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

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

After we did our integration tests with QUnit, it's time to move on to end-to-end tests (e2e tests). We will use the Playwright framework for our Front-End tests. This means we will interact with the UI elements to perform tests and make sure everything works as expected – we will simulate user behavior.

Let's start by setting up our project for testing.

## Configure The Project for UI Testing

As we did in the previous part of our practical project, we created the **"Playwright\_tests" folder**:

Картина, която съдържа текст, екранна снимка, софтуер, Компютърна икона

Описанието е генерирано автоматично

We are going to **use it for our e2e tests now**. Let's start by setting up everything we need:

1. In the "Playwright\_tests" folder, **create a JS file and name it "e2e.test.js"**. We will write our tests in it:

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

Описанието е генерирано автоматично

1. If you are using a **newly opened project**, you may need to **run the command "npm init -y"** to initialize the node modules:



1. **Install Playwright** by running the command **"npm i @playwright/test"**:

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

Описанието е генерирано автоматично

1. In the e2e.test.js file, **write the imports we need** for our test as shown in the picture below:



Let's analyze what we are importing to better understand the framework:

* **describe** – used for our test suites.
* **test** – used for each test case.
* **beforeAll** – runs once before all tests.
* **afterAll** – runs once after all tests.
* **beforeEach** – runs before each test.
* **afterEach** – runs after each test.

This is all we need to do to set up for testing.

## Set up Playwright and Create Variables

To make testing easier, we need to **create a few variables** **that we will use in most of our tests**:

1. **Create a variable for the URL and name it "host":**



1. **Create variables "browser", "context", and "page"** – we will need them for Playwright before and after test executions:

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

Описанието е генерирано автоматично

1. **Create a user variable** to hold the user data. We can reuse this for tests where it is needed:

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

Описанието е генерирано автоматично

1. **Create a describe block** and write all tests and execution settings in it:

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

Описанието е генерирано автоматично

1. **Use "beforeAll" and "afterAll" to initialize the browser for test execution and to close it after all tests**:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

1. **Use "beforeEach" and "afterEach" to create a new incognito browser context and a new page inside the context**:

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

Описанието е генерирано автоматично

So, the whole file should look like this for now:

Картина, която съдържа текст, екранна снимка, софтуер

Описанието е генерирано автоматично

## Before Writing Tests

We need to set up a few things to make the testing process easier:

1. First, check the **"package.json" file**:

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

Описанието е генерирано автоматично

1. There is a **section (JSON property) named "scripts"**:

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

Описанието е генерирано автоматично

1. You should have **"start" and "server" JSON properties**, which **we can execute as commands in Terminal**:

* When you run **the command "npm run server"** in Terminal, it **will start the back-end server for the application**:



* When you run **the command "npm start"** in Terminal, it **will start the HTTP server**, which acts as an interface between the back-end server and the client (browser)



1. You will need to **start the back-end and HTTP server in different Terminals**:

* First, open a Terminal by **choosing "Terminal" from the menu, then choose "New Terminal"**:

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

Описанието е генерирано автоматично

* **Run the command "npm run server"** to start the back-end server of the application:
* **Open another Terminal** by clicking the plus sign in the Terminal window:

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

Описанието е генерирано автоматично

* **Run the command "npm start"** to start the HTTP server **in the newly opened Terminal**.
* **Open a third Terminal**. We will use this one later for executing e2e tests.

**NB! Ensure that both the back-end and HTTP servers are started for test execution. Do not use "lite-server" for test execution, as lite-server has pop-ups that can interrupt test execution.**

## Write Frond-End Authentication Tests

To understand the process of front-end authentication testing, let's start by planning. We will begin with user stories and test scenarios, and based on these, we will write our test cases—first as descriptions and then as implementations using Playwright.

### User Stories, Test Scenarios and Test Cases for Authentication Testing

For the current test suite, we have the following **user stories**:

1. **Register**: As a new user, I want to create an account so that I can access the website's features.
2. **Login**: As a registered user, I want to log into my account so that I can access my personal data and features.
3. **Logout**: As a logged-in user, I want to log out of my account so that my personal information remains secure.

Based on these stories, we create the following example **test scenarios and their test cases**:

1. **Register**: Test Scenario 1: Register with **valid** data
2. **Register**: Test Scenario 2: Register with **empty fields**
3. **Login**: Test Scenario 1: Login with **valid credentials**
4. **Login**: Test Scenario 2: Login with **empty fields**
5. **Logout**: Test Scenario 1: Logout from the application

Let's describe our test cases, which we will later implement using Playwright:

1. **Register: Test Scenario 1: Register with valid data**:

* **Precondition**: Active back-end and HTTP servers, guest user.
* **Steps**:

1. **Go to** [**http://localhost:3000**](http://localhost:3000)
2. **Click on the Register button/link** on the navigation bar (top right corner)
3. **Wait for the registration form to load**
4. **Fill the email input field with a unique email value**
5. **Fill the password input field**
6. **Fill the confirm password input field**
7. **Press the submit button** of the form
8. **Wait for response**
9. **Assert that the response is okey**
10. **Assert the user data** (email and password) are as expected

* **Expected Result**: The user is registered successfully, and the user data is correct.
* **Actual Result** – To be recorded after testing.

1. **Register: Test Scenario 2: Register with empty fields:**

* **Precondition**: Active back-end and HTTP servers, guest user.
* **Steps**:

1. **Go to** [**http://localhost:3000**](http://localhost:3000)
2. **Click on the Register button/link** on the navigation bar (top right corner)
3. **Wait for the registration form to load**
4. **Leave the input fields empty**
5. **Press the submit button** on the form
6. **Wait for response**
7. **Assert that the URL** is <http://localhost:3000/register>

* **Expected Result**: There is no redirection.
* **Actual Result** – To be recorded after testing.

1. **Login: Test Scenario 1: Login with valid credentials**:

* **Precondition**: Active back-end and http servers, guest user.
* **Steps**:

1. **Go to** [**http://localhost:3000**](http://localhost:3000)
2. **Click on the Login button/link** on the navigation bar (top right corner)
3. **Wait for the login form to load**
4. **Fill the email input field** with an **existing email**
5. **Fill the password input field** with the **correct password**
6. **Press the submit button** on the form
7. **Wait for response**
8. **Assert that the response is okey**
9. **Assert the user data** (email and password) are as expected

* **Expected Result**: The user is logged in successfully, and the user data is correct.
* **Actual Result** – To be recorded after testing.

1. **Login: Test Scenario 2: Login with empty fields**:

* **Precondition**: Active back-end and http servers, guest user.
* **Steps**:

1. **Go to** [**http://localhost:3000**](http://localhost:3000)
2. **Click on the Login button/link** on the navigation bar (top right corner)
3. **Wait for the login form to load**
4. **Leave the input fields empty**
5. **Press the submit button** on the form
6. **Wait for the response**
7. **Assert that the URL** is <http://localhost:3000/login>

* **Expected Result**: There is no redirection.
* **Actual Result** – To be recorded after testing.

1. **Logout: Test Scenario 1: Logout from the application**:

* **Precondition**: Active back-end and http servers, guest user.
* **Steps**:

1. **Go to** [**http://localhost:3000**](http://localhost:3000)
2. **Click on the Login button/link** on the navigation bar (top right corner)
3. **Wait for the login form to load**
4. **Fill the email input field** with an **existing email**
5. **Fill the password input field** with the **correct password**
6. **Press the submit button** on the form
7. **Wait for redirection to the homepage** for **logged-in users**
8. **Click on the Logout button** on the navigation bar (top right corner)
9. **Wait for the response**
10. **Assert that the response is okey**
11. **Wait for redirection to the homepage** for **guest users**
12. **Assert that the URL** is <http://localhost:3000/>

* **Expected Result**: The user is logged out successfully, and there is a redirection to the homepage for guests.
* **Actual Result** – To be recorded after testing.

**IMPORTANT: We will only cover a part of all possible test cases. Feel free to write more test cases once you are done with these.**

### Create a Test Suit for Authentication Tests

Inside the main describe block for e2e tests, **create another describe block for the Authentication test suite**:

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

Описанието е генерирано автоматично

### Playwright Test for "Register: Test Scenario 1: Register with Valid Data"

Let's **write our first Playwright test for our SPA project**:

1. **Create a test scope**:

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

Описанието е генерирано автоматично

**IMPORTANT: The function is asynchronous, which is necessary for await operations.**

1. **Go to** [**http://localhost:3000**](http://localhost:3000)

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

Описанието е генерирано автоматично

**IMPORTANT: The variable "host" holds the value "http://localhost:3000," which we created earlier outside the test scope.**

1. **Click on the Register button**:

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

Описанието е генерирано автоматично

1. **Wait for the register form to load**:

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

Описанието е генерирано автоматично

1. **Create a unique user email value** using a **random value** and **set it for the user object created earlier outside the test scope**:

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

Описанието е генерирано автоматично

1. **Locate the input field for email**:



1. **Fill the email field** with the user email value:

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

Описанието е генерирано автоматично

1. **Locate the input field for password**:



1. **Fill the password field** with the password value:



1. **Locate the input field for confirm password**:



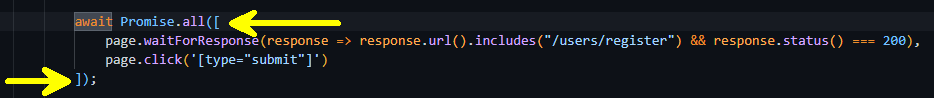
1. **Fill the confirm password field** with the corresponding value:



1. **Create a Promise scope** for parallel execution of multiple operations:



1. In the Promise scope, **wait for the registration response**. In this case, wait for a response that includes the "/users/register" endpoint in the URL and has a status code of 200. **Note: the conditions are given as elements of array:**

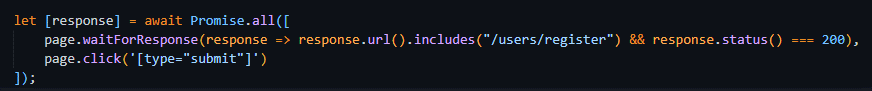


**IMPORTANT: The endpoint is different from the application's URL in the browser because it points to the application API. You can find the correct endpoint in the "api" folder in the "users.js" file (similarly, you can find endpoints for games in the "games.js" file and for comments in the "comments.js" file):**

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

Описанието е генерирано автоматично

1. The Promise scope **returns the response** in an array, so **save it in a variable**:



1. **Assert that the response is okey**:



1. **Parse the response to JSON**:



1. **Assert** that the **email and password** are as expected:

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

Описанието е генерирано автоматично

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

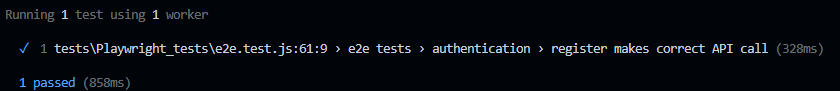
Картина, която съдържа текст, екранна снимка

Описанието е генерирано автоматично

1. **Use the command "npx playwright test e2e.test.js"** to execute the test and check if it passes or fails:



If the test passes, you should get a tick:



**IMPORTANT: Make sure that the back-end and http servers are active before executing the test. Also, keep in mind that some tests depend on others, so it's better to execute all tests, not one by one**.

### Playwright Test for "Register: Test Scenario 2: Register with Empty Fields"

1. Create a test scope:

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

Описанието е генерирано автоматично

1. **Go to** [**http://localhost:3000**](http://localhost:3000)

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

Описанието е генерирано автоматично

**IMPORTANT: The variable "host" holds the value "**[**http://localhost:3000**](http://localhost:3000)**," which we created earlier outside the test scope.**

1. **Click on the Register button**:

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

Описанието е генерирано автоматично

1. **Wait for the register form to load**:

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

Описанието е генерирано автоматично

1. **Assert that the URL of the page doesn’t change**, meaning the register request isn’t successful and we are still on the same page:



**NB: There is a dialog window (an alert with a message) that we do not check because Playwright will handle it automatically.**

1. Use the command "npx playwright test e2e.test.js" to execute the tests and check if they pass or fail.

### Playwright Test for "Login: Test Scenario 1: Login with Valid Credentials"

1. Create a test scope.
2. **Go to** [**http://localhost:3000**](http://localhost:3000) and **click on the Login button**:

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

Описанието е генерирано автоматично

1. **Wait for the login form to load**:



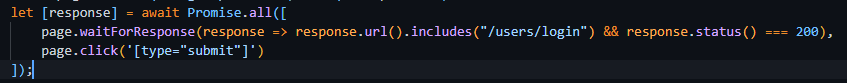
1. **Locate the input field for email** and **fill it with an existing email value**:



1. **Locate the input field for the password** and **fill it with the correct password**:



1. **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".** **Press the submit button** for the form. Both actions (response waiting and submit button) will be executed simultaneously:



1. **Assert that the response is okey**:



1. **Parse the response to JSON**:



1. You may want **to check the response JSON using "console.log()"**. When you execute the test, you can see the JSON in the Terminal:



1. **Assert that the user email and password** are as expected:



1. Use the command "npx playwright test e2e.test.js" to execute the tests and check if they pass or fail.

### Playwright Test for "Login: Test Scenario 2: Login with Empty Fields"

1. Create a test scope.
2. **Go to** [**http://localhost:3000**](http://localhost:3000) and **click the Login button.**
3. **Wait for the login form to load**.
4. **Assert that the URL of the page doesn’t change**, meaning the login request isn’t successful and we are still on the same page. Keep in mind that there will be a dialog window (an alert) which Playwright will handle automatically.
5. Use the command "npx playwright test e2e.test.js" to execute the tests and check if they pass or fail.
6. The whole test should look like this:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

### Playwright Test for " Logout: Test Scenario 1: Logout from the Application"

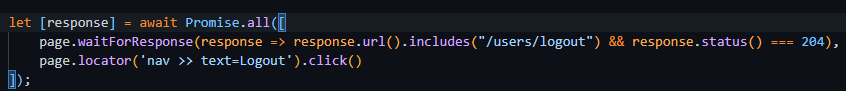
1. Create a test scope.
2. As **we start as a guest user**, first we need to log into the application:

* Go to <http://localhost:3000>
* Click the Login button.
* Wait for login form to load.
* Locate and fill the email and password input fields.
* Press the submit button of the form.
* Here is the code realization of this:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

1. **Create a Promise scope** that expects a response with **status code 204** and the **URL to contain "/users/logout"**. Then **get the response and set it in a variable**:



1. **Assert that the response is okey**.
2. After **pressing the Logout button**, we expect to be **redirected to the home page** with the navigation bar for guest users. **Wait for the Login button to load** on the page before **checking the URL**:



1. **Assert** that the **URL is for the home page**:



NOTE: Keep in mind that the "host" variable has the value <http://localhost:3000>.

1. Use the command "npx playwright test e2e.test.js" to execute the tests and check if they pass or fail.
2. The whole test should look like this:

Картина, която съдържа текст, екранна снимка, софтуер

Описанието е генерирано автоматично

After executing all tests, you should **get a report**:

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

Описанието е генерирано автоматично

NOTE: We wrote a console.log(json) in the login test with correct credentials, so during the test execution, we see the printed JSON.

## Write Front-End Navigation Bar Tests

We won't write user stories, test scenarios, or test case descriptions this time, but you can do it yourself if you wish to plan before writing the navigation bar tests. Let’s start with **creating of another test suit for out navigation tests**:

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

Описанието е генерирано автоматично**

### Playwright Test for Navigation Bar for Logged-In User

1. Create a test scope.
2. **Go to** [**http://localhost:3000**](http://localhost:3000)
3. **Log in to the application** (we need to execute authentication steps first to gather the user data we use here).
4. **Check that "All Games", "Create Games", and "Logout" buttons/links are visible**, and **"Login" and "Register" buttons/links are hidden**:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

1. Use the command "npx playwright test e2e.test.js" to execute the tests and check if they pass or fail. Keep in mind that you have to execute all tests, including authentication tests.

### Playwright Test for Navigation Bar for Guest User

1. Create a test scope.
2. **Go to** [**http://localhost:3000**](http://localhost:3000)
3. **Check that "All Games", "Create Games", and "Logout" buttons/links are hidden**, and **"Login" and "Register" buttons/links are visible**.
4. Use the command "npx playwright test e2e.test.js" to execute the tests and check if they pass or fail.

## Write Front-End CRUD Operations Tests

### Create a Test Suite for CRUD Operations Tests and BeforeEach Configuration

1. **Create a test suite for CRUD operations tests**, so we will have 3 test suites for now:

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

Описанието е генерирано автоматично

1. For **each test**, we are going to **need a login action**, so let’s **create a beforeEach scope** in this test suite where we will **perform these steps**:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

**By adding this beforeEach scope**, we will **perform these login user actions in each test** of the test suite, so we won’t need to write login steps in the tests. As you see from the picture, we are doing the following steps here:

1. Go to <http://localhost:3000>
2. Click on Login button/link.
3. Wait for login form to load.
4. Locate and fill the email input field.
5. Locate and fill the password input field
6. Press the submit button of the form.

**NB! We are using the user data from previous tests. So, to perform this action, we need to execute authentication tests first**.

### Playwright Test for Create a Game with Empty Fields

**IMPORTANT: As we execute the login steps in the beforeEach scope, we start this test case from the logged-in user home page. It will be the same for next tests.**

1. Create a test scope.
2. **Click on the "Create Game"** button/link.
3. **Wait for the create a game form to load**.
4. **Click on the submit form button** (we do not fill the input fields in this test case).
5. **Assert that the URL** is <http://localhost:3000/create>
6. Use the command "npx playwright test e2e.test.js" to execute the tests and check if they pass or fail.
7. The whole test should look like this:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

### Playwright Test for Create a Game with Valid Input Values

1. Create a test scope
2. **Click on the "Create Game" button/link**.
3. **Wait for the create a game form to load**.
4. **Fill the input fields** for **title**, **category**, **maxLevel**, **imageUrl**, and **summary** with data of your choice.
5. **Get the response by creating a Promise scope** in which we wait for a response with **status 200** and **URL containing "/data/games"** and **press the submit form button**. Both actions will be executed at the same time.
6. **Assert that the response is okey**.
7. **Parse the response to JSON** (you can check the JSON structure by using "console.log")
8. **Assert that title, category, maxLevel and summary** are as expected.
9. Use the command "npx playwright test e2e.test.js" to execute the tests and check if they pass or fail.
10. The whole test should look like this:

Картина, която съдържа текст, екранна снимка, софтуер

Описанието е генерирано автоматично

### Playwright Test for Edit/Delete Buttons for Owner

One of the application specifications is to **allow a user to create a game**. For each created game, **its owner can see the edit and delete buttons**. So, let’s write a test to check if this functionality works correctly for the game owner.

1. Create a test scope.
2. Go to <http://localhost:3000/catalog>
3. **Locate and click on the detail button** **of a game created by the current user**. Use a CSS selector to locate this button:

Картина, която съдържа екранна снимка, текст, Шрифт

Описанието е генерирано автоматично

NOTE: Replace the "Random title" value with the title you gave in the previous test for creating a game.

1. **Assert that the Delete button is visible**.
2. **Assert that the Edit button is visible**.
3. Use the command "npx playwright test e2e.test.js" to execute the tests and check if they pass or fail.
4. The whole test should look like this:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

### Playwright Test for Edit / Delete Button for Non-Owner

1. Create a test scope.
2. Go to <http://localhost:3000/catalog>
3. **Locate and click on the detail button** of a game that isn’t created by the current user. Choose a title of the game in the CSS selector that is not created by the current user.
4. **Assert that the Delete button is hidden**.
5. **Assert that the Edit button is hidden**.
6. Use the command "npx playwright test e2e.test.js" to execute the tests and check if they pass or fail.

### Playwright Test for Edit Button for Game Owner

1. Create a test scope.
2. Go to <http://localhost:3000/catalog>.
3. **Locate and click on the detail button** of a game created by the current user.
4. **Click on the Edit button**.
5. **Wait for the edit form to load**.
6. **Locate the title input field and give a different value than the game has now**. This is an edit form, so the input fields will be filled with game data. You must change at least one of the elements to check if the edit action is working as expected. Feel free to change all the game data by giving different values for all of the input fields.
7. **Get the response by creating a Promise scope** that waits for a **response with status code 200** and a **URL containing "/data/games"**. Also, there should be an **action for clicking the submit button**.
8. **Parse the response to JSON**.
9. **Assert that the title is changed**, and the category, maxLevel, and summary have the same values as at the time of game creation.
10. Use the command "npx playwright test e2e.test.js" to execute the tests and check if they pass or fail.
11. The whole test should look like this:

Картина, която съдържа текст, екранна снимка

Описанието е генерирано автоматично

### Playwright Test for Delete Button for Game Owner

1. Create a test scope.
2. Go to <http://localhost:3000/catalog>.
3. **Locate and click on the detail button** of a game created by the current user.
4. **Get the response by creating a Promise scope** that waits for a **response with status code 200** and a **URL containing "/data/games"**. Also, there should be an action for clicking the Delete button.
5. **Assert that the response is okey**.
6. The whole test should look like this:

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

Описанието е генерирано автоматично**

## Write Front-End Home Page Tests

The application's home page has the same functionality for both logged-in and guest users. Therefore, there will be a description only for testing the guest user home page. You can write almost the same test for logged-in users by simply adding the steps for login. Here are the steps for **testing the home page for guest users**:

1. Create a test scope (don't forget to create a test suite before that).
2. Go to <http://localhost:3000>
3. **Locate the welcome messages and assert their text**.
4. **Locate the game list title and assert its text**.
5. There is seeded data in the application, so **you will always get at least 3 already created games**. **Locate all game div elements and assert that there are three or more**.
6. Use the command "npx playwright test e2e.test.js" to execute the tests and to check if they pass or fail.
7. The whole test should look like this:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

**Write the home page test for logged-in users:** Use the test above and create another test but for logged users.

## Project Testing Summery and Conclusions

In our testing, we can notice **that tests depend on each other** because we are **using variables that are settled in one test and reused in another**. This request to **execute all the tests to ensure test passing**. Also, **we depend on two servers** for executing the tests:

* **Back-End server**, which we start with the command "npm run server".
* **HTTP server**, which we start with the command "npm start".

For the tests, we **had to locate different elements with CSS Selectors**, so it will be **helpful to use the browser** **and its DEV tools (F12)** to find the right CSS Selector:

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

Описанието е генерирано автоматично

Also, the **browser has an option that allows us to easily locate an element**:

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

Описанието е генерирано автоматично

You can use **the shortcut CTRL+F** to get a **search bar where you can write a CSS selector**:

Картина, която съдържа софтуер, Мултимедиен софтуер, екранна снимка

Описанието е генерирано автоматично

For example:

1. **Open the application in the browser** (both servers that we earlier noticed should be active) and **go to the "All Game" page**.

**IMPORTANT: There is a possibility to get debugging through the browser. If you first open the Dev tools, just press the "Continue" button until you load the page**:

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

Описанието е генерирано автоматично

1. **Open Dev tools by pressing F12** and **go to the "Elements" tab**.
2. Use **the shortcut CTRL+F to open the search bar**.
3. Type this CSS selector ".allGames .allGames-info .details-button":

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

Описанието е генерирано автоматично

1. Check the test that we wrote for edit/delete buttons for owner and non-owners. We used the same CSS selector there, but **to specify we added a pseudo selector ":has-text("Game title")"**. **This pseudo selector won’t be recognized by the browser, but Playwright will**, so we can get a DOM element with specific text value:

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

Описанието е генерирано автоматично

Also, as you saw in the test, we used a **browser URL link to go to a specific page, but to get a response, we did a request to the application API**. So, you can **find the browser link by opening the application in the browser** and simply navigating through the UI to the needed page:

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

Описанието е генерирано автоматично

On the other side, **to get the API URL, you need to search in the project files**. **You will find all URLs in the "api" folder**:

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

Описанието е генерирано автоматично

As you can see, there are **different JS files for each of the functionalities**. So, **when you search for the API URL for user functionalities (such as login, register, logout), you have to open the "users.js” file**. It is the **same for game and comment functionalities as logic**.

In summary, there are **14 tests in total** as you can see **from their execution and Playwright report**:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

And **they didn’t cover most of the application functionalities**. So, **feel free to make more tests to increase the application test coverage**.

## \*\*Make a Better Test Coverage

In the current practical project, we **didn’t test most of the application's functionalities**. **The idea of this practical project is to show you Playwright and how to use it**, so feel free to **write more tests to increase the coverage**. There are a **few hints on how and what you must do**:

1. It is possible to write additional tests such as:

* Registration with incorrect data.
* Registration with just one empty field.
* Registration where password and confirm password fields don’t match.
* Login with incorrect data.
* Login without password or email.
* Creating a game with a single empty field.
* Performing game editing with an empty input field.
* Etc.

1. Also, there aren’t tests for the comment functionality, so you can try to write tests for it.
2. Get more familiar with Playwright by using its documentation: [Fast and reliable end-to-end testing for modern web apps | Playwright](https://playwright.dev/)
3. Check the previous section "Project Testing Summary and Conclusions" – there are some hints for element location and URL finding of the application.
4. Use test debugging by adding "--debug" to the end of the execution command:



1. Use VS Code extensions. We are recommending the "Playwright Test for VSCode". With it you can debug and record tests. There are tutorials for it:

* How to install the extension: [Getting Started with Playwright and VS Code (youtube.com)](https://www.youtube.com/watch?v=Xz6lhEzgI5I)
* How to debug through the extension: [Debugging Playwright tests in VS Code (youtube.com)](https://www.youtube.com/watch?v=tJF7UhA59Gc)
* How to record/generate test through the extension: [Generating Playwright Tests in VS Code (youtube.com)](https://www.youtube.com/watch?v=LM4yqrOzmFE)

1. Enjoy the tests writing 😊