# **Practical Project: Automated Testing of Single Page App with QUnit**

This document defines the practical project assignments for the   
**"QA Front-End Automation" Course @ SoftUni**

It's time to start **testing our SPA application**. In this practical project, we will **perform QUnit integration testing on the application**. This testing will be **very close to end-to-end testing,** but not exactly the same, because the tests will **check requests and responses using the JS fetch API, but not the UI functionality**. We can **also consider these as black box tests** since the test design will **not focus on the implementation of the SPA app or the frameworks it uses** (such as lit-html and page.js).

So, let's start with configuring our project for the testing.

## Configure The Project for Testing

To make our tests, we will need to set up a few things so we can easily write our test and to check our results. We will **use the ES6 standard**. Here are the steps:

1. Open your SPA project, which you implemented in the previous two practical project parts, in VS Code.
2. **Create a folder** by right-clicking on the EXPLORER tab, then choosing "New Folder" and **name it "tests"**:

Картина, която съдържа текст, екранна снимка, софтуер, Мултимедиен софтуер

Описанието е генерирано автоматично

1. Create two sub-folders and **name them "QUnit\_tests" and "Playwright\_tests"**:

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

Описанието е генерирано автоматично

**NB! For now, leave the "Playwright\_tests" folder empty. We will need it for the next practical project session.**

1. In the "QUnit\_tests" folder, **create an HTML** **document** **named "test.html"**:

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

Описанието е генерирано автоматично

1. Open the HTML document and **write the following HTML script in it**:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

**NB! Don't forgot to save the file before closing it.**

1. Now, **create another file in the same folder**. We will write our tests in this file later. **Name it "integration.test.js":**

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

Описанието е генерирано автоматично

1. We have to **include this file as a script in our "test.html" file**:

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

Описанието е генерирано автоматично

This is all we need do for the configuration for testing. Note that **we do not install QUnit through NMP manager** because **we won't use CommonJS**. As we said earlier, **we are going to use ES6**, so **we imported QUnit as scripts in our html document**:

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

Описанието е генерирано автоматично

Also, **there won't be a need of import QUnit into our "integration.test.js" file**. We will simply write our test there, and **they will be executed through HTML with the help of the "Live Server" extension in VS Code**.

## Before Testing

There are **few things that we need to know before writing our tests**:

1. When we write unit tests, we need to fully isolate them so each test can be executed independently without any dependencies. However, since we are writing integration tests in this practical project, we don't need to follow this strategy. **We can write our integration tests to depend on each other**.
2. **QUnit is mainly a framework for unit testing**, so the execution of the **tests follows a random order each time it is performed**. This can be **problematic for integration testing with QUnit**. Therefore, we **need to add a specific rule in our QUnit test file so the tests will be performed in the order they are written**. Here is the **piece of code we need to place before the tests in our "integration.test.js" file**:



1. We need **to install an extension for easy execution of our integration test**s. We **strongly recommend the "Live Server" extension in VS Code**. Here are the steps to install it:

* In VS Code, **open the "Extensions" tab** in the sidebar:

Картина, която съдържа текст, екранна снимка, софтуер, Мултимедиен софтуер

Описанието е генерирано автоматично

* **Search for "Live Server”** in the search bar:

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

Описанието е генерирано автоматично

* **Press the "Install" button**:

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

Описанието е генерирано автоматично

* **Go to your HTML page for testing ("test.html"),** **right-click somewhere in the script**, and **choose "Open with Live Server" to load the page in the browser and see the results of our tests**:

Картина, която съдържа текст, екранна снимка, софтуер, Мултимедиен софтуер

Описанието е генерирано автоматично

* For now, we **don’t have any written tests**, so we **will not see any results on the page**. We may **see an error message**:

Картина, която съдържа текст, екранна снимка, Уеб страница, Уебсайт

Описанието е генерирано автоматично

We will explain how to execute tests in the next section after we write our first integration test.

1. Our tests will be executed on the working application, so we need to start our app server before writing the tests. Starting the server at this stage will allow us to use console.log() to check responses in the browser when we make them through the API. To start the server, follow these steps:

* Open the "package.json" file:
* Check the "scripts" property for the command "server". If there isn’t such a command, add "server" : "cd server && node server.js". The scripts part of the JSON should look like this:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

* Once you have or add this, you can start the server by executing the command "npm run server" in the Terminal:

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

Описанието е генерирано автоматично

## Write Your First Integration Test with QUnit

Let's write our first integration test for our SPA project:

1. **Open the "integration.test.js" file**, which we created earlier**:**

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

Описанието е генерирано автоматично

1. Don't forgot to **write the piece of code that we noted earlier for the test execution order:**

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

Описанието е генерирано автоматично

1. Sinse our first test will be a part of **user functionality**, let’s **create a test suit** for it:

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

Описанието е генерирано автоматично

1. In the QUnit module, **write a test scope** for user registration:

Картина, която съдържа екранна снимка, текст, софтуер, Мултимедиен софтуер

Описанието е генерирано автоматично

**IMPORTANT!!! Note that our delegate function is asynchronous. We will need to use async/await operation when using fetch API.**

1. We need to **set a variable outside the test scope to hold the base URL** for our requests through the fetch API. This will **allow us to reuse it** for the next test we write:

Картина, която съдържа текст, екранна снимка, софтуер, Мултимедиен софтуер

Описанието е генерирано автоматично

1. Inside the test scope, **create another variable for the path of the registration request**. So, our final URL will be "<http://localhost:3030/users/register>”:

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

Описанието е генерирано автоматично

1. There's one tricky part when testing registration. **Each user must have a unique email address**, and we **can’t register two users with the same email**. We need to **use different emails for each test execution**. Let’s **use Math.random() to set unique emails**. Here is the implementation for this:

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

Описанието е генерирано автоматично

**Note:** *We use Math.random() to get a random number between 0 and 1 (e.g., 0.167564). We multiply it by 10,000 and use Math.floor() to get only the integer part, cutting off all digits after the decimal point. Then, we set a string variable using a string literal to insert the random number we generated.*

1. Thinking ahead, it's better to **set the user object outside the current test** so **we can later use it for login**. We will need **the same user data to log into the SPA application**:

Картина, която съдържа текст, екранна снимка, софтуер, дисплей

Описанието е генерирано автоматично

**IMPORTANT!!! Leave the email property of the user object empty but set a password. We don’t need a unique password.**

1. Then **set value to the user email inside the test scope**:

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

Описанието е генерирано автоматично

1. Now, it's time to **make our request through the fetch API and get the response**:

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

Описанието е генерирано автоматично

As you can see from the picture, **our fetch API request contains**:

* For the **first parameter – baseUrl** (the variable we set earlier outside the test) **plus the path variable**, so the whole 1st parameter after interpolation of the string will be equal to **"**[**http://localhost:3030/users/register**](http://localhost:3030/users/register)**"** (the address where we send the request).
* The **second parameter** is **an object** where we define **the HTTP method type, headers of the request, and the body that contains the user data as a string in JSON format**.

1. Let’s **make an assertion to check if the response is successful**:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

1. We are not done with the whole test, but let's **check if the already written assertion is passing**. We need to **start our app server first**, then **go to "test.html"** and **load the page in the browser** with the help of the "Live Server" extension. Here are the steps:

* **Save the file** where we are writing our integration test.
* **Open the Terminal and execute "npm run server".**
* **Go to "test.html" page**, right-click somewhere on the script file and **choose "Open with Live Server"**.
* You should see **the test page without any error**:

Картина, която съдържа текст, екранна снимка, софтуер, Уеб страница

Описанието е генерирано автоматично

**Note: You can check details for these steps in the section "2. Before Testing".**

1. Now, let's **make a deeper check of the data being transferred through the response** (HTTP protocol). To do this, we will **transform our response into JSON**:



**The expected JSON** should be like this:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

Assertions should **check each property of the JSON** to **verify if the property exists, its type, and its value** (where possible):

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

1. **Check again if all tests passed**. If you already have "test.html" open through the "Live Server" extension, just press CTRL + S to save the changes. The test page will automatically load the new test information:

Картина, която съдържа текст, екранна снимка, софтуер, Уеб страница

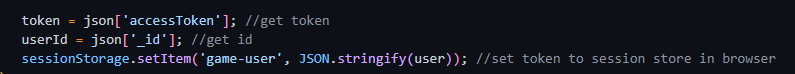
Описанието е генерирано автоматично

1. **Create two variables outside the test scope**. We will need them to **hold values for our user ID and token**, so we can use them in the next tests:

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

Описанието е генерирано автоматично

1. **Set values for both created variables inside the test** and **set user data in the session storage**:



**Note: We need user information in session storage because our SPA application uses it for user authentication.**

1. The **whole test** should look like this:

Картина, която съдържа текст, екранна снимка

Описанието е генерирано автоматично

And **the file** should look like this:

Картина, която съдържа текст, екранна снимка, дисплей, софтуер

Описанието е генерирано автоматично

## Write Integration Tests for All Functionalities

Now that you're familiar with integration testing with QUnit, let's make tests for all other functionalities:

### Test Login (User Functionalities)

1. **Create Test Scope:** Make a new test scope in the test suite for user functionality**:**

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

Описанието е генерирано автоматично

1. **Define URL Path:** Create a URL path variable inside the test**:**



1. **Send Login Request:** Use the fetch API to send a request to **"**[**http://localhost:3030/users/login**](http://localhost:3030/users/login)**"** whit**:**

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

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

Описанието е генерирано автоматично

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



1. **Convert Response to JSON:** Transform the response into JSON format**:**



1. **Expected JSON Response:**

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

1. **Verify JSON Properties:** Check each property of the JSON response:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

1. **Store Variables:** Set values to variables outside the test scope for use in other tests:



1. **Save User Data:** Store user data in the browser session storage:



1. **Test Structure:** The complete test should look like this example:

Картина, която съдържа текст, екранна снимка

Описанието е генерирано автоматично

1. **Check Test Result:** Review the test result on the "test.html" page.

### Test Game Functionality

**Create a test suite (QUnit module) for game functionality**. Write tests for the following:

#### Get All Games Functionality Testing

1. **Create Test Scope:** Define a QUnit test scope**.**
2. **Define Path Variable:** Set a path variable with value **'data/games'.**
3. **Define Query Parameters:** Create a **variable "queryParams" with the value '?sortBy=\_createdOn%20desc'** to ensure games are sorted in descending order**.**
4. **Send Request:** Use the fetch API to send a request to[**http://localhost:3030/data/games?sortBy=\_createdOn%20desc**](http://localhost:3030/data/games?sortBy=_createdOn%20desc)**.** No need to specify the method (GET), headers, or body.
5. **Check Response:** Verify if the response is successful**.**
6. **Convert Response to JSON**: Transform the response into JSON. The expected JSON should be an array of objects:

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

Описанието е генерирано автоматично**

1. **Verify JSON as Array:** Use **"Array.isArray()"** to check if the response is an array.
2. **Loop Through Array:** Use a loop to check each object in the array:

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

**NB! Do not check the values of "\_createdOn" and "\_id" as they are auto-generated. Ensure "\_createdOn" is a number.**

1. **Check Test Results:** Review the test results on the "test.html" page**.**
   * 1. **Create Game Functionality Testing**
2. **Create Test Scope:** Define a QUnit test scope**.**
3. **Define Path Variable:** Set a path variable with value 'data/games'**.**
4. **Define Game Object:** Create a variable outside the test scope to hold a game object**:**

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

Описанието е генерирано автоматично**

1. **Set Game Properties:** Inside the test scope, set the properties for the game (title, category, maxLevel, and summary).
2. **Send Create Game Request:** Use the fetch API to send a request to **"**[**http://localhost:3030/data/games**](http://localhost:3030/data/games)**"** with**:**

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

**NB! This operation requires authorization with the header 'X-Authorization': token, using the access token saved earlier from user registration and login tests.**

1. **Check Response:** Verify if the response is successful**.**
2. **Convert Response to JSON:** Transform the response into JSON format. The expected JSON should look like the example provided:

**Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично**

1. **Verify JSON Properties:** Check all properties of the JSON response**.**
2. **Store Game ID:** Create a variable **"lastCreatedGameId"** outside the test scope to use in later tests. Inside the test, set the game ID to this variable**.**
3. **Check Test Results**: Review the test results on the "test.html" page**.**

#### Get By Id Functionality Testing

1. **Create Test Scope:** Define a QUnit test scope**.**
2. **Define Path Variable:** Set a path variable with value 'data/games'**.**
3. **Send Get by Id Game Request:** Use the fetch API to send a request to**"**[**http://localhost:3030/data/games/lastCreatedGameId**](http://localhost:3030/data/games/lastCreatedGameId)**".** The **"lastCreatedGameId" should be the value stored outside the test scope (defined in the previous test for creating a game).** No need to specify the method (GET), headers, or body.
4. **Check Response:** Verify if the response is successful**.**
5. **Convert Response to JSON:** Transform the response into JSON format. The expected JSON should look like the example provided:

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

Описанието е генерирано автоматично**

1. **Verify JSON Properties:** Check all properties of the JSON response**.**
2. **Check Test Results**: Review the test results on the "test.html" page**.**
   * 1. **Edit Game Functionality Testing**
3. **Create Test Scope:** Define a QUnit test scope**.**
4. **Define Path Variable:** Set a path variable with value 'data/games'**.**
5. **Change Game Properties:** Inside the test scope, change the properties for the game (title, category, maxLevel, and summary) that we created and stored outside the test scopes earlier.
6. **Send Edit Game Request:** Use the fetch API to send a request to **"**[**http://localhost:3030/data/games/lastCreatedGameId**](http://localhost:3030/data/games/lastCreatedGameId)**"** with**:**

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

**NB! This operation requires authorization with the header 'X-Authorization': token, using the access token saved earlier from user registration and login tests. Also, the "lastCreatedGameId" should be the value stored outside the test scope (defined in the test for creating a game)**

1. **Check Response:** Verify if the response is successful**.**
2. **Convert Response to JSON:** Transform the response into JSON format. The expected JSON should look like the example provided:

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

Описанието е генерирано автоматично**

1. **Verify JSON Properties:** Check all properties of the JSON response**. Be sure that values are changed as you expected.**
2. **Check Test Results**: Review the test results on the "test.html" page**.**

#### Delete Game Functionality Testing

1. **Create Test Scope:** Define a QUnit test scope**.**
2. **Define Path Variable:** Set a path variable with value 'data/games'**.**
3. **Send Delete Game Request:** Use the fetch API to send a request to **"**[**http://localhost:3030/data/games/lastCreatedGameId**](http://localhost:3030/data/games/lastCreatedGameId)**"** with**:**

* **Method DELETE**
* **Headers**:
  + 'X-Authorization' : token
* **Body:** Stringified game data- **JSON.stringify(game)**

**NB! This operation requires authorization with the header 'X-Authorization': token, using the access token saved earlier from user registration and login tests. Also, the "lastCreatedGameId" should be the value stored outside the test scope (defined in the test for creating a game). Don’t need Body for the delete request.**

1. **Check Response:** Verify if the response is successful**.**
2. **Don’t need to check JSON here,** but if you wish, you can check the "\_deleteOn" property**:**

**Картина, която съдържа Шрифт, текст, екранна снимка, Графика

Описанието е генерирано автоматично**

1. **Check Test Results:** Review the test results on the "test.html" page**.**

### \*Test Comment Functionality

**Create a test suite (QUnit module) for comment functionality**. Write tests for the following:

#### Test Newly Created Game with No Comments

1. Create Test Scope**.**
2. **Create New Game:** Send a request to **"**[**http://localhost:3030/data/games**](http://localhost:3030/data/games/lastCreatedGameId)**"** to create a new game and get (store) the game id.
3. **Get Comment:** Send a request to "[**http://localhost:3030/data/comments?where=gameId%3D%22${gameId}%22**](http://localhost:3030/data/comments?where=gameId%3D%22$%7bgameId%7d%22)**".** No need to specify the method (GET), headers, or body**.**

**NB! Use the stored gameId in the query parameters of the URL.**

1. **Check Response:** Verify if the response is successful**.**
2. **Convert Response to JSON:** Ensure you get an array and it is empty.
3. Check Test Results**.**

#### Post New Comment

1. Create Test Scope**.**
2. **Create Comment:** Create a comment with the following structure:

**Картина, която съдържа текст, Шрифт, екранна снимка, линия

Описанието е генерирано автоматично**

**NB! Use the gameId from the previous test for the value gameIdForComments.**

1. **Send Post Comment Request:** Send a request to **"**[**http://localhost:3030/data/comments**](http://localhost:3030/data/comments)**"** with**:**

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

1. **Check Response:** Verify if the response is successful**.**
2. **Convert Response to JSON:**

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

Описанието е генерирано автоматично**

1. **Verify JSON Properties.**
2. Check Test Results**.**

#### \*\* Comments For Specific Game Functionality Testing

Reuse the test "4.3.1. Test Newly Created Game with No Comments" and modify it to check the comments for a specific game. You might want to verify the JSON, so here is the expected JSON response:

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

Описанието е генерирано автоматично

**Hint:** Check the test for " 4.2.1. Get All Games Functionality Testing".

## \*Optimize the tests

Think about how to optimize your tests. To avoid repeating variables, refactor your code to reuse already created variables. Use dynamic values for variables by using Math.random(). Replace hard-coded values for titles, summaries, and categories of game objects with dynamic values. Do the same for comments.