Test Automation Assignment Documentation

Setup Instructions

Following framework was implemented using JavaScript based tool Cypress.

1. **Install Cypress:** If Cypress isn't installed yet, you can add it by running this command in your project directory:

npm install cypress --save-dev

2. **Run Cypress**: Open Cypress Test Runner by running:

npx cypress open

3. **Add the Test File:** Create the test file (e.g., openWeatherTest.js) in the "`cypress/integration`" directory of your project.

Cypress Configuration File Documentation

This configuration file (`cypress/support/e2e.js`) is processed and loaded automatically before your test files. It is used to set up global configuration and behavior that modifies Cypress. Below is a detailed explanation of each part of the configuration.

File Path

- **Path:** `cypress/support/e2e.js`
- **Description:** This file is for setting up global configurations and behaviors for Cypress end-to-end tests.

File Content

Import Custom Commands

- **Code:** `import './commands'`
- **Description:** Imports custom commands from the `commands.js` file using ES2015 syntax. This allows you to define custom commands that can be used throughout your test suite.

Handling Uncaught Exceptions

- **Code:**

```
Cypress.on('uncaught:exception', (err, runnable) => {
  // returning false here prevents Cypress from
  // failing the test
  return false
});
```

Description: This code sets up an event listener for uncaught exceptions. If an uncaught exception occurs during a test, returning false prevents Cypress from failing the test. This can be useful for handling non-critical errors that should not cause the test to fail.

CommonJS Syntax Alternative

- **Code:** `require('./commands')`
- **Description:** This line shows how to import the `commands.js` file using CommonJS syntax. It is commented out because the ES2015 syntax is already used above.

Summary

This configuration file (`cypress/support/e2e.js`) includes:

1. Importing custom commands from `commands.js`.
2. Setting up an event listener to handle uncaught exceptions and prevent them from failing the tests.

These settings help to ensure that your Cypress tests have the necessary custom commands available and can handle uncaught exceptions gracefully without causing unnecessary test failures.

Additional Information

- **Custom Commands:** You can define custom commands in the `cypress/support/commands.js` file. These commands can be used throughout your test suite to simplify repetitive tasks.
- **Handling Exceptions:** While the provided exception handling prevents test failures due to uncaught exceptions, it is important to log or handle these exceptions appropriately within your tests to ensure they do not mask actual issues.

For more information on Cypress configuration, visit the Cypress Configuration Documentation.

AUTOMATED UI FLOW

Cypress Test Documentation

Purpose

This Cypress test script is designed to automate the process of logging into the OpenWeatherMap website, generating a new API key, editing the API key, and saving the API key to a file.

File

Path: `cypress/integration/openWeatherTest.cy.js`

Test Suite:

This test suite includes a single test case that performs the following steps:

Test Case:

Log in, generate an API key, edit it, and save the key

Step 1: Visit the OpenWeatherMap home page

- **Code**: `cy.visit('https://home.openweathermap.org/');`
- **Description**: This command navigates the browser to the OpenWeatherMap home page.

Step 2: Fill in the login form

- **Email Input:**
  - **Code:**

```
cy.get('input[placeholder="Enter email"]')
  .first()
  .click()
  .type('OpenWeatherTest@proton.me');
```

  - **Description**: Selects the email input field, clicks it, and types in the email address `OpenWeatherTest@proton.me`.

- **Password Input:**

- **Code:**

```
cy.get('input[placeholder="Password"]')
  .first()
  .click()
  .type('Abcdefg123!');
```

  - **Description**: Selects the password input field, clicks it, and types in the password `Abcdefg123!`.

Step 3: Submit the login form

- **Code**: `cy.get('input[value="Submit"]').click();`
- **Description**: Clicks the submit button to log in to the OpenWeatherMap account.

Step 4: Navigate to the API keys tab

- **Code:** `cy.contains('API keys').click({ force: true });`
- **Description:** Clicks on the "API keys" tab to navigate to the API keys management page.

Step 5: Confirm that the API keys tab is displayed

- **Code:** `cy.url().should('include', '/api_keys');`
- **Description:** Verifies that the URL includes /api_keys, confirming that the API keys page is displayed.

Step 6: Enter key name and click generate

- **Key Name Input:**
  - **Code:** `cy.get('input[placeholder="API key name"]').type('Automated_test_key');`
  - **Description**: Enters the name `Automated_test_key` into the API key name input field.
- **Generate Key:**
  - **Code:** `cy.get('input[value="Generate"]').click();`
  - **Description:** Clicks the generate button to create a new API key.

Step 7: Validate if key was created

- **Code:** `cy.contains('Automated_test_key').should('exist');`
- **Description:** Verifies that the newly created API key with the name `Automated_test_key` exists on the page.

Step 8: Click Edit and change key name

- **Select Edit:**
  - **Code:**

```
cy.contains('Automated_test_key');

cy.get('[class="fa fa-edit"]').first().click();

cy.contains('Automated_test_key');

cy.get('[class="fa fa-edit"]').first().click();
```

Description: Clicks the edit icon next to the `Automated_test_key`.

- **Edit Key Name:**

- **Code:**

```
cy.get('input[id="edit_key_form_name"]').clear().type('Ed
ited_test_key');

cy.contains('Save').click();
```

Description: Clears the existing key name, enters `Edited_test_key`, and saves the changes

Step 9: Validate if key name was saved

- **Code:** `cy.contains('Edited_test_key').should('exist');`
- **Description:** Verifies that the API key name has been successfully updated to `Edited_test_key`.

Step 10: Save key in the framework repository

- **Code**

```
cy.contains('Edited_test_key');

cy.get('pre').first()

  .invoke('text').then((apiKey) => {

    cy.writeFile('cypress/fixtures/apiKey.json', {
apiKey: apiKey });

  });
```

Description: Retrieves the text of the first `<pre>` element (which contains the API key) and writes it to the `apiKey.json` file in the `cypress/fixtures` directory.

Summary

This Cypress test script automates the entire process of logging into OpenWeatherMap, generating an API key, editing it, and saving the API key for later use. It ensures that the key management functionalities are working correctly and that the key is stored properly for future tests or usage.

AUTOMATED API FLOW

Cypress Test Documentation

Purpose

The purpose of this test suite is to verify the functionality of the OpenWeatherMap API by performing both positive and negative tests. These tests ensure that the API returns the expected results for valid requests and appropriate error messages for invalid requests. This validation helps in maintaining the reliability and accuracy of the weather data provided by the API.

File structure:

1. **Path:** `cypress/integration/openWeatherAPI.cy.js`
   Description: Contains the test cases.
2. **Path:** `cypress/fixtures/apiKey.json`
   Description: Contains your OpenWeatherMap API key.

Test Suite:

This test suite includes two Positive tests and three negative tests that performs the following steps:

Test Cases:

Positive Tests:

1. **Fetch current weather data for a valid city:**
   - **Code:**

```
// Positive test: Fetch current weather for a valid city
    it('should fetch current weather data for a valid city', () => {
        const city = 'London';
        cy.request({
            method: 'GET',
            url:
`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}`
        }).then((response) => {
            expect(response.status).to.eq(200);
            expect(response.body).to.have.property('weather');
            expect(response.body).to.have.property('main');
            expect(response.body.name).to.eq(city);
        });
    });
```

- **Description:** This test verifies that the API returns the correct weather data for a valid city (e.g., London).
- **Assertions:**
  - Status code should be 200.
  - Response should contain the `weather` and `main` properties.
  - The city name in the response should match the requested city.

2. **Fetch current weather data using geographic coordinates:**
   - **Code:**

```
// Positive test: Fetch current weather using geographic coordinates
    it('should fetch current weather data for valid geographic coordinates', () =>
{
        const lat = 51.5073;
        const lon = -0.1276; // Coordinates for London
        cy.request({
            method: 'GET',
            url:
`https://api.openweathermap.org/data/2.5/weather?lat=${lat}&lon=${lon}&appid=${apiK
ey}`
        }).then((response) => {
            expect(response.status).to.eq(200);
            expect(response.body).to.have.property('weather');
            expect(response.body).to.have.property('main');
            expect(response.body.coord.lat).to.eq(lat);
            expect(response.body.coord.lon).to.eq(lon);
        });
    });
```

- **Description:** This test verifies that the API returns the correct weather data for valid geographic coordinates (e.g., latitude and longitude for London)
- **Assertions:**
  - Status code should be 200.
  - Response should contain the `weather` and `main` properties.
  - The coordinates in the response should match the requested coordinates.

Negative Tests:

1. **Fetch current weather data for an invalid city:**

- **Code:**

```
// Negative test: Fetch current weather for an invalid city
    it('should return an error for an invalid city', () => {
        const invalidCity = 'InvalidCityName';
        cy.request({
            method: 'GET',
            url:
`https://api.openweathermap.org/data/2.5/weather?q=${invalidCity}&appid=${apiKey}`,
            failOnStatusCode: false
        }).then((response) => {
            expect(response.status).to.eq(404);
            expect(response.body).to.have.property('message', 'city not found');
        });
    });
```

- **Description:** This test verifies that the API returns an error for an invalid city name.
- **Assertions:**
  - Status code should be 404.
  - Response should contain a `message` property with the value `city not found`.

2. **Fetch current weather data with an invalid API key**:

- **Code:**

```
// Negative test: Fetch current weather with an invalid API key
    it('should return an error for an invalid API key', () => {
        const city = 'London';
        const invalidApiKey = 'InvalidApiKey';
        cy.request({
            method: 'GET',
            url:
`https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${invalidApiKey}`,
            failOnStatusCode: false
        }).then((response) => {
            expect(response.status).to.eq(401);
            expect(response.body).to.have.property('message', 'Invalid API key.
Please see https://openweathermap.org/faq#error401 for more info.');
        });
    });
```

- **Description:** This test verifies that the API returns an error when using an invalid API key.
- **Assertions:**
  - Status code should be 401.
  - Response should contain a `message` property with a message indicating an invalid API key.

3. **Fetch current weather data with missing required parameters**:

- **Code:**

```
// Negative test: Fetch current weather with missing required parameters
    it('should return an error when required parameters are missing', () => {
        cy.request({
            method: 'GET',
            url: `https://api.openweathermap.org/data/2.5/weather?appid=${apiKey}`,
            failOnStatusCode: false
        }).then((response) => {
            expect(response.status).to.eq(400);
            expect(response.body).to.have.property('message', 'Nothing to
geocode');
        });
    });
```

- **Description:** This test verifies that the API returns an error when required parameters (e.g., city name or coordinates) are missing.
- **Assertions:**
  - Status code should be 400.
  - Response should contain a `message` property with the value `Nothing to geocode`.

Summary

This documentation outlines a Cypress test suite designed to validate the functionality of the OpenWeatherMap API. The test suite includes both positive and negative tests to ensure that the API returns correct weather data for valid requests and appropriate error messages for invalid requests. The tests cover scenarios such as fetching weather data for valid cities, using geographic coordinates, and handling invalid city names or API keys. This comprehensive testing approach helps in maintaining the reliability and accuracy of the weather data provided by the API.