# **Practical Project: Automated Testing with Playwright**

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.

## 1. Preparing the Environment

To initialize the application, execute the following commands in the Visual Studio Code terminal:

|  |
| --- |
| **npm install** |

This will install the **Playwright** framework and other required packages.

Then, start the server by using the following command:

|  |
| --- |
| **npm run start** |

Then open another CLI and type this command:

|  |
| --- |
| **npm run server** |

Navigate to <http://localhost:3000/> to start using the application.

After that, create a new folder, named "**tests**" and in it, create a new file, named "**e2e.test.js**". This file will contain the UI tests that we'll write using **Playwright**.

**NOTE:** To execute the command for running the Playwright tests, just open a new terminal in Visual Studio code and write the following:

|  |
| --- |
| **npm test** |

## 2. Application Specifications

### 2.1 All Games Page (Catalogue)

This page displays a list of all games in the system, with their title and category. Clicking on any of the cards leads to the details page for the selected game.

A screenshot of a video game

Description automatically generated

If there are nogames, the following view is displayed:

A screenshot of a video game

Description automatically generated

### 2.2 Create Game

The Create page is available to **logged-in** users. It contains a form for creating new games. All the fields are required.

A screenshot of a computer

Description automatically generated

### 2.3 Details

All users should be able to **view details** aboutgames. Clicking the **Details** link in of a **game** should **display** the **Details** page:

* If the currently logged in user is the **creator** of the game, the **Edit** and **Delete** buttons should be displayed, otherwise they should not be available.
* The view for the **creators** should look like:

A screenshot of a video game

Description automatically generated

The view for **guests** and **logged-in users** should look like:

A screenshot of a video game

Description automatically generated

### 2.4 Edit Game

The **Edit page** is accessible to logged-in users and allows the author to **edit** their own games. Clicking the **Edit** a specific game link on the **details page** displays the Edit page. It contains a form with input fields for all relevant properties.

A screenshot of a video game

Description automatically generated

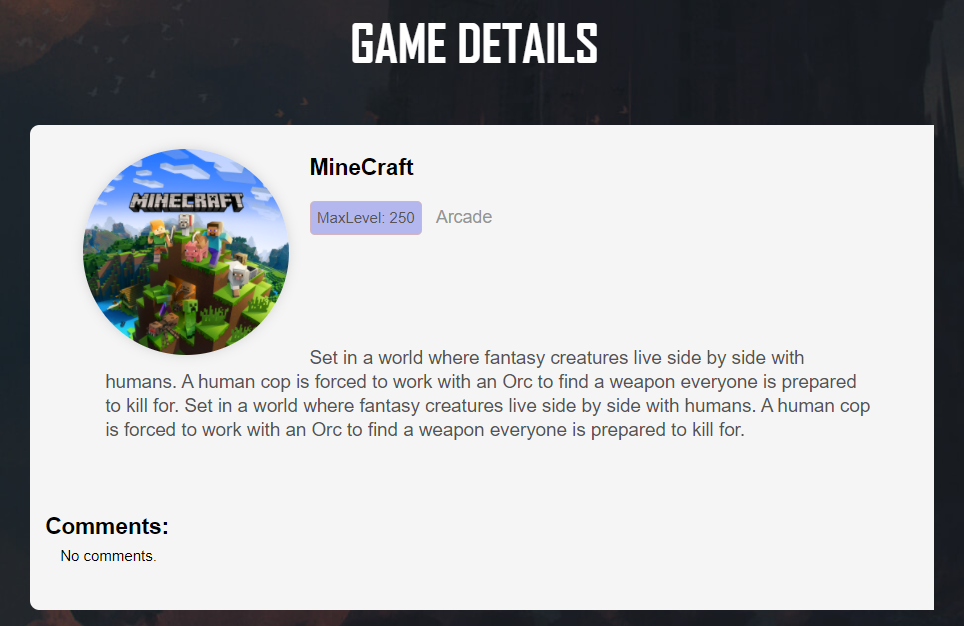
### 2.4 Delete Game

The delete action is available to logged-in users, for games they have created. When the author clicks on the Delete action on any of their games, a confirmation dialog is displayed, and upon confirming this dialog, the game is deleted from the system.

### 2.5 Comments

Every logged-in user is able to **comments other games**, but **not his own**.

**Guest** are not able to see the section **Add new comment**, but are able to see the section **Comments** The view for **guests** should look like:



If there are **no comments all** users see:

A white screen with a black border

Description automatically generated

If there are **comments all** users see:

A purple rectangle with white text

Description automatically generated

Logged-in users see a **form** for adding a new comment. Every **registered** user can leave a **comment** under any games. **Authors** can't comment on their **own** games.

The view when there are no comments yet and the user did not press **[Add Comment]** button looks like:

A screenshot of a video game

Description automatically generated

The view when the logged-in user adds **Comment** to the game looks like this:

A screenshot of a computer

Description automatically generated

## 3. Set up Playwright and Create Variables

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.

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 game variable** to hold the game data. We can reuse this for tests where it is needed:

A computer screen with text

Description automatically generated

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

## 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. **Assert the user is redirected to the Home page**
9. **Assert the user is logged in**

* **Expected Result**: The user is registered successfully.
* **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. **Assert the user is redirected to the Home page**
8. **Assert the user is logged in**

* **Expected Result**: The user is logged in successfully.
* **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**:

A black screen with white text

Description automatically generated

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



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



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



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:

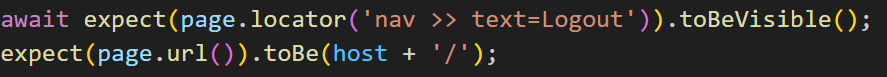


1. **Click** the submit button:

A black background with orange letters

Description automatically generated

1. Assert that the url is changed and the Logout button is visible:



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

A computer screen shot of a program code

Description automatically generated

1. Use the command "npm test" to execute the tests and check if they pass or fail.

### 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

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

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



1. Use the command "npm test" 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**:

A black background with white text

Description automatically generated

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**:

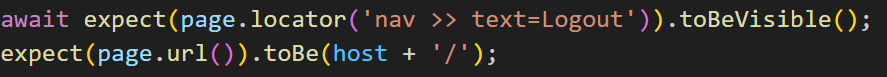


1. **Click the submit button**:

A black background with orange letters

Description automatically generated

1. Assert that the url is changed and the Logout button is visible:



1. Use the command "npm test" 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. The whole test should look like this:

A computer code with text

Description automatically generated with medium confidence

1. Use the command "npm test" to execute the tests and check if they pass or fail.

### 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:

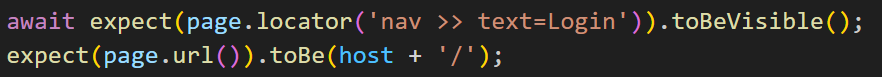
Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

1. **Press the Logout button**:



1. **Assert** that the **URL is for the home page and the Login button is visible**:



1. Use the command "npm test" to execute the tests and check if they pass or fail.

## 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 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 "npm test" to execute the tests and check if they pass or fail.

### 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 "npm test" 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 "npm test" to execute the tests and check if they pass or fail.
7. The whole test should look like this:

A computer code with text

Description automatically generated

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

1. Create a test scope
2. **Create unique game title and game category** using a **random value** and **set it for the game object created earlier outside the test scope**:

A black rectangular with yellow and blue text

Description automatically generated with medium confidence

1. **Click on the "Create Game" button/link**.
2. **Wait for the create a game form to load**.
3. **Fill the input fields** for **title**, **category**, **maxLevel**, **imageUrl**, and **summary** with data from our game object.
4. **Press the submit form button**
5. **Assert the url is the Home page**.
6. **Assert that a book with the title we generated is present in the list.**
7. Use the command "npm test" to execute the tests and check if they pass or fail.
8. The whole test should look like this:

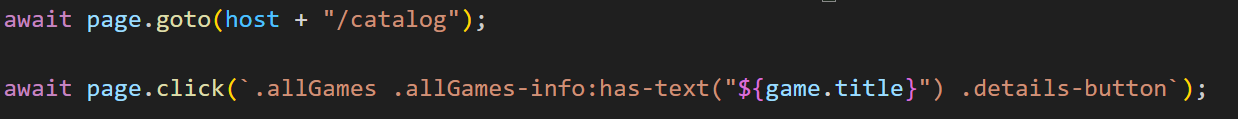
A screen shot of a computer program

Description automatically generated

### 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 the game created in the previous test**. Use a CSS selector to locate this button:



1. After we are redirected to the Details page, we can save the id of the game we created for other purposes. This is done by extracting it from the url:



1. **Assert that the Delete button is visible**.
2. **Assert that the Edit button is visible**.
3. Use the command "npm test" to execute the tests and check if they pass or fail.

### 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 "npm test" 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 the game we created.
4. **Click on the Edit button**.
5. **Wait for the edit form to load**.
6. **Modify the title of the game object we use. 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:

A screen shot of a computer code

Description automatically generated

1. **Click the submit button**.
2. **Assert that the title is changed and the url is redirected to the details page**.

**NB! We are using the game id from previous tests to verify the url is correct.**

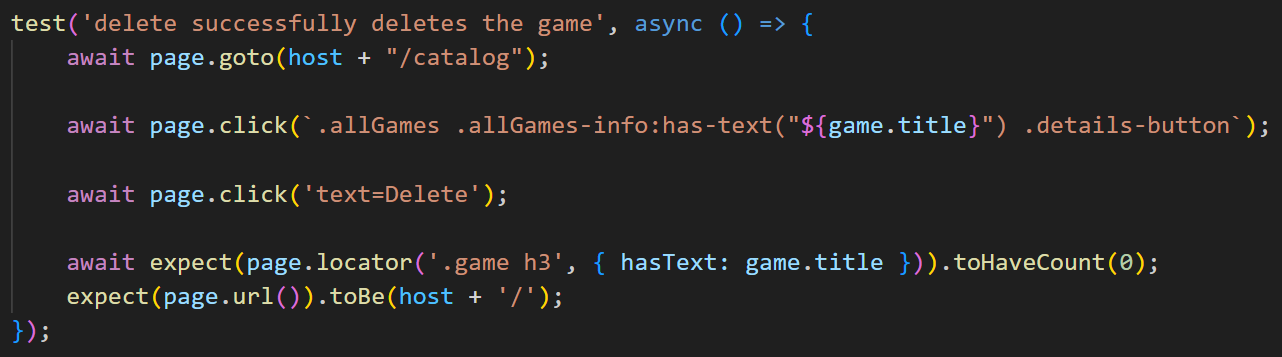
1. Use the command "npm test" to execute the tests and check if they pass or fail.
2. The whole test should look like this:

A screen shot of a computer program

Description automatically generated

### 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 Details button** of the game we created.
4. **Locate and click on the Delete button**.
5. **Assert** that we are **redirected to the Home page** and the **game we deleted is not present in the list**.
6. The whole test should look like this:



## 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 "npm test" to execute the tests and check if they pass or fail.
7. The whole test should look like this:

A screen shot of a computer code

Description automatically generated

## Project Testing Summary 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**. We need 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**.
2. **Open Dev tools by pressing F12** and **go to the "Elements" tab**.
3. Use **the shortcut CTRL+F to open the search bar**.
4. 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:

A screen shot of a computer code

Description automatically generated

Also, as you saw in the test, we used a **browser URL link to go to a specific page**. 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

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

A screen shot of a computer code

Description automatically generated

And **they didn’t cover most of the application functionalities**. So, **feel free to do 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. 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 tests through the extension: [Generating Playwright Tests in VS Code (youtube.com)](https://www.youtube.com/watch?v=LM4yqrOzmFE)

1. Enjoy the test writing 😊