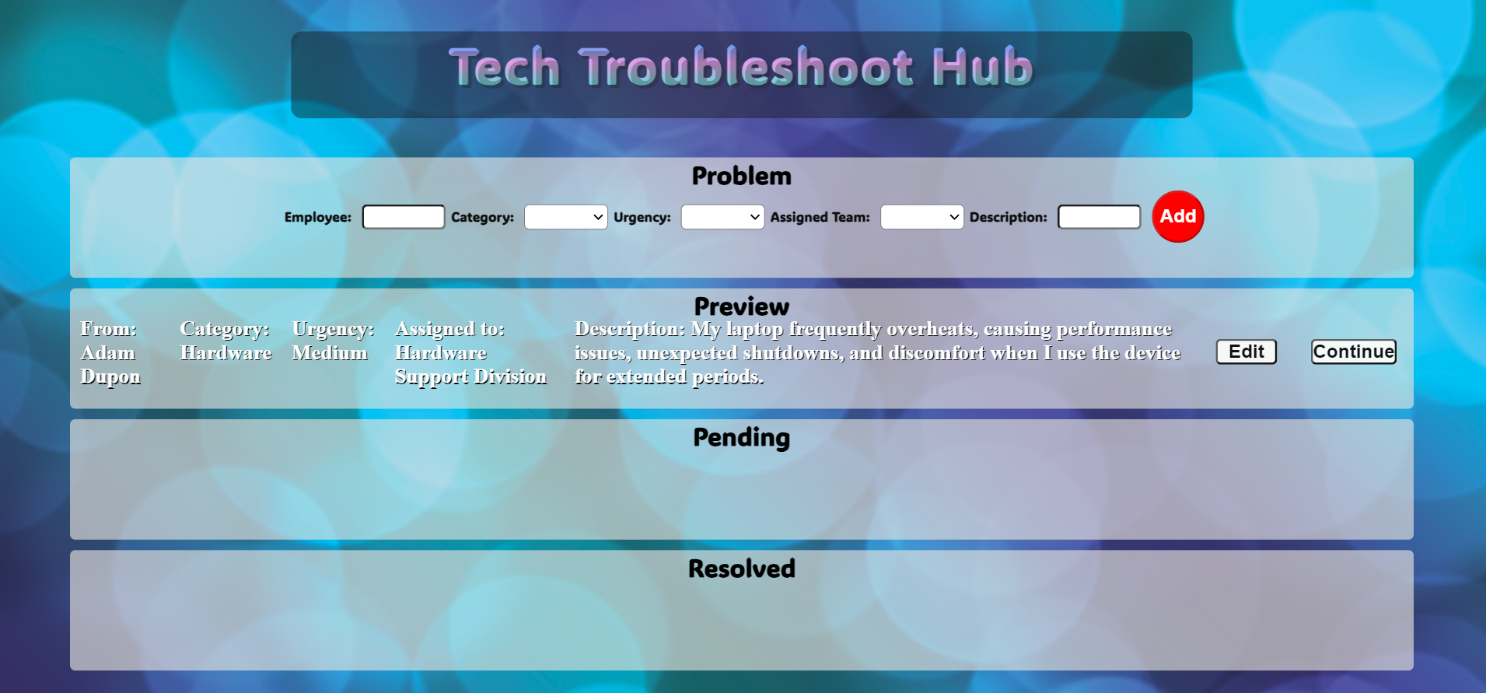
### Your Task

**Write the missing JavaScript code** to make the **Tech Troubleshoot Hub** work as expected:

All fields **(Employee, Category, Urgency, Assigned Team,** and **Description)** are **filled with the correct input**

* **Employee, Category, Urgency, Assigned Team,** and **Description** are **non**-**empty** **strings**. If any of them is empty, the program should not do anything.

### Getting the Information from the Form

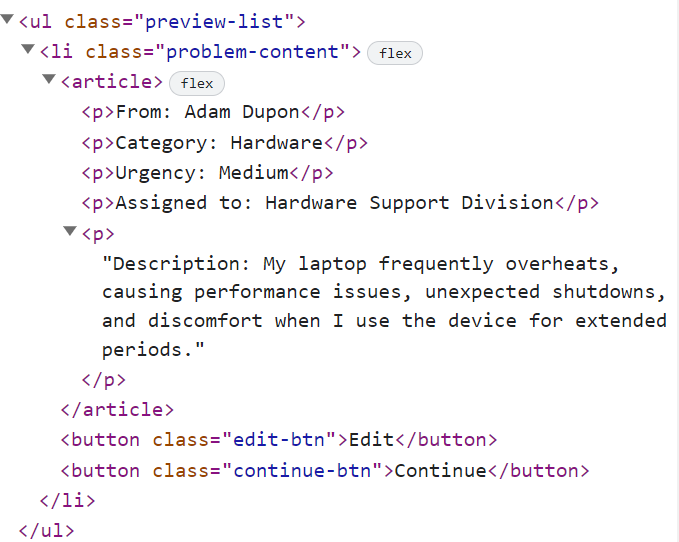


On clicking the **[Add]** button, the information from the input fields is listed in the "**Preview**" section by adding a **list item** to the **".preview-list"** unordered list.

The list item should follow the same text format and order as shown in the provided picture.

Upon clicking the button, the **input** fields must be **cleared**, and the **[Add]** button should be **disabled**. Additionally, the **[Edit]** and **[Continue]** buttons need to be added.

The HTML structure looks like this:

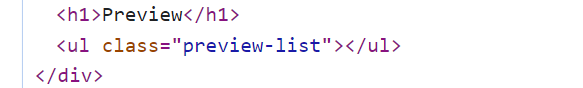


### Editing the Information

**When the [**Edit] **button is clicked, all of the information is loaded in the input fields from step 1 and all buttons in Preview section are removed while the** [Add] **button is enabled again.**

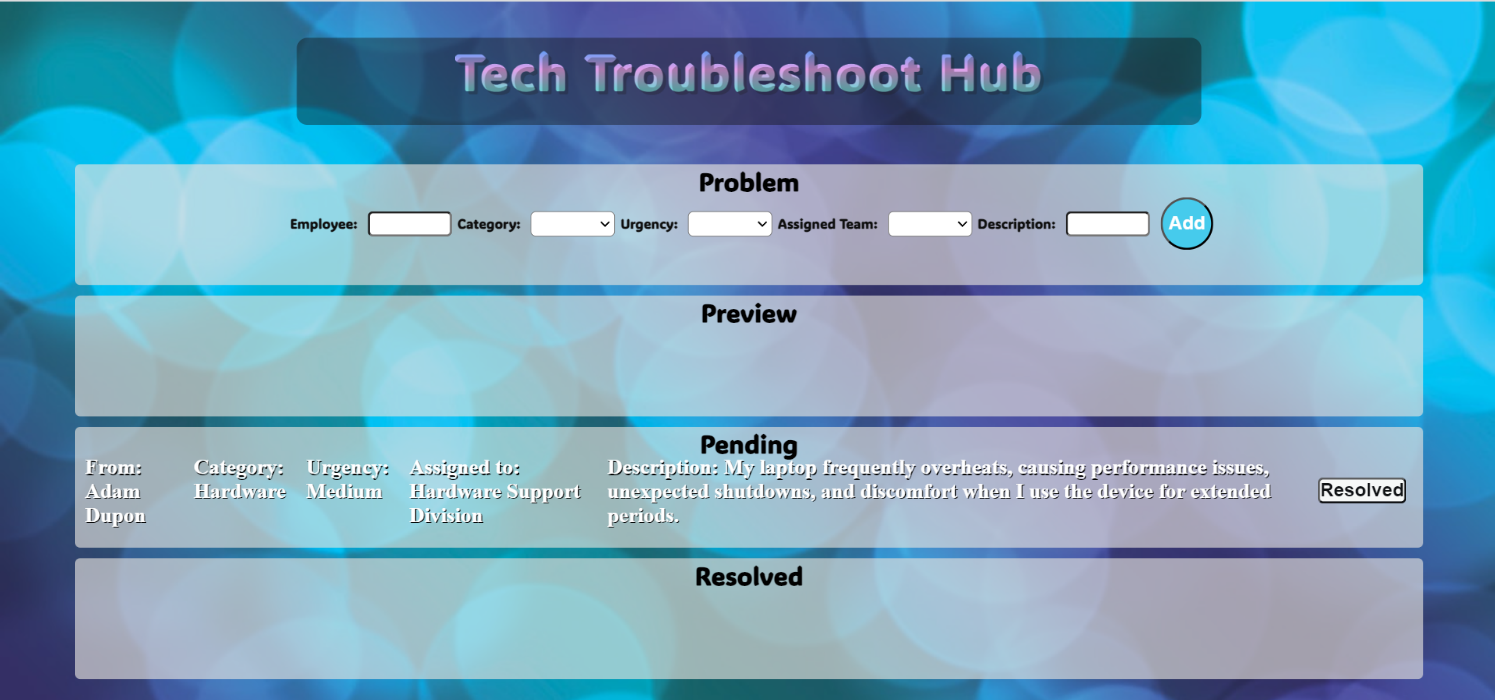
****

**The list items must be removed from the** "preview-list" and **all of the information must go back to the input fields again**.

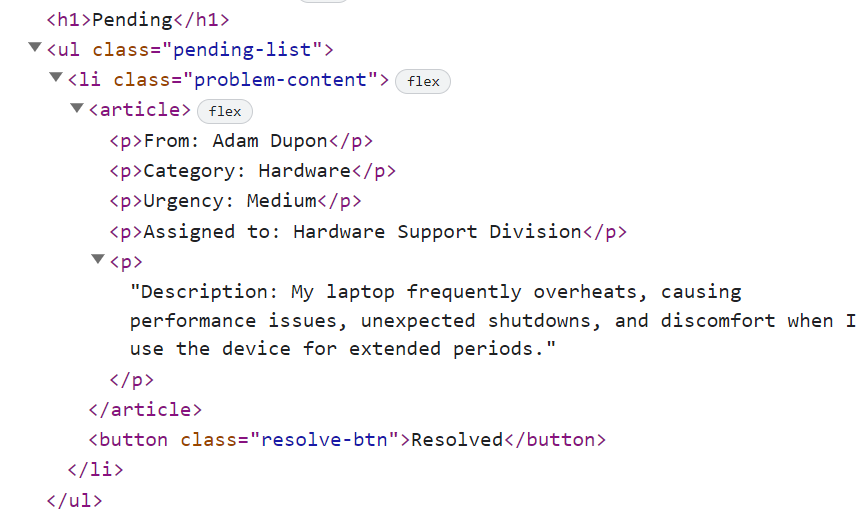
****

### Continue to Pending

**When the** [Continue] **button is clicked,** **the information from** "preview-list" unordered list must be transferred to "pending-list" in the same HTML structure**. For you, this means removing everything inside of the ul with class =** "preview-list" **and adding in** "pending-list", **the list item with same information** and the [Resolved] button must be **added.**

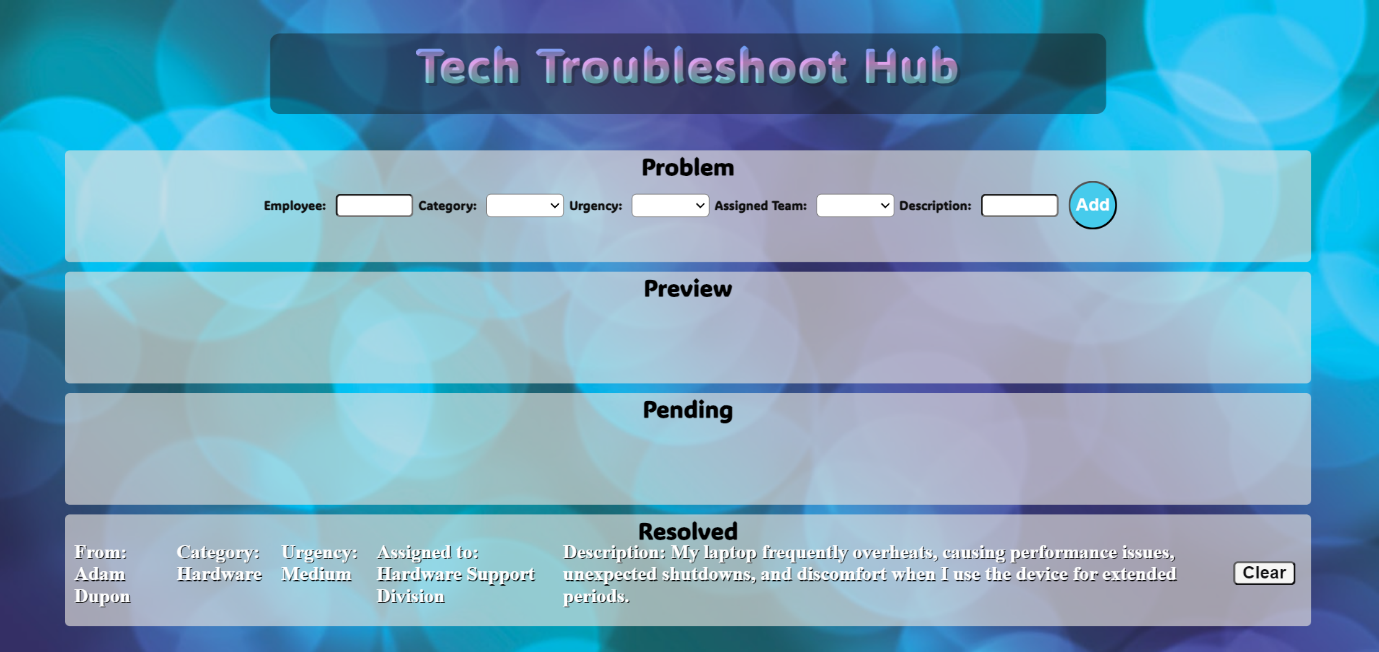
****

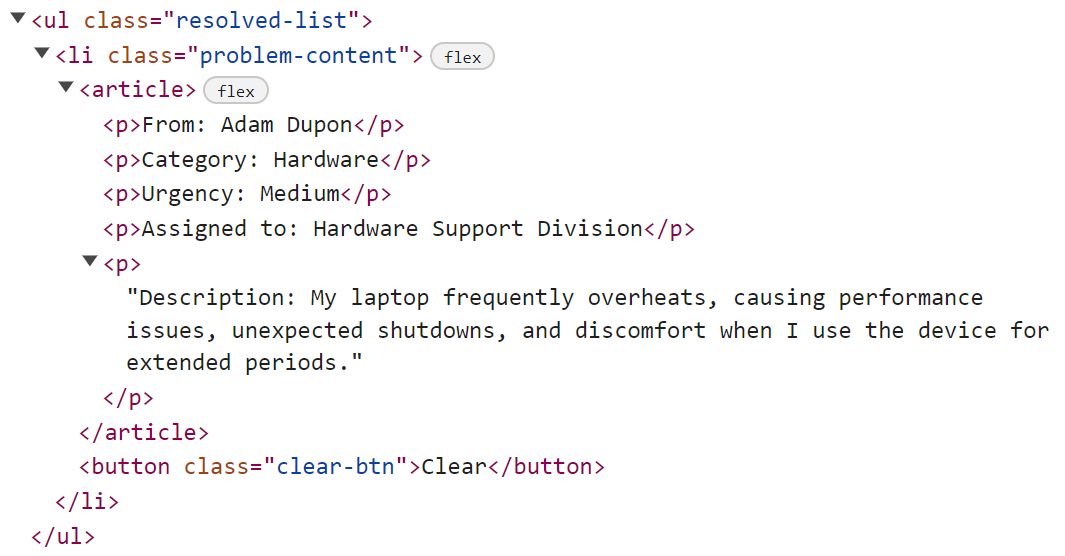
**This is HTML** structure **of** "pending-list" unordered list**:**



### Add to Resolved

**When the** [Resolved] **button is clicked,** **the information from** "pending-list" unordered list must be transferred to "resolved-list" in the same HTML structure**. For you, this means removing everything inside of the ul with class =** "pending-list" **and adding in** "resolved-list", **the list item with same information** and the [Clear] button must be **added.**

****

**This is HTML** structure **of** "resolved-list" unordered list**:**

### Clear Information

**When the** [Clear] **button is clicked,** **the list item must be removed, from the** "resolved-list".

#### 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 integration tests using QUnit and end-to-end (Front-End) tests using Playwright.

### App Introduction

The **application for testing** is called **"My Theater"**. 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 and **all events**.
* **Register/Login page**: Option to register a new account or log in.

1. For **Logged-In user**:

* **Home Page:** View a brief introduction to the app and **all events** (both their own and others').
* **Create Event Page**: Ability to create new theater event.
* **Profile Page**: View their profile and all events they have created.
* **Logout**: Option to log out of the app.

1. **Event details functionality**:

* **Detail View**: Logged-in users can see detailed information about an event.
* **Edit/Delete Buttons**: Only the owner of an event can see and use the Edit and Delete buttons in the detail view to modify or remove the event.

1. **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 theater events, that will always be loaded when you start the application.

### Instructions

A **folder named "tests"** is prepared for you. This folder contains **two subfolders**:

* **"QUnit\_tests" folder** – for your integration tests with QUnit tests.
* **"Playwright\_tests”** – for your Front-End tests with Playwright.

In **the "QUnit\_tests" folder have 2 files**:

* **"integration.test.js"** – In this file, you have to write your QUnit tests.
* **"test.html"** – This is an HTML document that allows you to visualize your QUnit test results in a web browser.

**Note:** To **view the HTML document and test results from your QUnit tests in the browser**, you'll need the **"Live Server" extension** installed in VS Code.

In **the "Playwright\_tests" folder**, you will find **a single file named "e2e.test.js"**. In it, you have to 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 two commands in two Terminals**:

* **"npm run server"** – to start the application back-end server.
* **"npm start"** – to start the HTTP server.

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

### Integration Testing with QUnit

**Hint:** Before you start writing your integration test, you may need to **configure the test execution order**. Also, **use console.log() to check the JSON structure of the response**.

Write the following integration tests with QUnit for the "**My Theater**" application:

#### Register Testing (User Functionality)

1. **Create Test Scope.**
2. **Send Register Request:** Use the fetch API to send a request to **"**<http://localhost:3030/users/register>**"** with**:**

* **Method POST**
* **Headers**:
  + 'content-type' : 'application/json'
* **Body:** Stringified user data - **JSON.stringify(user)**
* **User data Example:**

{

"email": "abv12546@abv.bg",

"password": "123456"

}

1. **Check Response:** Verify if the response is successful**.**
2. **Convert Response to JSON.**
3. **Verify JSON Properties:** Check each property of the JSON response.

**Hint:** You may need to use Math.random() for username and email.

#### Login Testing (User Functionality)

1. **Create Test Scope.**
2. **Send Login Request:** Use the fetch API to send a request to **"**<http://localhost:3030/users/login>**"** with**:**

* **Method POST**
* **Headers**:
  + 'content-type' : 'application/json'
* **Body:** Stringified user login data - **JSON.stringify({email, password})**

1. **Check Response:** Verify if the response is successful**.**
2. **Convert Response to JSON.**
3. **Verify JSON Properties:** Check each property of the JSON response.

**Hint:** You will **need user data such as access token and user Id for CRUD functionalities**.

#### Get All Events Testing (Event Functionality)

1. **Create Test Scope.**
2. **Send Get All Request:** Use the fetch API to send a request to **"**<http://localhost:3030/data/theaters?sortBy=_createdOn%20desc&distinct=title>**"** with**:**

* **Method GET**

1. **Check Response:** Verify if the response is successful**.**
2. **Convert Response to JSON.**
3. **Verify JSON as Array**.
4. **Loop Through Array**:

* Ensure each property exists.
* Verify the property type.

#### Create Event Testing (Event Functionality)

1. **Create Test Scope.**
2. **Send Create Request:** Use the fetch API to send a request to **"**<http://localhost:3030/data/theaters>**"** with**:**

* **Method POST**
* **Headers**:
  + 'content-type' : 'application/json'
  + 'X-Authorization' : token
* **Body:** Stringified event data - **JSON.stringify(event)**
* **Event data Example:**

{

"author": "Random Author",

"date": "24.06.2024",

"title": "random\_title\_123456",

"description": "random\_description\_534554",

"imageUrl": "/images/2.png"

}

1. **Check Response:** Verify if the response is successful**.**
2. **Convert Response to JSON.**
3. **Verify JSON Properties:** Check each property of the JSON response.

#### Edit Event Testing (Event Functionality)

1. **Create Test Scope.**
2. **Send Edit Request:** Use the fetch API to send a request to **"**<http://localhost:3030/data/theaters/:eventId>**"** with**:**

* **Method PUT**
* **Headers**:
  + 'content-type' : 'application/json'
  + 'X-Authorization' : token
* **Body:** Stringified event data - **JSON.stringify(event)**

1. **Edit at least one property of the event**.
2. **Check Response:** Verify if the response is successful**.**
3. **Convert Response to JSON.**
4. **Verify JSON Properties:** Check each property of the JSON response.

**Hint**: Replace ":eventId" with saved event Id value from the previous test.

#### Delete Event Testing (Event Functionality)

1. **Create Test Scope.**
2. **Send Delete Request:** Use the fetch API to send a request to **"**<http://localhost:3030/data/theaters/:eventId>**"** with**:**

* **Method DELETE**
* **Headers**:
  + 'X-Authorization' : token

1. **Check Response:** Verify if the response is successful**.**

**Hint**: Replace ":eventId" with saved event Id value.

### Front-End Testing with Playwright

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

* Needed imports for Playwright:



* Predefined variables that you can use:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

* Before and after test configurations:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

* Test suits:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

Use them and write the following e2e tests with Playwright for the "**My Theater**" application:

#### Registration with Valid Data (Authentication Functionality)

1. Create a test scope.
2. **Go to** [**http://localhost:3000**](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 repeat password**.
9. **Get the response by creating a Promise scope**. Wait for **the response to have a status of 200** and for the **URL to contain "/users/register"** and **press the submit button** for the form.
10. **Assert that the response is okey**.
11. **Parse the response to JSON**:
12. **Assert** that the **email and password** are as expected.

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

#### Login with Valid Data (Authentication Functionality)

1. Create a test scope.
2. **Go to** [**http://localhost:3000**](http://localhost:3000)
3. **Locate and click on the Login button**.
4. **Wait for the register form to load**.
5. **Locate and fill the input field for email.**
6. **Locate and fill the input field for password**.
7. **Get the response by creating a Promise scope**. Wait for **the response to have a status of 200** and for the **URL to contain "/users/login"** and **press the submit button** for the form.
8. **Assert that the response is okey**.
9. **Parse the response to JSON**.
10. **Assert** that the **email and password** are as expected.

#### Logout from the Application (Authentication Functionality)

1. **Create a test scope**.
2. **Go to** [**http://localhost:3000**](http://localhost:3000)
3. **Log in to the application**.
4. **Get the response by creating a Promise scope**. Wait for **the response to have a status of 204** and for the **URL to contain "/users/logout"** and **click on Logout button**.
5. **Assert that the response is okey**.
6. **Wait for Login button.**
7. **Assert that the URL is for home page.**

#### Navigation for Logged-In User Testing

1. Create a test scope.
2. **Go to** [**http://localhost:3000**](http://localhost:3000)
3. **Log in to the application**.
4. **Assert that "Theater", "Create Event", "Profile" and "Logout" buttons are visible**, and **"Login" and "Register" buttons are hidden.**

#### Navigation for Guest User Testing

1. Create a test scope.
2. **Go to** [**http://localhost:3000**](http://localhost:3000)
3. **Assert that "Create Event", "Profile" and "Logout" buttons are hidden**, and **"Theater", "Login" and "Register" buttons are visible.**

#### Create an Event Testing (CRUD Functionality)

1. Create a test scope.
2. **Go to** [**http://localhost:3000**](http://localhost:3000)
3. **Log in to the application**.
4. **Locate and click the "Create Event" button**.
5. **Wait for the create event form to load**.
6. **Locate and fill the input field for title**.
7. **Locate and fill the input field for date**.
8. **Locate and fill the input field for author**.
9. **Locate and fill the input field for description**.
10. **Locate and fill the input field for imageUrl**.
11. **Get the response by creating a Promise scope**. Wait for the **response to have a status of 200** and for the **URL to contain "/data/theaters"** and **press the submit button** for the form.
12. **Assert that the response is okey**.
13. **Parse the response to JSON**.
14. **Assert that title, date, author, description and imageUrl** are as expected.

#### Edit an Event Testing (CRUD Functionality)

1. Create a test scope.
2. **Go to** [**http://localhost:3000**](http://localhost:3000)
3. **Log in to the application**.
4. **Locate and click on "Profile" button**.
5. **Locate and click on first event’s Detail button**.
6. **Locate and click on Edit button**.
7. **Wait for the Edit form to load**.
8. **Locate and fill at least one of edit form fields to change/edit an event**.
9. **Get the response by creating a Promise scope**. Wait for the **response to** **have a status of 200** and for the **URL to contain "/data/theaters"** and **press the submit button** for the form.
10. **Assert that response is okey**.
11. **Parse the response to JSON**.
12. **Assert that title, date, author, description and imageUrl are as expected** (**with** **edited values**).

#### Delete an Event Testing (CRUD Functionality)

1. Create a test scope.
2. **Go to** [**http://localhost:3000**](http://localhost:3000)
3. **Log in to the application**.
4. **Locate and click on "Profile" button**.
5. **Locate and click on first event’s Detail button**.
6. **Get the response by creating a Promise scope**. Wait for the **response to have a status of 200** and for the **URL to contain "/data/theaters"** and **press the delete button**.
7. **Assert that the response is okey**.

## 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:

Картина, която съдържа текст, Шрифт, екранна снимка

Описанието е генерирано автоматично

1. Go to the JS Application Testing app folder and delete the "node\_modules" folder:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

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