To automate the testing of the requirements using JavaScript, BDD, and CodeceptJS, you can follow these steps. First, ensure you have Node.js and CodeceptJS installed. You'll also need a CodeceptJS project set up. Then, create a test suite for the Oranum website.

Step1: Create a CodeceptJS project (if not already done):

npx codeceptjs init

Step2: Install the required dependencies for CodeceptJS:

npm install codeceptjs puppeteer –save

Step3: Create a new feature file for your test suite. Let's call it **oranum\_search.feature**:

Gherkin:

Feature('Oranum Search');

Scenario('Searching for partial text should display matching psychics', (I) => {

I.amOnPage('https://www.oranum.com');

I.seeInTitle('Oranum');

// Test case 1: Search for 'Matt'

I.fillField({ css: 'input[name="search"]' }, 'Matt');

I.pressKey('Enter');

I.see('Matt');

// Test case 2: Search for 'Myst'

I.fillField({ css: 'input[name="search"]' }, 'Myst');

I.pressKey('Enter');

I.see('Myst');

// Test case 3: Search for 'Ann'

I.fillField({ css: 'input[name="search"]' }, 'Ann');

I.pressKey('Enter');

I.see('Ann');

// Test case 4: Search for 'psy'

I.fillField({ css: 'input[name="search"]' }, 'psy');

I.pressKey('Enter');

I.see('psy');

});

Step 4: Create step definitions for your feature by generating them:

npx codeceptjs gherkin:snippets

Step6: Implement the generated step definitions in the **oranum\_search\_steps.js** file. Modify the steps to interact with the Oranum website as needed.

const { I } = inject();

Given('I am on the Oranum website', () => {

I.amOnPage('https://www.oranum.com');

I.seeInTitle('Oranum');

});

When('I search for {string}', (searchText) => {

I.fillField({ css: 'input[name="search"]' }, searchText);

I.pressKey('Enter');

});

Then('I should see {string} in the results', (resultText) => {

I.see(resultText);

});

Step5: Now, you can run the tests using CodeceptJS:

npx codeceptjs run –steps

Ensure that you have Docker installed and running if you plan to run these tests in a Dockerized environment. You can create a Dockerfile and a Makefile for easier containerization and execution, but they are not mandatory.

Here's a simple Dockerfile to run your CodeceptJS tests in a Docker container:

# Use an official Node.js runtime as a parent image

FROM node:14

# Set the working directory in the container

WORKDIR /app

# Copy package.json and package-lock.json to the working directory

COPY package\*.json ./

# Install CodeceptJS and Puppeteer

RUN npm install

# Copy the CodeceptJS project to the container

COPY . .

# Expose the default port used by CodeceptJS (if applicable)

# EXPOSE 8080

# Run the tests

CMD ["npx", "codeceptjs", "run", "--steps"]

Step6: Here's a simple Dockerfile to run your CodeceptJS tests in a Docker container:

And here's a simple Makefile to build and run the Docker container:

build:

docker build -t oranum-test .

run:

docker run --rm oranum-test

With these files, you can build the Docker image using **make build** and run the tests in a Docker container using **make run**.

REQUIRMENT2

I'll provide you with a feature file and step definitions to validate the buttons on the provided URL.

Step 1: Create a new feature file for your test suite. Let's call it **oranum\_buttons.feature**:

Feature('Oranum Buttons Validation');

Scenario('Open the livestream of a psychic and validate overlay buttons', (I) => {

I.amOnPage('https://oranum.com/en/chat/LovePsychyicAnie');

I.waitForText('LovePsychyicAnie', 15);

I.see('Live', { css: '.label-live' });

// Validate 'Get Credits' button

I.click({ css: 'button[data-action="purchaseCredits"]' });

I.see('Sign Up', { css: '.modal-title' });

I.click({ css: '.modal-close' });

// Validate 'Add to favorites' button

I.click({ css: 'button[data-action="addFavorite"]' });

I.see('Sign Up', { css: '.modal-title' });

I.click({ css: '.modal-close' });

// Validate 'Surprise' button

I.click({ css: 'button[data-action="surprise"]' });

I.see('Sign Up', { css: '.modal-title' });

I.click({ css: '.modal-close' });

// Validate 'Start Session' button

I.click({ css: 'button[data-action="startSession"]' });

I.see('Sign Up', { css: '.modal-title' });

I.click({ css: '.modal-close' });

// Validate 'Get Coins' button

I.click({ css: 'button[data-action="getCoins"]' });

I.see('Sign Up', { css: '.modal-title' });

I.click({ css: '.modal-close' });

});

Step2: Create step definitions for your feature by generating them:

npx codeceptjs gherkin:snippets

Step2: Implement the generated step definitions in the **oranum\_buttons\_steps.js** file. Modify the steps to interact with the Oranum website as needed.  
  
  
  
 const { I } = inject();

Given('I am on the psychic's livestream page', () => {

I.amOnPage('https://oranum.com/en/chat/LovePsychyicAnie');

I.waitForText('LovePsychyicAnie', 15);

I.see('Live', { css: '.label-live' });

});

When('I click on the {string} button', (buttonName) => {

const buttonSelectors = {

'Get Credits': 'button[data-action="purchaseCredits"]',

'Add to favorites': 'button[data-action="addFavorite"]',

Surprise: 'button[data-action="surprise"]',

'Start Session': 'button[data-action="startSession"]',

'Get Coins': 'button[data-action="getCoins"]',

};

I.click({ css: buttonSelectors[buttonName] });

});

Then('I should see a "Sign Up" overlay', () => {

I.see('Sign Up', { css: '.modal-title' });

I.click({ css: '.modal-close' });

});

Step3: Run the tests using CodeceptJS:  
  
npx codeceptjs run –steps

To automate the testing of REQ-3 using JavaScript, BDD, and CodeceptJS, you can create a new feature file and step definitions to validate the topics on the home page of the Oranum website.

Step1: Create a new feature file for your test suite. Let's call it **oranum\_topics.feature**:  
  
  
  
 Feature('Oranum Topics Validation');

Scenario Outline('Selecting <Topic> should display matching psychics without duplicates', (I, current) => {

I.amOnPage('https://www.oranum.com');

I.waitForText('Oranum', 15);

I.click(current.topic);

I.wait(3); // Adjust the wait time as needed

I.see(current.topic);

// Validate that no duplicate content is displayed

const psychicNames = I.grabTextFrom({ css: '.live-psychic .nickname' });

const uniquePsychicNames = [...new Set(psychicNames)];

I.assertEqual(psychicNames.length, uniquePsychicNames.length);

}).tag('@topics');

Examples:

| Topic |

| Love |

| Clairvoyant |

| Tarot |

| Astrology |

| Dreams |

| Guides |

| Family |

| Career |

| Fortune Teller |

| Numerology |

| Sounds Baths |

| Pet Psychic |

Step2: Create step definitions for your feature by generating them:

npx codeceptjs gherkin:snippets

Step3: mplement the generated step definitions in the **oranum\_topics\_steps.js** file. Modify the steps to interact with the Oranum website as needed.

const { I } = inject();

Given('I am on the Oranum home page', () => {

I.amOnPage('https://www.oranum.com');

I.waitForText('Oranum', 15);

});

When('I click on {string}', (topic) => {

I.click(topic);

I.wait(3); // Adjust the wait time as needed

});

Then('I should see matching psychics without duplicates', () => {

const psychicNames = I.grabTextFrom({ css: '.live-psychic .nickname' });

const uniquePsychicNames = [...new Set(psychicNames)];

I.assertEqual(psychicNames.length, uniquePsychicNames.length);

});

Step4: Run the tests using CodeceptJS:

npx codeceptjs run --steps --grep @topics

This will execute the tests for each topic and validate that selecting a topic displays matching psychics without duplicates on the Oranum home page. Adjust the wait time as needed to allow the page to load.