Certainly! Let's tackle this step by step.

**PlantUML Sequence Diagram**

Here's the PlantUML code for the sequence diagram you described:

plantuml

Copy code

@startuml

actor User

entity Homepage

entity AddSubComponent\_Page

entity DisplaySubComponent\_Page

entity Backend\_Server

database Database

User -> Homepage: Hits Home button

Homepage -> Homepage: Render Homepage

User -> AddSubComponent\_Page: Hits AddSubComponent button

AddSubComponent\_Page -> AddSubComponent\_Page: Render AddSubComponent\_Page

User -> AddSubComponent\_Page: Enter form details

AddSubComponent\_Page -> AddSubComponent\_Page: Validate JSON format

AddSubComponent\_Page -> User: Error message (Invalid JSON format)

AddSubComponent\_Page -> Backend\_Server: Valid JSON format

Backend\_Server -> Database: Add data to "stub" table

Database --> Backend\_Server: Acknowledgement

Backend\_Server --> AddSubComponent\_Page: Acknowledgement

User -> DisplaySubComponent\_Page: Hits DisplaySubComponent button

DisplaySubComponent\_Page -> DisplaySubComponent\_Page: Render DisplaySubComponent\_Page

User -> DisplaySubComponent\_Page: Enter form details

DisplaySubComponent\_Page -> Backend\_Server: Send form details

Backend\_Server -> Database: Check details in "stub" table

Database --> Backend\_Server: Data not found

Backend\_Server --> DisplaySubComponent\_Page: Data not found message

Database --> Backend\_Server: Data found

Backend\_Server --> DisplaySubComponent\_Page: Table data

@enduml

**Product Document for "STUB GENERATOR"**

**Product Document: Stub Generator**

**Table of Contents**

1. **Overview**
2. **High-Level Design**
3. **Low-Level Design**
4. **Sequence Diagram**
5. **User Interaction Flow**
6. **Database Schema**
7. **Error Handling**
8. **Screenshots**
9. **Future Enhancements**

**1. Overview**

The "Stub Generator" application ensures continuous operation of the bank application by generating stubs when any dependency fails. It connects to a database to generate and manage these stubs, providing seamless integration during testing and development phases.

**2. High-Level Design**

**Description:**

* The High-Level Design (HLD) provides an abstract overview of the system architecture, outlining the main components and their interactions.

**Diagram:**

* *Insert High-Level Design Diagram here*

**3. Low-Level Design**

**Description:**

* The Low-Level Design (LLD) offers detailed information about each component, including class diagrams, methods, and attributes.

**Diagram:**

* *Insert Low-Level Design Diagram here*

**4. Sequence Diagram**

**Description:**

* The Sequence Diagram illustrates the interaction between the user, pages, backend server, and database for different user actions.

**Diagram:**

* *Insert Sequence Diagram here*
* See below for the PlantUML code used to generate the diagram.

plantuml

Copy code

@startuml

actor User

entity Homepage

entity AddSubComponent\_Page

entity DisplaySubComponent\_Page

entity Backend\_Server

database Database

User -> Homepage: Hits Home button

Homepage -> Homepage: Render Homepage

User -> AddSubComponent\_Page: Hits AddSubComponent button

AddSubComponent\_Page -> AddSubComponent\_Page: Render AddSubComponent\_Page

User -> AddSubComponent\_Page: Enter form details

AddSubComponent\_Page -> AddSubComponent\_Page: Validate JSON format

AddSubComponent\_Page -> User: Error message (Invalid JSON format)

AddSubComponent\_Page -> Backend\_Server: Valid JSON format

Backend\_Server -> Database: Add data to "stub" table

Database --> Backend\_Server: Acknowledgement

Backend\_Server --> AddSubComponent\_Page: Acknowledgement

User -> DisplaySubComponent\_Page: Hits DisplaySubComponent button

DisplaySubComponent\_Page -> DisplaySubComponent\_Page: Render DisplaySubComponent\_Page

User -> DisplaySubComponent\_Page: Enter form details

DisplaySubComponent\_Page -> Backend\_Server: Send form details

Backend\_Server -> Database: Check details in "stub" table

Database --> Backend\_Server: Data not found

Backend\_Server --> DisplaySubComponent\_Page: Data not found message

Database --> Backend\_Server: Data found

Backend\_Server --> DisplaySubComponent\_Page: Table data

@enduml

**5. User Interaction Flow**

**Home Button:**

* User clicks Home button.
* Renders Homepage.

**Add SubComponent:**

* User clicks AddSubComponent button.
* Renders AddSubComponent\_Page.
* User enters form details.
* Form details are validated for JSON format.
  + If invalid, an error message is displayed.
  + If valid, details are sent to the backend server and stored in the "stub" table in the database.

**Display SubComponent:**

* User clicks DisplaySubComponent button.
* Renders DisplaySubComponent\_Page.
* User enters form details.
* Form details are sent to the backend server.
* Backend server checks the details in the "stub" table.
  + If not found, a "Data not found" message is displayed.
  + If found, the table data is sent back to DisplaySubComponent\_Page.

**6. Database Schema**

**Tables:**

* stub: Stores stub details.
* generatedstubs: Stores generated stubs.

**Schema:**

* *Insert database schema details here*

**7. Error Handling**

**Validation Errors:**

* Invalid JSON format in AddSubComponent form.

**Database Errors:**

* Data not found in the "stub" table when queried from DisplaySubComponent form.

**8. Screenshots**

**Homepage:**

* *Insert Homepage screenshot here*

**AddSubComponent Page:**

* *Insert AddSubComponent Page screenshot here*

**DisplaySubComponent Page:**

* *Insert DisplaySubComponent Page screenshot here*

**9. Future Enhancements**

* Implementing advanced validation for form inputs.
* Adding more detailed logging for debugging purposes.
* Extending support for additional dependency failures.
* Enhancing user interface for better user experience.

This product document outlines the structure, design, and functionality of the "Stub Generator" application, providing a comprehensive guide for developers and stakeholders.

4o

Give a detailed product documentation

ChatGPT

Sure, here's a detailed product documentation for the "Stub Generator" application:

**Product Documentation: Stub Generator**

**Table of Contents**

1. [Overview](#overview)
2. [System Architecture](#system-architecture)
   * [High-Level Design](#high-level-design)
   * [Low-Level Design](#low-level-design)
3. [Sequence Diagram](#sequence-diagram)
4. [User Interaction Flow](#user-interaction-flow)
5. [Database Schema](#database-schema)
6. [Error Handling](#error-handling)
7. [Screenshots](#screenshots)
8. [Future Enhancements](#future-enhancements)

**Overview**

The "Stub Generator" application ensures the uninterrupted operation of the bank application by generating stubs when any dependency fails. This tool connects to a database to generate and manage these stubs, facilitating seamless integration during testing and development.

**System Architecture**

**High-Level Design**

**Description:** The high-level design (HLD) provides an abstract overview of the system architecture, outlining the main components and their interactions.

**Diagram:** *Insert High-Level Design Diagram here*

**Low-Level Design**

**Description:** The low-level design (LLD) offers detailed information about each component, including class diagrams, methods, and attributes.

**Diagram:** *Insert Low-Level Design Diagram here*

**Sequence Diagram**

**Description:** The sequence diagram illustrates the interaction between the user, pages, backend server, and database for different user actions.

**Diagram:**

plantuml

Copy code

@startuml

actor User

entity Homepage

entity AddSubComponent\_Page

entity DisplaySubComponent\_Page

entity Backend\_Server

database Database

User -> Homepage: Hits Home button

Homepage -> Homepage: Render Homepage

User -> AddSubComponent\_Page: Hits AddSubComponent button

AddSubComponent\_Page -> AddSubComponent\_Page: Render AddSubComponent\_Page

User -> AddSubComponent\_Page: Enter form details

AddSubComponent\_Page -> AddSubComponent\_Page: Validate JSON format

AddSubComponent\_Page -> User: Error message (Invalid JSON format)

AddSubComponent\_Page -> Backend\_Server: Valid JSON format

Backend\_Server -> Database: Add data to "stub" table

Database --> Backend\_Server: Acknowledgement

Backend\_Server --> AddSubComponent\_Page: Acknowledgement

User -> DisplaySubComponent\_Page: Hits DisplaySubComponent button

DisplaySubComponent\_Page -> DisplaySubComponent\_Page: Render DisplaySubComponent\_Page

User -> DisplaySubComponent\_Page: Enter form details

DisplaySubComponent\_Page -> Backend\_Server: Send form details

Backend\_Server -> Database: Check details in "stub" table

Database --> Backend\_Server: Data not found

Backend\_Server --> DisplaySubComponent\_Page: Data not found message

Database --> Backend\_Server: Data found

Backend\_Server --> DisplaySubComponent\_Page: Table data

@enduml

**User Interaction Flow**

**Home Button**

* **Action:** User clicks Home button.
* **Result:** Renders Homepage.

**Add SubComponent**

* **Action:** User clicks AddSubComponent button.
* **Result:** Renders AddSubComponent\_Page.
* **Form Details:**
  + User enters form details.
  + Form details are validated for JSON format.
    - If invalid, an error message is displayed.
    - If valid, details are sent to the backend server and stored in the "stub" table in the database.

**Display SubComponent**

* **Action:** User clicks DisplaySubComponent button.
* **Result:** Renders DisplaySubComponent\_Page.
* **Form Details:**
  + User enters form details.
  + Form details are sent to the backend server.
  + Backend server checks the details in the "stub" table.
    - If not found, a "Data not found" message is displayed.
    - If found, the table data is sent back to DisplaySubComponent\_Page.

**Database Schema**

**Tables**

* **stub:** Stores stub details.
* **generatedstubs:** Stores generated stubs.

**Schema**

**stub Table:**

* id (Primary Key): Integer
* component\_name: String
* component\_details: JSON
* created\_at: Timestamp

**generatedstubs Table:**

* id (Primary Key): Integer
* stub\_id (Foreign Key referencing stub): Integer
* response\_data: JSON
* generated\_at: Timestamp

**Error Handling**

**Validation Errors**

* **Invalid JSON format:** When the form data entered in AddSubComponent\_Page is not in a valid JSON format, an error message is displayed to the user.

**Database Errors**

* **Data not found:** When the form details entered in DisplaySubComponent\_Page are not found in the "stub" table, a "Data not found" message is displayed.

**Screenshots**

**Homepage**

*Insert Homepage screenshot here*

**AddSubComponent Page**

*Insert AddSubComponent Page screenshot here*

**DisplaySubComponent Page**

*Insert DisplaySubComponent Page screenshot here*

**Future Enhancements**

* **Advanced Validation:** Implementing advanced validation for form inputs.
* **Detailed Logging:** Adding more detailed logging for debugging purposes.
* **Extended Support:** Extending support for additional dependency failures.
* **UI Enhancements:** Enhancing user interface for better user experience.

This comprehensive product documentation provides a detailed guide for understanding the "Stub Generator" application, its design, functionality, and future enhancement possibilities.

4o

Give Functional and Non Functional requirements i am using Flask for backend

ChatGPT

Certainly! Here are the functional and non-functional requirements for the "Stub Generator" application using Flask for the backend.

**Functional Requirements**

**1. User Authentication and Authorization**

* **FR1.1:** The system shall allow users to register with a username and password.
* **FR1.2:** The system shall authenticate users using their credentials.
* **FR1.3:** The system shall authorize users based on roles (e.g., admin, developer).

**2. Homepage**

* **FR2.1:** The system shall display a homepage when the user hits the home button.
* **FR2.2:** The homepage shall provide navigation links to AddSubComponent and DisplaySubComponent pages.

**3. Add SubComponent**

* **FR3.1:** The system shall display the AddSubComponent\_Page when the user hits the AddSubComponent button.
* **FR3.2:** The system shall validate the form details to ensure they are in valid JSON format.
* **FR3.3:** The system shall display an error message if the form details are not in valid JSON format.
* **FR3.4:** The system shall send the valid form details to the backend server.
* **FR3.5:** The backend server shall add the data to the "stub" table in the database.
* **FR3.6:** The system shall acknowledge the user upon successful addition of data.

**4. Display SubComponent**

* **FR4.1:** The system shall display the DisplaySubComponent\_Page when the user hits the DisplaySubComponent button.
* **FR4.2:** The system shall allow the user to enter form details on DisplaySubComponent\_Page.
* **FR4.3:** The system shall send the form details to the backend server.
* **FR4.4:** The backend server shall check the details in the "stub" table.
* **FR4.5:** The system shall display a "Data not found" message if the details are not found.
* **FR4.6:** The system shall display the table data if the details are found.

**5. Database Management**

* **FR5.1:** The system shall store stub details in the "stub" table.
* **FR5.2:** The system shall store generated stubs in the "generatedstubs" table.

**6. Error Handling**

* **FR6.1:** The system shall handle validation errors and display appropriate messages to the user.
* **FR6.2:** The system shall handle database errors and display appropriate messages to the user.

**Non-Functional Requirements**

**1. Performance**

* **NFR1.1:** The system shall respond to user requests within 2 seconds under normal load conditions.
* **NFR1.2:** The system shall be able to handle up to 100 concurrent users without significant performance degradation.

**2. Security**

* **NFR2.1:** The system shall use HTTPS for all communications to ensure data security.
* **NFR2.2:** The system shall encrypt user passwords in the database.
* **NFR2.3:** The system shall implement measures to prevent SQL injection and cross-site scripting (XSS) attacks.

**3. Scalability**

* **NFR3.1:** The system shall be designed to easily scale horizontally to accommodate increased load.
* **NFR3.2:** The database shall be designed to handle large volumes of data efficiently.

**4. Reliability**

* **NFR4.1:** The system shall have an uptime of 99.9%.
* **NFR4.2:** The system shall implement backup and recovery mechanisms to prevent data loss.

**5. Usability**

* **NFR5.1:** The system shall have an intuitive and user-friendly interface.
* **NFR5.2:** The system shall provide clear error messages and guidance for users.

**6. Maintainability**

* **NFR6.1:** The system shall be modular to allow for easy updates and maintenance.
* **NFR6.2:** The system shall have comprehensive documentation for developers and users.

**7. Compatibility**

* **NFR7.1:** The system shall be compatible with major web browsers (Chrome, Firefox, Edge, Safari).
* **NFR7.2:** The system shall be compatible with mobile and desktop devices.

**8. Logging and Monitoring**

* **NFR8.1:** The system shall log all user actions and system events.
* **NFR8.2:** The system shall provide monitoring capabilities to track system performance and detect issues.

These functional and non-functional requirements provide a comprehensive overview of what the "Stub Generator" application should achieve and the quality attributes it should possess.

4o

Give testcases also by keeping in mind the above description and The add subcomponent will have a form which contains fields (Subcomponent name , URL , ACTION(POST or GET etc),endpoint,reqbody(json),parameter(json),response structure(json)) and a submit button The display subcomponent will have a toggle button value between "URL","SubComponent" and a textinput of Value

ChatGPT

Certainly! Here are the test cases for the "Stub Generator" application based on the provided description and functional requirements.

**Test Cases for "Stub Generator"**

**1. User Authentication and Authorization**

**Test Case 1.1: User Registration**

* **Description:** Verify user can register with a username and password.
* **Pre-condition:** None.
* **Steps:**
  1. Navigate to the registration page.
  2. Enter a valid username and password.
  3. Click the register button.
* **Expected Result:** User is successfully registered and redirected to the login page.

**Test Case 1.2: User Login**

* **Description:** Verify user can log in with valid credentials.
* **Pre-condition:** User is registered.
* **Steps:**
  1. Navigate to the login page.
  2. Enter the registered username and password.
  3. Click the login button.
* **Expected Result:** User is successfully logged in and redirected to the homepage.

**2. Homepage**

**Test Case 2.1: Homepage Navigation**

* **Description:** Verify clicking the home button navigates to the homepage.
* **Pre-condition:** User is logged in.
* **Steps:**
  1. Click the home button.
* **Expected Result:** Homepage is displayed.

**3. Add SubComponent**

**Test Case 3.1: AddSubComponent Page Navigation**

* **Description:** Verify clicking the AddSubComponent button navigates to the AddSubComponent\_Page.
* **Pre-condition:** User is logged in.
* **Steps:**
  1. Click the AddSubComponent button.
* **Expected Result:** AddSubComponent\_Page is displayed.

**Test Case 3.2: AddSubComponent Form Validation (Invalid JSON)**

* **Description:** Verify error message is displayed when form details are not in valid JSON format.
* **Pre-condition:** User is on AddSubComponent\_Page.
* **Steps:**
  1. Enter invalid JSON in the reqbody, parameter, or response structure fields.
  2. Click the submit button.
* **Expected Result:** Error message "Invalid JSON format" is displayed.

**Test Case 3.3: AddSubComponent Form Submission (Valid JSON)**

* **Description:** Verify form submission with valid JSON format.
* **Pre-condition:** User is on AddSubComponent\_Page.
* **Steps:**
  1. Enter valid details in the form fields (Subcomponent name, URL, ACTION, endpoint, reqbody, parameter, response structure).
  2. Click the submit button.
* **Expected Result:** Data is sent to the backend server and added to the "stub" table in the database. Acknowledgment message is displayed.

**4. Display SubComponent**

**Test Case 4.1: DisplaySubComponent Page Navigation**

* **Description:** Verify clicking the DisplaySubComponent button navigates to the DisplaySubComponent\_Page.
* **Pre-condition:** User is logged in.
* **Steps:**
  1. Click the DisplaySubComponent button.
* **Expected Result:** DisplaySubComponent\_Page is displayed.

**Test Case 4.2: DisplaySubComponent Search (Toggle URL)**

* **Description:** Verify searching by URL in DisplaySubComponent\_Page.
* **Pre-condition:** User is on DisplaySubComponent\_Page.
* **Steps:**
  1. Select "URL" from the toggle button.
  2. Enter a valid URL in the text input.
  3. Click the search button.
* **Expected Result:** Backend server checks the details in the "stub" table. If not found, "Data not found" message is displayed. If found, the table data is displayed.

**Test Case 4.3: DisplaySubComponent Search (Toggle SubComponent)**

* **Description:** Verify searching by SubComponent name in DisplaySubComponent\_Page.
* **Pre-condition:** User is on DisplaySubComponent\_Page.
* **Steps:**
  1. Select "SubComponent" from the toggle button.
  2. Enter a valid SubComponent name in the text input.
  3. Click the search button.
* **Expected Result:** Backend server checks the details in the "stub" table. If not found, "Data not found" message is displayed. If found, the table data is displayed.

**5. Database Management**

**Test Case 5.1: Database Entry for AddSubComponent**

* **Description:** Verify data is correctly added to the "stub" table.
* **Pre-condition:** Form submission with valid JSON in AddSubComponent\_Page.
* **Steps:**
  1. Submit the form in AddSubComponent\_Page with valid details.
  2. Check the "stub" table in the database.
* **Expected Result:** The new entry with the correct details is added to the "stub" table.

**6. Error Handling**

**Test Case 6.1: JSON Validation Error Handling**

* **Description:** Verify error handling for invalid JSON in AddSubComponent\_Page.
* **Pre-condition:** User is on AddSubComponent\_Page.
* **Steps:**
  1. Enter invalid JSON in the reqbody, parameter, or response structure fields.
  2. Click the submit button.
* **Expected Result:** Error message "Invalid JSON format" is displayed.

**Test Case 6.2: Data Not Found Error Handling in DisplaySubComponent**

* **Description:** Verify error handling when data is not found in DisplaySubComponent\_Page.
* **Pre-condition:** User is on DisplaySubComponent\_Page.
* **Steps:**
  1. Enter a non-existent URL or SubComponent name.
  2. Click the search button.
* **Expected Result:** "Data not found" message is displayed.

**Test Data**

**AddSubComponent Form Fields:**

* Subcomponent name: "TestComponent"
* URL: "<http://testurl.com>"
* ACTION: "POST"
* Endpoint: "/test"
* ReqBody: { "key": "value" }
* Parameter: { "param1": "value1" }
* Response Structure: { "responseKey": "responseValue" }

**DisplaySubComponent Search:**

* URL: "<http://testurl.com>"
* SubComponent name: "TestComponent"

**Note:**

* Ensure to have a mechanism to reset the database or test environment before running these test cases to maintain consistency and avoid conflicts.
* Perform both positive and negative testing to ensure robustness.

These test cases should cover the primary functionalities and error scenarios for the "Stub Generator" application using Flask for the backend.

4o

Give example testcases covering all situations

ChatGPT

Sure, here are example test cases covering various scenarios for the "Stub Generator" application using Flask for the backend.

**Test Cases for "Stub Generator"**

**User Authentication and Authorization**

**Test Case 1.1: User Registration**

* **Description:** Verify user can register with a username and password.
* **Pre-condition:** None.
* **Steps:**
  1. Navigate to the registration page.
  2. Enter a valid username (e.g., "testuser") and password (e.g., "Test@123").
  3. Click the register button.
* **Expected Result:** User is successfully registered and redirected to the login page.

**Test Case 1.2: User Login**

* **Description:** Verify user can log in with valid credentials.
* **Pre-condition:** User is registered.
* **Steps:**
  1. Navigate to the login page.
  2. Enter the registered username (e.g., "testuser") and password (e.g., "Test@123").
  3. Click the login button.
* **Expected Result:** User is successfully logged in and redirected to the homepage.

**Homepage**

**Test Case 2.1: Homepage Navigation**

* **Description:** Verify clicking the home button navigates to the homepage.
* **Pre-condition:** User is logged in.
* **Steps:**
  1. Click the home button.
* **Expected Result:** Homepage is displayed.

**Add SubComponent**

**Test Case 3.1: AddSubComponent Page Navigation**

* **Description:** Verify clicking the AddSubComponent button navigates to the AddSubComponent\_Page.
* **Pre-condition:** User is logged in.
* **Steps:**
  1. Click the AddSubComponent button.
* **Expected Result:** AddSubComponent\_Page is displayed.

**Test Case 3.2: AddSubComponent Form Validation (Invalid JSON)**

* **Description:** Verify error message is displayed when form details are not in valid JSON format.
* **Pre-condition:** User is on AddSubComponent\_Page.
* **Steps:**
  1. Enter invalid JSON (e.g., "{"key": value}") in the reqbody field.
  2. Click the submit button.
* **Expected Result:** Error message "Invalid JSON format" is displayed.

**Test Case 3.3: AddSubComponent Form Submission (Valid JSON)**

* **Description:** Verify form submission with valid JSON format.
* **Pre-condition:** User is on AddSubComponent\_Page.
* **Steps:**
  1. Enter valid details in the form fields:
     + Subcomponent name: "TestComponent"
     + URL: "<http://testurl.com>"
     + ACTION: "POST"
     + Endpoint: "/test"
     + ReqBody: {"key": "value"}
     + Parameter: {"param1": "value1"}
     + Response Structure: {"responseKey": "responseValue"}
  2. Click the submit button.
* **Expected Result:** Data is sent to the backend server and added to the "stub" table in the database. Acknowledgment message is displayed.

**Display SubComponent**

**Test Case 4.1: DisplaySubComponent Page Navigation**

* **Description:** Verify clicking the DisplaySubComponent button navigates to the DisplaySubComponent\_Page.
* **Pre-condition:** User is logged in.
* **Steps:**
  1. Click the DisplaySubComponent button.
* **Expected Result:** DisplaySubComponent\_Page is displayed.

**Test Case 4.2: DisplaySubComponent Search (Toggle URL)**

* **Description:** Verify searching by URL in DisplaySubComponent\_Page.
* **Pre-condition:** User is on DisplaySubComponent\_Page.
* **Steps:**
  1. Select "URL" from the toggle button.
  2. Enter a valid URL (e.g., "<http://testurl.com>") in the text input.
  3. Click the search button.
* **Expected Result:** Backend server checks the details in the "stub" table. If not found, "Data not found" message is displayed. If found, the table data is displayed.

**Test Case 4.3: DisplaySubComponent Search (Toggle SubComponent)**

* **Description:** Verify searching by SubComponent name in DisplaySubComponent\_Page.
* **Pre-condition:** User is on DisplaySubComponent\_Page.
* **Steps:**
  1. Select "SubComponent" from the toggle button.
  2. Enter a valid SubComponent name (e.g., "TestComponent") in the text input.
  3. Click the search button.
* **Expected Result:** Backend server checks the details in the "stub" table. If not found, "Data not found" message is displayed. If found, the table data is displayed.

**Database Management**

**Test Case 5.1: Database Entry for AddSubComponent**

* **Description:** Verify data is correctly added to the "stub" table.
* **Pre-condition:** Form submission with valid JSON in AddSubComponent\_Page.
* **Steps:**
  1. Submit the form in AddSubComponent\_Page with valid details.
  2. Check the "stub" table in the database.
* **Expected Result:** The new entry with the correct details is added to the "stub" table.

**Error Handling**

**Test Case 6.1: JSON Validation Error Handling**

* **Description:** Verify error handling for invalid JSON in AddSubComponent\_Page.
* **Pre-condition:** User is on AddSubComponent\_Page.
* **Steps:**
  1. Enter invalid JSON (e.g., "{"key": value}") in the reqbody field.
  2. Click the submit button.
* **Expected Result:** Error message "Invalid JSON format" is displayed.

**Test Case 6.2: Data Not Found Error Handling in DisplaySubComponent**

* **Description:** Verify error handling when data is not found in DisplaySubComponent\_Page.
* **Pre-condition:** User is on DisplaySubComponent\_Page.
* **Steps:**
  1. Enter a non-existent URL or SubComponent name (e.g., "<http://nonexistenturl.com>" or "NonExistentComponent").
  2. Click the search button.
* **Expected Result:** "Data not found" message is displayed.

**Test Data**

**AddSubComponent Form Fields:**

* Subcomponent name: "TestComponent"
* URL: "<http://testurl.com>"
* ACTION: "POST"
* Endpoint: "/test"
* ReqBody: {"key": "value"}
* Parameter: {"param1": "value1"}
* Response Structure: {"responseKey": "responseValue"}

**DisplaySubComponent Search:**

* URL: "<http://testurl.com>"
* SubComponent name: "TestComponent"

**Additional Test Cases**

**Test Case 7.1: User Login with Invalid Credentials**

* **Description:** Verify login fails with invalid credentials.
* **Pre-condition:** User is registered.
* **Steps:**
  1. Navigate to the login page.
  2. Enter an invalid username (e.g., "wronguser") or password (e.g., "wrongpassword").
  3. Click the login button.
* **Expected Result:** Login fails, and an error message "Invalid credentials" is displayed.

**Test Case 8.1: AddSubComponent Form Submission with Missing Fields**

* **Description:** Verify form submission fails when required fields are missing.
* **Pre-condition:** User is on AddSubComponent\_Page.
* **Steps:**
  1. Enter valid details in some fields, leave others blank.
  2. Click the submit button.
* **Expected Result:** Form submission fails, and an error message indicating the missing fields is displayed.

**Test Case 9.1: DisplaySubComponent Search with Empty Input**

* **Description:** Verify search fails when the input field is empty.
* **Pre-condition:** User is on DisplaySubComponent\_Page.
* **Steps:**
  1. Leave the text input field empty.
  2. Click the search button.
* **Expected Result:** Search fails, and an error message "Input cannot be empty" is displayed.

These test cases cover various functional scenarios, including form validation, database interactions, and error handling, ensuring the robustness of the "Stub Generator" application.

4o

Give a table format take examples your choice with proper nouns

ChatGPT

Certainly! Here's a table format for test cases with examples for the "Stub Generator" application:

| **Test Case ID** | **Description** | **Pre-conditions** | **Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| TC-1.1 | User Registration Success | None | 1. Navigate to registration page.<br>2. Enter valid username and password.<br>3. Click register button. | User is successfully registered and redirected to the login page. |
| TC-1.2 | User Login Success | User is registered | 1. Navigate to login page.<br>2. Enter valid username and password.<br>3. Click login button. | User is successfully logged in and redirected to the homepage. |
| TC-2.1 | Navigate to Homepage | User is logged in | 1. Click home button. | Homepage is displayed. |
| TC-3.1 | Navigate to AddSubComponent Page | User is logged in | 1. Click AddSubComponent button. | AddSubComponent\_Page is displayed. |
| TC-3.2 | AddSubComponent Form Validation (Invalid JSON) | User is on AddSubComponent\_Page | 1. Enter invalid JSON in reqbody field.<br>2. Click submit button. | Error message "Invalid JSON format" is displayed. |
| TC-3.3 | AddSubComponent Form Submission (Valid JSON) | User is on AddSubComponent\_Page | 1. Enter valid details in form fields.<br>2. Click submit button. | Data is added to "stub" table. Acknowledgment message is displayed. |
| TC-4.1 | Navigate to DisplaySubComponent Page | User is logged in | 1. Click DisplaySubComponent button. | DisplaySubComponent\_Page is displayed. |
| TC-4.2 | DisplaySubComponent Search (Toggle URL) | User is on DisplaySubComponent\_Page | 1. Select "URL" from toggle button.<br>2. Enter valid URL in text input.<br>3. Click search button. | Table data related to URL is displayed or "Data not found" message is displayed. |
| TC-4.3 | DisplaySubComponent Search (Toggle SubComponent) | User is on DisplaySubComponent\_Page | 1. Select "SubComponent" from toggle button.<br>2. Enter valid SubComponent name in text input.<br>3. Click search button. | Table data related to SubComponent name is displayed or "Data not found" message is displayed. |
| TC-5.1 | Verify Database Entry for AddSubComponent | Form submitted in AddSubComponent\_Page | 1. Check "stub" table in database for added data. | New entry with correct details exists in "stub" table. |
| TC-6.1 | AddSubComponent Form Validation Error Handling | User is on AddSubComponent\_Page | 1. Enter invalid JSON in reqbody field.<br>2. Click submit button. | Error message "Invalid JSON format" is displayed. |
| TC-6.2 | Data Not Found Error Handling in DisplaySubComponent | User is on DisplaySubComponent\_Page | 1. Enter non-existent URL or SubComponent name.<br>2. Click search button. | "Data not found" message is displayed. |

**Additional Test Cases**

| **Test Case ID** | **Description** | **Pre-conditions** | **Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| TC-7.1 | User Login Failure (Invalid Credentials) | User is registered | 1. Navigate to login page.<br>2. Enter invalid username or password.<br>3. Click login button. | Login fails. Error message "Invalid credentials" is displayed. |
| TC-8.1 | AddSubComponent Form Submission with Missing Fields | User is on AddSubComponent\_Page | 1. Enter valid details in some fields, leave others blank.<br>2. Click submit button. | Form submission fails. Error message indicating missing fields is displayed. |
| TC-9.1 | DisplaySubComponent Search with Empty Input | User is on DisplaySubComponent\_Page | 1. Leave text input field empty.<br>2. Click search button. | Search fails. Error message "Input cannot be empty" is displayed. |

**Notes:**

* Ensure to adapt the test cases to match the specific functionality and requirements of the "Stub Generator" application.
* These test cases cover a range of scenarios including positive, negative, and boundary conditions to ensure thorough testing.
* Adjust test case IDs and descriptions according to your own naming conventions and specific requirements documentation.

This table format provides a structured way to document and organize test cases, making it easier to track coverage and ensure comprehensive testing of the application's functionalities.

3.5

There is no user login Authentications

ChatGPT

Understood. Since user login authentication is not a part of the application, we'll adjust the test cases accordingly. Here's a revised set of test cases for the "Stub Generator" application, focusing on the core functionalities without user authentication:

**Test Cases for "Stub Generator"**

**Homepage**

| **Test Case ID** | **Description** | **Pre-conditions** | **Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| TC-1.1 | Navigate to Homepage | None | 1. Navigate to homepage. | Homepage is displayed. |

**Add SubComponent**

| **Test Case ID** | **Description** | **Pre-conditions** | **Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| TC-2.1 | Navigate to AddSubComponent Page | User is on homepage | 1. Click AddSubComponent button. | AddSubComponent\_Page is displayed. |
| TC-2.2 | AddSubComponent Form Validation (Invalid JSON) | User is on AddSubComponent\_Page | 1. Enter invalid JSON in reqbody field.<br>2. Click submit button. | Error message "Invalid JSON format" is displayed. |
| TC-2.3 | AddSubComponent Form Submission (Valid JSON) | User is on AddSubComponent\_Page | 1. Enter valid details in form fields.<br>2. Click submit button. | Data is added to "stub" table. Acknowledgment message is displayed. |

**Display SubComponent**

| **Test Case ID** | **Description** | **Pre-conditions** | **Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| TC-3.1 | Navigate to DisplaySubComponent Page | User is on homepage | 1. Click DisplaySubComponent button. | DisplaySubComponent\_Page is displayed. |
| TC-3.2 | DisplaySubComponent Search (Toggle URL) | User is on DisplaySubComponent\_Page | 1. Select "URL" from toggle button.<br>2. Enter valid URL in text input.<br>3. Click search button. | Table data related to URL is displayed or "Data not found" message is displayed. |
| TC-3.3 | DisplaySubComponent Search (Toggle SubComponent) | User is on DisplaySubComponent\_Page | 1. Select "SubComponent" from toggle button.<br>2. Enter valid SubComponent name in text input.<br>3. Click search button. | Table data related to SubComponent name is displayed or "Data not found" message is displayed. |

**Database Management**

| **Test Case ID** | **Description** | **Pre-conditions** | **Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| TC-4.1 | Verify Database Entry for AddSubComponent | Form submitted in AddSubComponent\_Page | 1. Check "stub" table in database for added data. | New entry with correct details exists in "stub" table. |

**Error Handling**

| **Test Case ID** | **Description** | **Pre-conditions** | **Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| TC-5.1 | AddSubComponent Form Validation Error Handling | User is on AddSubComponent\_Page | 1. Enter invalid JSON in reqbody field.<br>2. Click submit button. | Error message "Invalid JSON format" is displayed. |
| TC-5.2 | Data Not Found Error Handling in DisplaySubComponent | User is on DisplaySubComponent\_Page | 1. Enter non-existent URL or SubComponent name.<br>2. Click search button. | "Data not found" message is displayed. |

**Additional Test Cases**

| **Test Case ID** | **Description** | **Pre-conditions** | **Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| TC-6.1 | AddSubComponent Form Submission with Missing Fields | User is on AddSubComponent\_Page | 1. Enter valid details in some fields, leave others blank.<br>2. Click submit button. | Form submission fails. Error message indicating missing fields is displayed. |
| TC-7.1 | DisplaySubComponent Search with Empty Input | User is on DisplaySubComponent\_Page | 1. Leave text input field empty.<br>2. Click search button. | Search fails. Error message "Input cannot be empty" is displayed. |

**Notes:**

* These test cases focus on functional testing of the "Stub Generator" application's core features.
* Adjust test case IDs, descriptions, and steps according to your specific requirements and functionality.
* Ensure comprehensive coverage of positive, negative, and boundary test scenarios to validate the application's robustness.

This revised table format aligns with the absence of user authentication in the application, focusing solely on testing the functionality related to managing subcomponents and database interactions.

3.5

Now give test cases for my backend which generates stubs

ChatGPT

Certainly! Here are test cases for testing the backend functionality of the "Stub Generator" application, focusing on generating stubs:

**Test Cases for Backend Stub Generation**

**Stub Generation**

| **Test Case ID** | **Description** | **Pre-conditions** | **Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| TC-1.1 | Generate Stub Successfully | Backend server is running | 1. Send a request to the backend API with required parameters (e.g., Subcomponent name, URL, Action, ReqBody, Parameter, Response Structure).<br>2. Verify response status code is 200. | Backend generates stub successfully and returns a stub data object. |
| TC-1.2 | Generate Stub with Invalid JSON | Backend server is running | 1. Send a request to the backend API with invalid JSON in ReqBody or Parameter.<br>2. Verify response status code is 400. | Backend returns an error response with message "Invalid JSON format". |
| TC-1.3 | Generate Stub with Missing Parameters | Backend server is running | 1. Send a request to the backend API without providing all required parameters (e.g., Action, ReqBody, Parameter).<br>2. Verify response status code is 400. | Backend returns an error response indicating missing parameters. |

**Error Handling**

| **Test Case ID** | **Description** | **Pre-conditions** | **Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| TC-2.1 | Handle Backend Server Error | Backend server encounters internal error | 1. Send a request to the backend API.<br>2. Backend server returns a 500 status code. | Frontend or client application handles the error gracefully and displays an appropriate error message to the user. |
| TC-2.2 | Handle Invalid Endpoint | Backend server is running | 1. Send a request to a non-existent endpoint or incorrect endpoint.<br>2. Verify response status code is 404. | Backend returns a "Not Found" error message or page. |

**Performance and Scalability**

| **Test Case ID** | **Description** | **Pre-conditions** | **Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| TC-3.1 | Performance Test - Generate Multiple Stubs | Backend server is running | 1. Send concurrent requests to generate multiple stubs simultaneously.<br>2. Measure response times and server load. | Backend handles concurrent requests efficiently without significant degradation in response times or server performance. |

**Integration with Database**

| **Test Case ID** | **Description** | **Pre-conditions** | **Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| TC-4.1 | Verify Stub Data in Database | Backend server is running | 1. Generate a stub through the backend API.<br>2. Check the "generatedstubs" table in the database for the generated stub data. | The generated stub data is correctly stored in the "generatedstubs" table in the database. |

**Additional Test Cases**

| **Test Case ID** | **Description** | **Pre-conditions** | **Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| TC-5.1 | Handle Timeout Scenario | Backend server is running | 1. Send a request to the backend API that takes longer than the timeout period.<br>2. Verify response or error handling for timeout. | Backend returns an appropriate error response or message indicating timeout. |

**Notes:**

* Adjust the test case IDs, descriptions, and steps based on the specific API endpoints, parameters, and expected behavior of your backend for stub generation.
* Consider adding more test cases to cover edge cases, error scenarios, and performance testing based on your application's requirements and potential use cases.
* Ensure that test environments are set up to mimic production conditions as closely as possible to validate real-world performance and scalability.

These test cases provide a comprehensive approach to testing the backend functionality of the "Stub Generator" application, focusing on stub generation, error handling, performance, scalability, and integration with the database. Adjust them according to your specific backend implementation and requirements.