Generate Test Cases for Specific Use Case Using Generative AI

Estimated time needed: 30 minutes

As a software developer, you may be very comfortable with coding. Code should be tested for functionality across multiple scenarios as and when created. Test-driven development (TDD) is one of the proven methodologies for ensuring that the code works as expected. It is an approach to software development in which you first write the test cases for the code you *wish* you had and then write the code to make the test cases pass. Visit this link on TDD to learn more. In the past, writing the correct test cases was challenging and required expertise and experience. Now, you can avail the help of Generative AI to generate test cases for specific use cases.

Please remember that the prompts you feed to Generative AI are like a conversation with a subject matter expert and the consecutive prompts depend on the previous prompts and the response received. Change the wording when required to get a specific desired result. The example showcases one possible chat conversation to attain the objective.

Learning objectives

At the end of the lab you will be able to:

1. Use Generative AI to generate test cases for specific use cases.
2. Write test cases adhering to the laws of TDD.

Prerequisites

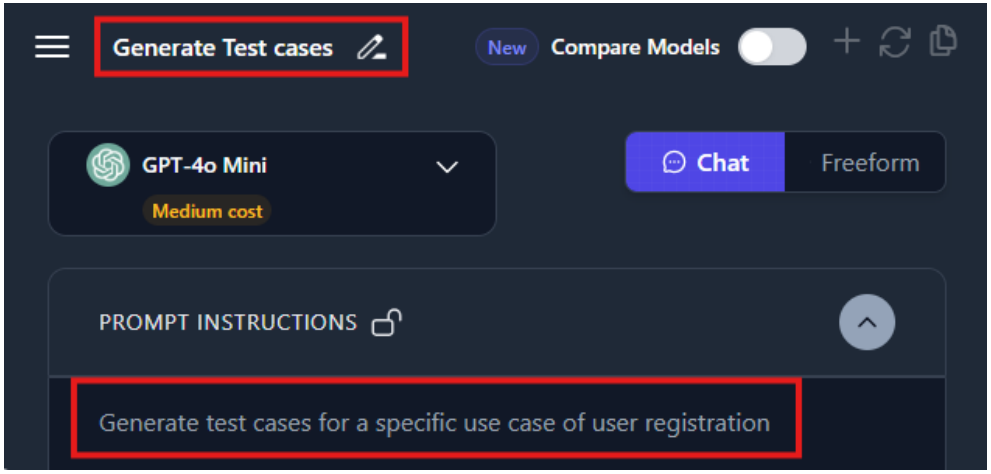
It will be good if you know how to code in any of the programming languages and have the ability to think from the perspective of software design.

Set up chat title and prompt instruction

When the chat session opens, a new chat conversation begins. Give the chat conversation an appropriate title. This will help you revisit the chat conversation. It is a good practice to segregate the conversations topically as it will help you continue the conversation later.

Also provide a prompt instruction which is specific to the conversation in this particular lab. Let's get started with the task of creating software documentation for the JavaScript code.

> Please note Generative AI is an evolving field. As you attempt the labs, your experience and output might be different than what is seen here.



Tool capability to generate test cases

You have been provided with the “User registration” module of the software and tasked with creating a test case for validating the registration. You decide to optimally utilize your time by using Gen AI to write the documentation.

Firstly, you need to determine if the Gen AI tool has the capability to do the required software documentation for JavaScript. In the prompt window, type the following:

```
I have a user registration module in my software. Can you help me create a test case for a specific use case?
```

If the GenAI model has the ability to generate test case, you will get a affirmative response in line with what is shown in the image below.You will need to use your judgment to perceive the confidence in the response, to decide on continuing to use the tool either entirely or use it as a support tool. Iterative prompts can be used to ensure the correctness of the responses.

▼ Click here to view the sample response generated

Of course! Please provide details about the specific use case for the user registration module, such as the required fields, validation rules, and any specific scenarios you want to test. This will help me generate relevant test cases for you.

Provide the use case to Gen AI

Paste the following in the prompt window to state the use case for which you need the test case.

```
Help me create a test case for validation of the user registration information.
```

The output could be something similar to what is shown in this image:

▼ Click here to view the sample response generated

Sure! Here’s a test case focused on validating user registration information.

Test Case: Validate User Registration Information

Test Case ID: TC_REG_001
Test Case Title: Validate Registration Fields for User Registration
Objective: To ensure that the user registration form correctly validates the input fields.
Preconditions: User is on the registration page of the application.

Test Steps:

1. **Input Valid Data:**
    - Fill in the registration form with valid data:
        - Username: “validUser123”
        - Email: “validemail@example.com“
        - Password: “StrongPassword!123”
    - Click the “Register” button.
    - **Expected Result:** Registration is successful, and the user is redirected to the welcome page.
2. **Input Invalid Email:**
    - Fill in the registration form with:
        - Username: “testUser”
        - Email: “invalidemail.com”
        - Password: “StrongPassword!123”
    - Click the “Register” button.
    - **Expected Result:** An error message is displayed indicating that the email format is invalid.
3. **Input Short Password:**
    - Fill in the registration form with:
        - Username: “testUser”
        - Email: “testuser@example.com“
        - Password: “short”
    - Click the “Register” button.
    - **Expected Result:** An error message is displayed indicating that the password must be at least 8 characters long.
4. **Input Missing Required Fields:**
    - Leave the email field blank and fill in:
        - Username: “testUser”
        - Password: “StrongPassword!123”
    - Click the “Register” button.
    - **Expected Result:** An error message is displayed indicating that the email field is required.
5. **Input Duplicate Username:**
    - Fill in the registration form with:
        - Username: “existingUser” (assuming this username already exists in the database)
        - Email: “newemail@example.com“
        - Password: “StrongPassword!123”
    - Click the “Register” button.
    - **Expected Result:** An error message is displayed indicating that the username is already taken.
6. **Input Weak Password:**

- Fill in the registration form with:
  - Username: “testUser”
  - Email: “testuser@example.com“
  - Password: “12345678”
- Click the “Register” button.
- **Expected Result:** An error message is displayed indicating that the password must include uppercase letters, lowercase letters, numbers, and special characters.

Postconditions:

- The user is not registered if any validation fails.
- The user is registered successfully if all validations pass.

Additional Considerations:

- Ensure that the error messages are clear and user-friendly.
- Test for edge cases, such as extremely long usernames or passwords.
- Check for SQL injection vulnerabilities in the input fields.
- Validate against XSS (Cross-Site Scripting) by entering script tags in the input fields.

Feel free to adjust the details based on your specific application requirements or any additional scenarios you want to include!

This response assumed that you needed the test case for manual testing with the provided GUI. If you need test case for testing the API, you need to be more specific and provide a context to generative AI for it to give the kind of response you require.

Refined test case generation prompt

What you need are the various scenarios that you must test to validate the registration information. In the prompt window, type the following:

```
Give me the test cases for registration validation, taking into consideration various possible scenarios and the required input and outcome to test for.
```

The output could be something similar to what is shown in this image:

▼ Click here to view the sample response generated

Here are comprehensive test cases for user registration validation, covering various scenarios, required inputs, and expected outcomes:

Test Case 1: Valid User Registration Information

- **Test Case ID**: TC001
- **Objective**: Verify successful registration with valid information.

- **Preconditions**: User is on the registration page.
- **Test Steps**:
  1. Enter a valid username (e.g., “validUser123”).
  2. Enter a valid email address (e.g., “validuser@example.com“).
  3. Enter a valid password (e.g., “StrongPass!123”).
  4. Click on the “Register” button.
- **Expected Result**: User registration is successful, and the user is redirected to a welcome page or receives a success message.

Test Case 2: Invalid Username (Too Short)

- **Test Case ID**: TC002
- **Objective**: Verify that the system rejects usernames that are too short.
- **Preconditions**: User is on the registration page.
- **Test Steps**:
  1. Enter an invalid username (e.g., “ab”).
  2. Enter a valid email address.
  3. Enter a valid password.
  4. Click on the “Register” button.
- **Expected Result**: An error message is displayed indicating that the username must be at least 3 characters long.

Test Case 3: Invalid Username (Special Characters)

- **Test Case ID**: TC003
- **Objective**: Verify that the system rejects usernames containing special characters.
- **Preconditions**: User is on the registration page.
- **Test Steps**:
  1. Enter an invalid username (e.g., “user@name”).
  2. Enter a valid email address.
  3. Enter a valid password.
  4. Click on the “Register” button.
- **Expected Result**: An error message is displayed indicating that the username cannot contain special characters.

Test Case 4: Invalid Email Format

- **Test Case ID**: TC004
- **Objective**: Verify that the system rejects invalid email formats.
- **Preconditions**: User is on the registration page.
- **Test Steps**:
  1. Enter a valid username.
  2. Enter an invalid email address (e.g., “invalidemail.com”).
  3. Enter a valid password.
  4. Click on the “Register” button.
- **Expected Result**: An error message is displayed indicating that the email format is invalid.

Test Case 5: Password Too Short

- **Test Case ID**: TC005
- **Objective**: Verify that the system rejects passwords that are too short.
- **Preconditions**: User is on the registration page.
- **Test Steps**:
  1. Enter a valid username.
  2. Enter a valid email address.
  3. Enter a short password (e.g., “short”).

4. Click on the “Register” button.
- **Expected Result**: An error message is displayed indicating that the password must be at least 8 characters long.

Test Case 6: Password Complexity Requirements

- **Test Case ID**: TC006
- **Objective**: Verify that the system enforces password complexity requirements.
- **Preconditions**: User is on the registration page.
- **Test Steps**:
  1. Enter a valid username.
  2. Enter a valid email address.
  3. Enter a password that does not meet complexity requirements (e.g., “password”).
  4. Click on the “Register” button.
- **Expected Result**: An error message is displayed indicating that the password must include uppercase letters, numbers, and special characters.

Test Case 7: Duplicate Username

- **Test Case ID**: TC007
- **Objective**: Verify that the system rejects a duplicate username.
- **Preconditions**: User is on the registration page, and “existingUser” already exists in the database.
- **Test Steps**:
  1. Enter an existing username (e.g., “existingUser”).
  2. Enter a valid email address.
  3. Enter a valid password.
  4. Click on the “Register” button.
- **Expected Result**: An error message is displayed indicating that the username is already taken.

Test Case 8: Duplicate Email Address

- **Test Case ID**: TC008
- **Objective**: Verify that the system rejects a duplicate email address.
- **Preconditions**: User is on the registration page, and “existinguser@example.com“ already exists in the database.
- **Test Steps**:
  1. Enter a valid username.
  2. Enter an existing email address (e.g., “existinguser@example.com“).
  3. Enter a valid password.
  4. Click on the “Register” button.
- **Expected Result**: An error message is displayed indicating that the email address is already registered.

Test Case 9: Empty Required Fields

- **Test Case ID**: TC009
- **Objective**: Verify that the system does not allow submission with empty required fields.
- **Preconditions**: User is on the registration page.
- **Test Steps**:
  1. Leave all fields empty.
  2. Click on the “Register” button.
- **Expected Result**: Error messages are displayed for each required field indicating that the fields cannot be empty (e.g., “Username is required.”, “Email is required.”, “Password is required.”).

Test Case 10: Maximum Length for Username and Password

- **Test Case ID**: TC010
- **Objective**: Verify that the system accepts the maximum length for username and password.

- **Preconditions**: User is on the registration page.
- **Test Steps**:
  1. Enter a username at the maximum length allowed (e.g., 30 characters).
  2. Enter a valid email address.
  3. Enter a password at the maximum length allowed (e.g., 20 characters).
  4. Click on the “Register” button.
- **Expected Result**: User registration is successful, and the user is redirected to a welcome page or receives a success message.

Additional Considerations:

- Ensure that all error messages are user-friendly and provide clear guidance on how to correct the input.
- Consider testing the performance of the registration process under load (e.g., multiple users registering simultaneously).
- Test cases should also include checks for the accessibility of the registration form and its compliance with relevant standards.

Feel free to adjust these test cases based on any specific requirements or additional validation rules you may have!

Your response might be different from what is shown above but at the core, the response is similar. This response is more exhaustive as the question was more specific.

Now, to see what one of the test cases would look like, ask for a code-specific example.

Check where the test cases belong

We have test cases based on multiple scenarios for validating the registration information. As a software developer, it is also important to segregate the test cases on whether it should be in the front end or the back end. All of the responses in the previous response is evidently for front end. But use the help of Generative AI to determine the same. In the prompt window, type the following:

```
Will these test cases fit better in front-end testing or back-end testing?
```

The output could be something similar to what is shown in this image:

▼ Click here to view the sample response generated

The test cases provided for user registration validation encompass both front-end and back-end testing aspects. Here’s how they fit into each category:

Front-End Testing
Front-end testing focuses on the user interface and user experience. The following test cases are primarily related to front-end testing:

1. **Valid User Registration Information (TC001)**: Tests the successful flow of registration through the UI.
2. **Invalid Username (Too Short) (TC002)**: Tests the user interface validation for username length.
3. **Invalid Username (Special Characters) (TC003)**: Tests the UI validation for acceptable characters in usernames.

4. **Invalid Email Format (TC004)**: Tests the UI validation for email input.
5. **Password Too Short (TC005)**: Tests the UI validation for password length.
6. **Password Complexity Requirements (TC006)**: Tests the UI validation for password strength.
7. **Empty Required Fields (TC009)**: Tests the UI response when required fields are left empty.
8. **Maximum Length for Username and Password (TC010)**: Tests how the UI handles maximum input lengths.

These test cases ensure that the user interface correctly validates inputs and provides appropriate feedback to users.

Back-End Testing
Back-end testing focuses on the server-side logic, database interactions, and business rules. The following test cases are primarily related to back-end testing:

1. **Duplicate Username (TC007)**: Tests the server-side logic to check if the username already exists in the database.
2. **Duplicate Email Address (TC008)**: Tests the server-side logic to check if the email already exists in the database.

These test cases ensure that the back-end correctly processes registration requests and enforces business rules related to data integrity.

Conclusion

- **Front-End Testing**: Most of the test cases are focused on validating user input at the interface level, ensuring a good user experience and proper error handling.
- **Back-End Testing**: A couple of test cases focus on the server-side logic to validate uniqueness constraints and ensure that the registration process adheres to business rules.

In practice, both front-end and back-end testing are essential for a comprehensive validation of the user registration module. It's important to ensure that both the user interface and the underlying logic work seamlessly together to provide a reliable registration process.

Segregate front end and back end test cases

Get the list of test cases with a clear segregation of what should be in the front end and what should be in the back end. In the prompt window, type the following:

```
Recommend for each of the test cases suggested whether it should belong to the front end or the back end. Include the recommendation for each test case and list it again.
```

The output could be something similar to what is shown in this image:

▼ Click here to view the sample response generated

Here's the list of the test cases along with recommendations on whether they belong to front-end or back-end testing:

Test Case List with Recommendations

1. **Test Case ID**: TC001
    - **Objective**: Verify successful registration with valid information.

    - **Recommendation**: **Front-End** (validates the UI flow and user experience).

2. **Test Case ID**: TC002
    - **Objective**: Verify that the system rejects usernames that are too short.
    - **Recommendation**: **Front-End** (validates client-side input rules).
3. **Test Case ID**: TC003
    - **Objective**: Verify that the system rejects usernames containing special characters.
    - **Recommendation**: **Front-End** (validates client-side input rules).
4. **Test Case ID**: TC004
    - **Objective**: Verify that the system rejects invalid email formats.
    - **Recommendation**: **Front-End** (validates client-side input rules).
5. **Test Case ID**: TC005
    - **Objective**: Verify that the system rejects passwords that are too short.
    - **Recommendation**: **Front-End** (validates client-side input rules).
6. **Test Case ID**: TC006
    - **Objective**: Verify that the system enforces password complexity requirements.
    - **Recommendation**: **Front-End** (validates client-side input rules).
7. **Test Case ID**: TC007
    - **Objective**: Verify that the system rejects a duplicate username.
    - **Recommendation**: **Back-End** (validates server-side logic and database checks).
8. **Test Case ID**: TC008
    - **Objective**: Verify that the system rejects a duplicate email address.
    - **Recommendation**: **Back-End** (validates server-side logic and database checks).
9. **Test Case ID**: TC009
    - **Objective**: Verify that the system does not allow submission with empty required fields.
    - **Recommendation**: **Front-End** (validates client-side input rules).
10. **Test Case ID**: TC010
    - **Objective**: Verify that the system accepts the maximum length for username and password.
    - **Recommendation**: **Front-End** (validates client-side input rules).

Summary

- **Front-End Test Cases**: TC001, TC002, TC003, TC004, TC005, TC006, TC009, TC010
- **Back-End Test Cases**: TC007, TC008

This classification helps ensure that both the user interface and server-side logic are thoroughly tested for the user registration module.

Let's prompt and find out what *processed data correctly* entails.

Specific backend test case

You can use Gen AI to find out the kind of processing to be done in the backend. Type the following prompt:

```
Elaborate on the back-end testing.
```

The output would be similar to what is given below. Notice that it includes more specific information about the kind of back-end validation to do.

▼ Click here to view the sample response generated

Back-end testing focuses on validating the server-side components of an application, including the database, APIs, and business logic. It ensures that the system behaves as expected when processing requests and that it adheres to the defined business rules. Here's a detailed elaboration on the back-end testing aspects for the user registration module, specifically for the test cases identified as back-end:

Back-End Testing Overview
Back-end testing involves several key areas:

1. **Database Validation**: Ensuring that data is stored correctly in the database and that constraints (like uniqueness) are enforced.
2. **API Testing**: Validating that the API endpoints process requests and responses correctly, returning the expected results.
3. **Business Logic Verification**: Ensuring that the application logic behaves as intended, including handling edge cases and error conditions.
4. **Security Testing**: Verifying that the system is secure against common vulnerabilities, such as SQL injection or unauthorized access.

Back-End Test Cases for User Registration

Test Case 1: Duplicate Username

- **Test Case ID**: TC007
- **Objective**: Verify that the system rejects a duplicate username.
- **Preconditions**: The username "existingUser" already exists in the database.
- **Test Steps**:
    1. Send a registration request with the existing username (e.g., "existingUser").
    2. Include a valid email address and password.
- **Expected Result**: The system returns an error response indicating that the username already exists. The database remains unchanged.

Back-End Testing Focus:

- Check the database to confirm that the existing username is indeed present.
- Ensure that the API correctly processes the request and returns the appropriate error code (e.g., 400 Bad Request).
- Verify that the error message returned is clear and user-friendly.

Test Case 2: Duplicate Email Address

- **Test Case ID**: TC008
- **Objective**: Verify that the system rejects a duplicate email address.
- **Preconditions**: The email "existinguser@example.com" already exists in the database.
- **Test Steps**:
  1. Send a registration request with the existing email address (e.g., "existinguser@example.com").
  2. Include a valid username and password.
- **Expected Result**: The system returns an error response indicating that the email address is already registered. The database remains unchanged.

Back-End Testing Focus:

- Check the database to confirm that the existing email address is indeed present.
- Ensure that the API correctly processes the request and returns the appropriate error code (e.g., 400 Bad Request).
- Verify that the error message returned is clear and user-friendly.

Additional Back-End Testing Considerations

1. **Database Constraints**:
   - Test the database constraints to ensure that they are correctly enforced. For example, check that unique constraints on usernames and email addresses prevent duplicates from being inserted.
2. **Performance Testing**:
   - Evaluate how the registration process performs under load. This can include simulating multiple users trying to register simultaneously to see how the system handles concurrent requests.
3. **Security Testing**:
   - Test for potential vulnerabilities, such as SQL injection attacks. Ensure that input validation and sanitization are implemented to prevent malicious data from being processed.
4. **Error Handling**:
   - Verify that the application gracefully handles errors, such as database connection failures, and returns meaningful error messages to the user.
5. **Integration Testing**:
   - Ensure that the registration module integrates correctly with other components of the system, such as email verification services or user management systems.
6. **Logging and Monitoring**:
   - Verify that appropriate logging is in place for registration attempts, including successful registrations and errors. This can help in troubleshooting issues and monitoring user activity.

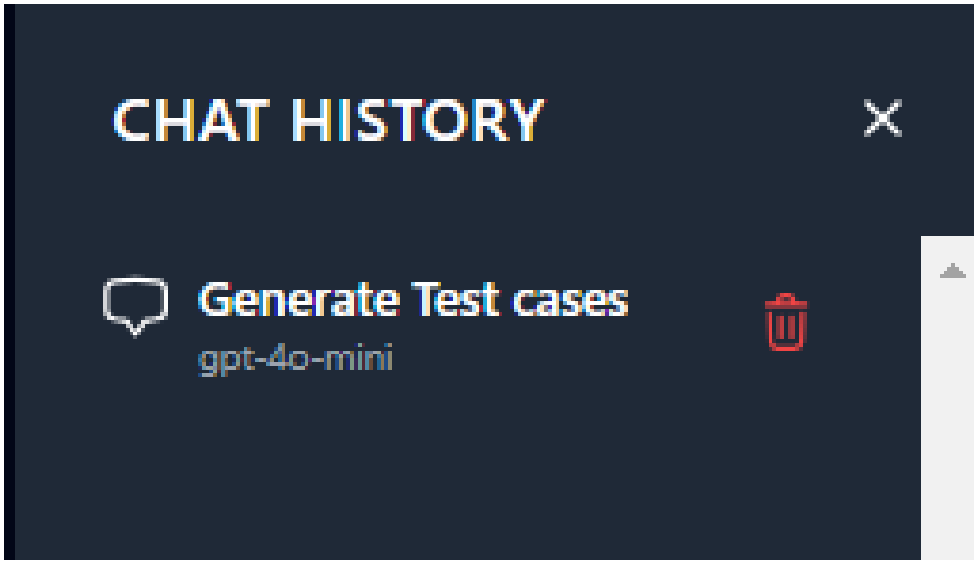
Conclusion
Back-end testing is crucial for ensuring that the user registration module operates correctly, securely, and efficiently. By focusing on database integrity, API functionality, and business logic, you can ensure a robust registration process that provides a positive user experience while maintaining data integrity and security.

The response above details the kind of check we can do.

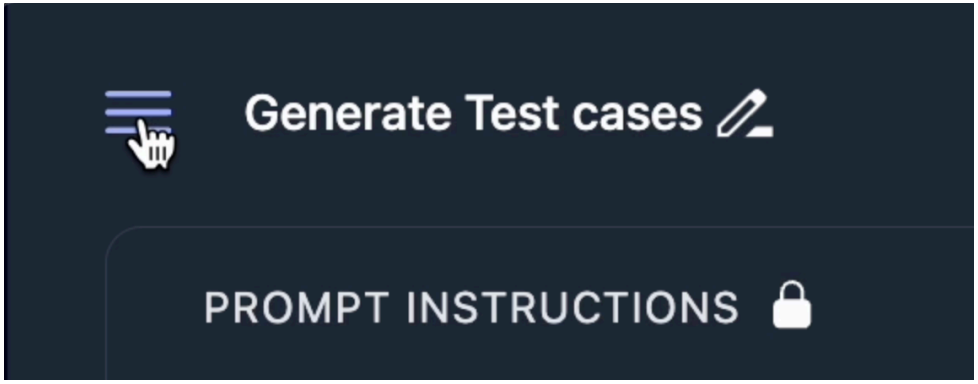
Conclusion

Congratulations! You have generated test cases that can be used in both the front end and the back end. If you are happy with the output produced, you may use it. If you want to improvise, you may ask additional relevant questions. If you want the code sample for development or testing, you can ask for specific code samples specifying the language preferred. Generative AI depends on external sources to supplement its responses with more facts and realism. However, it is your prerogative as a software developer to ascertain the accuracy of the output.

You can always visit the conversations during the live session using the history option by clicking on the main menu on the top-left.



Then, choose the chat by title among all the labs listed.



Author(s)

Rama
Lavanya