# **Front-End Technologies Basics Regular Exam – 29 June 2024**

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

## 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 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.

1. 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.

1. **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.

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 albums, 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 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.

**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 Music Application" 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": "abv2617@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 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 Albums Testing (Album Functionality)

1. **Create Test Scope.**
2. **Send Get All Request:** Use the fetch API to send a request to **"**<http://localhost:3030/data/albums?sortBy=_createdOn%20desc&distinct=name>**"** 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 Album Testing (Album Functionality)

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

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

{

"name": "Random album title\_67373",

"artist": "Unknown",

"description": "Description 67373",

"genre": "Random genre",

"imgUrl": "/images/pinkFloyd.jpg",

"price": "15.25",

"releaseDate": "29 June 2024",

}

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 Album Testing (Album Functionality)

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

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

1. **Edit at least one property of the album**.
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 ":albumId" with saved album Id value from the previous test.

#### Delete Album Testing (Album Functionality)

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

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

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

**Hint**: Replace ":albumId" with saved album 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

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 Music Application" 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 confirm 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 login 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 "Home", "Catalog", "Search", "Create Album" 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 "Home", "Catalog", "Search", "Login" and "Register" buttons are visible, and "Create Album" and "Logout" buttons are hidden.**

#### Create an Album 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 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. **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/albums"** and **press the submit button** for the form.
15. **Assert that the response is okey**.
16. **Parse the response to JSON**.
17. **Assert that name, imgUrl, price, releaseDate, artist, genre, description** are as expected.

#### Edit an Album 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 "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 at least one of edit form fields to change/edit an album (Keep in mind that you will need the random generated album name for delete test. Changing the name may be a problem for the delete test)**.
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/albums"** and **press the submit button** for the form.
12. **Assert that response is okey**.
13. **Parse the response to JSON**.
14. **Assert that name, imgUrl, price, releaseDate, artist, genre, description are as expected** (**with** **edited values**).

#### Delete an Album 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 "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. **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/albums"** and **press the delete button**.
9. **Assert that the response is okey**.