**Test Reordering and Rationale**

The original BDD document is well-structured, but to optimize for isolated development and minimize dependencies, I'll reorder the tests with the following strategy:

1. **Authentication:** These tests are fundamental as they establish the user context for all other features.
2. **Data Input:** Focusing on data input allows us to create the data that will be used in subsequent tests (e.g., viewing, editing, export).
3. **Client Stack Management:** These tests cover the core CRUD (Create, Read, Update, Delete) operations on client stacks.
4. **Data Visualization:** These depend on having client stack data, so they come after the creation tests.
5. **Search and Filtering:** These also rely on having client data to search and filter.
6. **Data Export:** These require client stack data to be present.
7. **User Roles and Permissions:** While related to authentication, these are separated to focus on authorization logic after basic authentication is established.
8. **Change Log:** This is related to data modification, so it comes after the edit tests.
9. **Data Management:** This is an independent feature (backup).
10. **UI/UX:** These are generally independent and can be done last.

**Reordered Test Structure**

Here's the reordered test structure:

1. Authentication
2. Data Input
3. Client Stack Management
4. Data Visualization
5. Search and Filtering
6. Data Export
7. User Roles and Permissions
8. Change Log
9. Data Management
10. UI/UX

**AI Prompts for Each Test Scenario**

Now, I'll provide AI prompts for each test scenario, formatted as you requested.

**1. Authentication**

* **Feature:** User Authentication
* **Goal:** Ensure users can securely log in and their roles are correctly assigned.
  + **Scenario 1: Successful Login**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for successful user login. The test should check that given the user is on the login page, when the user enters a valid username and password and clicks the 'Login' button, then the user is redirected to the main application page and authenticated as a specific role (e.g., 'Editor'). You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 2: Invalid Login - Incorrect Password**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for invalid login with an incorrect password. The test should check that given the user is on the login page, when the user enters a valid username and an incorrect password and clicks the 'Login' button, then the user sees an error message indicating invalid credentials and remains on the login page. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 3: Invalid Login - Incorrect Username**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for invalid login with an incorrect username. The test should check that given the user is on the login page, when the user enters an invalid username and a valid password and clicks the 'Login' button, then the user sees an error message indicating invalid credentials and remains on the login page. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 4: Admin User Login**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for admin user login. The test should check that given the user is on the login page, when the user enters the username and password for an 'Admin' user and clicks the 'Login' button, then the user is redirected to the main application page and has access to admin-specific features (e.g., user management). You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 5: Viewer User Login**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for viewer user login. The test should check that given the user is on the login page, when the user enters the username and password for a 'Viewer' user and clicks the 'Login' button, then the user is redirected to the main application page and has read-only access to client data. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."

**2. Data Input**

* **Feature:** Manual Data Entry
* **Goal:** Verify that users can manually input client stack data.
  + **Scenario 1: Successful Data Entry**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for successful data entry. The test should check that given the user is logged in as an 'Editor' and is on the client stack input form, when the user enters valid data into all the fields and clicks 'Save', then the data is saved correctly. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 2: Data Entry with Freeform Text**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for data entry with freeform text. The test should check that given the user is logged in as an 'Editor' and is on the client stack input form, when the user enters freeform text into fields like 'NextGen Firewall' (e.g., 'Palo Alto') and clicks 'Save', then the data is saved correctly. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 3: Inline Editing**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for inline editing. The test should check that given the user is logged in as an 'Editor' and is viewing a client stack, when the user edits data directly within the display and saves the changes, then the data is updated correctly. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."

**3. Client Stack Management**

* **Feature:** Creating a New Client Stack
* **Goal:** Verify that users can create new client stack records with the correct data structure.
  + **Scenario 1: Successful Client Stack Creation**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for successful client stack creation. The test should check that given the user is logged in as an 'Editor' and is on the client list page, when the user clicks the 'Create New Client' button, enters all required client details (name, etc.), enters stack details using the template (tiers, etc.), and clicks the 'Save' button, then a new client record is created in the database, the user is redirected to the new client's detail page, and the new client's details are displayed correctly. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 2: Client Stack Creation with Missing Required Fields**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for client stack creation with missing required fields. The test should check that given the user is logged in as an 'Editor' and is on the client list page, when the user clicks the 'Create New Client' button, enters some client details but leaves out a required field (e.g., client name), and clicks the 'Save' button, then an error message is displayed indicating the missing field and the client record is not created. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 3: Client Stack Creation with Invalid Data**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for client stack creation with invalid data. The test should check that given the user is logged in as an 'Editor' and is on the client list page, when the user clicks the 'Create New Client' button, enters data in an incorrect format (e.g., invalid date), and clicks the 'Save' button, then an error message is displayed indicating the invalid data and the client record is not created. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
* **Feature:** Editing an Existing Client Stack
* **Goal:** Ensure users can modify client stack data and that changes are saved correctly.
  + **Scenario 1: Successful Client Stack Edit**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for successful client stack edit. The test should check that given the user is logged in as an 'Editor' and is on a client's detail page, when the user modifies client details (e.g., adds a tag) and clicks the 'Save' button, then the client record is updated in the database and the updated details are displayed on the client's detail page. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 2: Editing Client Stack with Invalid Data**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for editing client stack with invalid data. The test should check that given the user is logged in as an 'Editor' and is on a client's detail page, when the user modifies client details with invalid data and clicks the 'Save' button, then an error message is displayed and the client record is not updated with the invalid data. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 3: Editing Client Stack with No Changes**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for editing client stack with no changes. The test should check that given the user is logged in as an 'Editor' and is on a client's detail page, when the user does not modify any client details and clicks the 'Save' button, then the client record is not updated and a message is displayed indicating no changes were made (optional). You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
* **Feature:** Deleting a Client Stack
* **Goal:** Verify that users with the correct permissions can delete client stack records.
  + **Scenario 1: Successful Client Stack Deletion**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for successful client stack deletion. The test should check that given the user is logged in as an 'Admin' and is on a client's detail page, when the user clicks the 'Delete Client' button and confirms the deletion, then the client record is deleted from the database, the user is redirected to the client list page, and the deleted client is not visible in the list. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 2: Client Stack Deletion Attempt by Non-Admin**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for client stack deletion attempt by non-admin. The test should check that given the user is logged in as an 'Editor' and is on a client's detail page, when the user attempts to delete the client stack, then the delete option is not available or is disabled, or the user receives an authorization error message. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."

**4. Data Visualization**

* **Feature:** Displaying Client Stack Data
* **Goal:** Ensure client stack data is displayed in both tiered and mind map formats.
  + **Scenario 1: Displaying Tiered View**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for displaying tiered view. The test should check that given the user is logged in and is on a client's detail page, when the user selects the 'Tiered View', then the client's stack data is displayed in a structured, tier-based format. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 2: Displaying Mind Map View**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for displaying mind map view. The test should check that given the user is logged in and is on a client's detail page, when the user selects the 'Mind Map View', then the client's stack data is displayed in an unstructured, mind map-style graphical display. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 3: Displaying Client Stack with Nested Tiers**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for displaying client stack with nested tiers. The test should check that given the user is logged in and is on a client's detail page, when the client's stack data includes nested tiers, then the tiered view displays the nested tiers correctly and the mind map view represents the nested tiers in its graphical display. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."

**5. Search and Filtering**

* **Feature:** Searching for Clients
* **Goal:** Verify that users can search for clients based on various criteria.
  + **Scenario 1: Successful Client Search**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for successful client search. The test should check that given the user is logged in and is on the client list page, when the user enters a client's name in the search bar and clicks the 'Search' button, then the client list is filtered to display only matching clients. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 2: Client Search with No Results**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for client search with no results. The test should check that given the user is logged in and is on the client list page, when the user enters a non-existent client name in the search bar and clicks the 'Search' button, then the client list displays a message indicating no matching clients were found. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 3: Client Search by Tag**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for client search by tag. The test should check that given the user is logged in and is on the client list page, when the user searches using a client tag, then the client list is filtered to show clients with that tag. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
* **Feature:** Filtering Clients
* **Goal:** Verify that users can filter clients based on criteria
  + **Scenario 1: Filter Clients by Flags**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for filtering clients by flags. The test should check that given the user is logged in and is on the client list page, when the user filters clients by a specific flag (e.g., 'Renewal Due'), then the client list only displays clients with that flag. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."

**6. Data Export**

* **Feature:** Exporting Client Stack Data
* **Goal:** Ensure client stack data can be exported in different formats.
  + **Scenario 1: Export to PDF - Mind Map**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for exporting to PDF in Mind Map format. The test should check that given the user is logged in and is on a client's detail page, when the user selects 'Export to PDF' and chooses the 'Mind Map' option, then a PDF file is generated and the PDF file contains the client's stack data in a mind map format. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 2: Export to PDF - Tiered List**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for exporting to PDF in Tiered List format. The test should check that given the user is logged in and is on a client's detail page, when the user selects 'Export to PDF' and chooses the 'Tiered List' option, then a PDF file is generated and the PDF file contains the client's stack data in a tiered list format. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 3: Export to CSV**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for exporting to CSV. The test should check that given the user is logged in and is on a client's detail page, when the user selects 'Export to CSV', then a CSV file is generated and the CSV file contains the client's stack data in a flat data table format. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 4: Export with Hidden Fields**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for exporting with hidden fields. The test should check that given the user has hidden certain fields in the UI, when the user exports data, then the exported file does not include the hidden fields. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 5: Export with Tags and Flags Toggled Off**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for exporting with tags and flags toggled off. The test should check that given the user is on the export settings, when the user toggles off the 'tags' and 'flags' options and exports the data, then the exported file does not include tags and flags. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."

**7. User Roles and Permissions**

* **Feature:** Access Control
* **Goal:** Verify that user roles and permissions are enforced correctly.
  + **Scenario 1: Admin User Access**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for admin user access. The test should check that given the user is logged in as an 'Admin', then the user has access to all features, including user management, client data editing, and export functionality. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 2: Editor User Access**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for editor user access. The test should check that given the user is logged in as an 'Editor', then the user is able to create and update client data records and does not have access to admin-specific features. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
  + **Scenario 3: Viewer User Access**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for viewer user access. The test should check that given the user is logged in as a 'Viewer', then the user has read-only access to client data and is not able to create, update, or delete client records. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."

**8. Change Log**

* **Feature:** Change Log Tracking
* **Goal:** Verify that changes to client stacks are logged.
  + **Scenario 1: Change Log Entry on Edit**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for change log entry on edit. The test should check that given the user is logged in as an 'Editor' and edits a client stack, when the user saves the changes, then a new entry is added to the change log and the change log entry includes user identity, timestamp, and action description. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."

**9. Data Management**

* **Feature:** Data Backup
* **Goal:** Verify that the database can be backed up.
  + **Scenario 1: Database Backup**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for database backup. The test should check that given the user is logged in as an 'Admin', when the user initiates a database backup, then the database is exported as a zipped JSON or CSV file. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."

**10. UI/UX**

* **Feature:** Responsive Design
* **Goal:** Verify the UI is responsive.
  + **Scenario 1: Mobile Responsiveness**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for mobile responsiveness. The test should check that given the user is on a mobile device (e.g., iPhone 15 Pro Max), when the user accesses the application, then the UI adapts to the device's screen size and orientation and the UI is touch-first. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."
* **Feature:** Dark Mode
* **Goal:** Verify dark mode functionality.
  + **Scenario 1: Persistent Dark Mode**
    - **Prompt:** "As a software engineer specializing in React, it is your goal to write a test for persistent dark mode. The test should check that given the user toggles dark mode on, when the user navigates through the application, then dark mode persists across all pages and sessions. You will write the test first, then execute <Test Command> and continue to fix errors until the test passes. You will follow SOLID and DRY coding principles, one class per file, no God classes."

**Important Considerations**

* <Test Command>: You'll need to replace this placeholder with the actual command used to run your tests (e.g., npm test, yarn test, pytest).
* Framework Adaptation: These prompts are geared toward React for the frontend. You'll need to adjust them if you're using a different framework. For backend tests (e.g., with Flask or FastAPI), the prompts should specify testing those components.
* SOLID and DRY: Emphasize these principles to the AI agent to ensure clean, maintainable code.
* Dependencies: While the tests are reordered, some tests might still have logical dependencies (e.g., you need a client to exist before you can edit it). You might need to create "setup" functions or mock data where appropriate.
* Mocking: For true isolation, encourage the use of mocking for external dependencies (e.g., API calls, database interactions).