**Testing Manual for Java Spring Application**

**1. Overview**

This manual provides details regarding the test that were performed in the application. The tests cover functionalities related to administrative settings, email services, faculty interactions, landing page operations, and user details handling.

**Unit Testing**

**2. Testing Environment**

* **SpringBootTest**: Used for integration tests requiring the full application context.
* **MockMvc**: Utilized for web layer testing without starting an actual HTTP server.
* **Mockito**: Used for mocking dependencies and verifying interactions.

**3. Tests**

**1. AdminSettingsControllerTest**

**Purpose**: Tests the administrative settings management capabilities.

**Tests**:

* **showSettings()**: Ensures an admin can view settings via a GET request. Validates the return of the correct view and model attributes.
* **updateSettings()**: Checks the admin's ability to update settings through a POST request with form data.

**2. EmailControllerTest**

**Purpose**: Validates the email sending functionalities.

**Tests**:

* **testSendEmailToList()**: Confirms emails are sent to a list of recipients.
* **testSendTwoFactorAuthEmail()**: Verifies sending a two-factor authentication email.
* **testVerificationEmail()**: Checks the construction of a verification link and email dispatch.
* **testUsernameRecovery()**: Ensures a username recovery email is sent correctly.

**3. FacultyControllerTest**

**Purpose**: Tests faculty interactions with advisees.

**Tests**:

* **testSendEmailToAdvisees()**: Ensures emails are sent to advisees based on the authenticated user's advisor ID.

**4. LandingPageControllerTest**

**Purpose**: Handles operations for the main user interface.

**Tests**:

* **showHomePageTest()**: Tests rendering of the home page.
* **showAdminPageTest()**: Verifies the admin page is rendered correctly.
* **changeStudentPassword\_ShouldChangePassword\_WhenValid()**: Confirms password changes.
* **changeEmailTest()**, **changePhoneNumberTest()**: Check updates to email and phone number.

**5. AdviseeServiceTest**

**Purpose**: Manages advisee data storage and retrieval.

**Tests**:

* **testGetAdviseeArray()**: Verifies the storage and retrieval of advisee data.

**6. UserDetailsServiceImplTest**

**Purpose**: Implements user details service for authentication and authorization.

**Tests**:

* **loadUserByUsername\_whenUserExists()**: Ensures user details are loaded correctly.
* **loadUserByUsername\_whenUserDoesNotExist()**: Tests behavior when the user does not exist.

**7. UserRoleTest**

**Purpose:** Tests the functionality of the **UserRole** class to ensure that getters and setters, as well as other related methods, operate correctly.

**Tests:**

* **testGetterAndSetterMethods()**: Verifies that the **UserRole** object correctly stores and retrieves its ID and authority name through setter and getter methods. Ensures that the set values for ID and authority name are accurately returned by the getters.
* **testGetAuthority()**: Checks the **getAuthority** method to confirm it accurately returns the set authority name, ensuring that the method functions as expected for authority management.

**8. IndexControllerTest**

**Purpose:** Tests the IndexController, which handles web requests related to index and login functionalities including the display of pages and two-factor authentication processes.

**Tests:**

* **showIndex()**: Verifies that accessing the index page returns the correct view and HTTP status OK, ensuring that the main landing page is rendered properly.
* **showLoginPage()**: Checks that the login page is accessible and displays correctly by confirming the returned view and HTTP status OK, ensuring the login page functions as intended.
* **verify2FA()**: Tests the verification process for two-factor authentication codes. It simulates a user session, submits a 2FA code, and checks for correct handling of the authentication process including redirection after successful verification. This test confirms that the **TwoFactorAuthentication** service's **verify2FACode** method is invoked correctly and behaves as expected under simulated conditions.

**9. AdminControllerTest**

**Purpose:** Tests the AdminController functionalities within a Spring MVC framework, focusing on admin user interactions such as viewing user lists, displaying system settings, and uploading advisor data.

**Tests Completed:**

* **testViewUsers()**: Confirms that admins can successfully view all users. This test checks for the correct HTTP response status, ensures the appropriate view is rendered, and verifies that the user list is correctly passed to the model.
* **testShowSystemSettings()**: Verifies the functionality to display the system settings page to the admin. It checks that the HTTP response is correct and that the view "systemSettings" is returned as expected.
* **testUploadAdvisorsEmptyFile()**: Evaluates the system's response to an attempted upload of an empty Excel file. This test ensures that the system redirects properly and provides an error message, testing the system's error handling capabilities.

**Tests Incomplete/failed:**

* **testAddAdvisor():** Failed due to a NullPointerException when accessing UserRoleRepository. This indicates an issue with the setup or injection of the repository in the testing environment, preventing validation of the advisor addition functionality.
* **testEditAdvisor():** Similar to the add advisor test, this failed due to a NullPointerException. The problem likely stems from improper mock setup or context configuration issues.
* **testDeleteAdvisor**(): This test also encountered a NullPointerException, suggesting a systemic issue with repository injection or a flaw in how the test environment manages repository interactions.

**Volume Testing**

1. **Objective:** Test the system’s ability to manage and display a high volume of data entries from an Excel file.

**Scenario:** We uploaded an Excel file with 1000 users into advisee upload.

**Results:**

* The database has 1000 entries, confirming successful upload.
* Only 600 entries are visible in the admin's user view, indicating a possible issue in the display layer.
* Even less were visible in the Faculty advisee table after upload, 100 or less were visible.

**Conclusion:**

* System needs more scalability in future work.

**Black Box Testing**

**Objective:** Evaluate the system's ability to validate Excel file uploads by checking for correct tagging, user uniqueness, and file format.

**Test Steps and Expected Results:**

1. **File Format Validation:**
   * **Test:** Upload a correct Excel file (.xlsx).
   * **Expected Result:** File is accepted without errors.
   * **Test:** Upload a non-Excel file (.txt, .csv).
   * **Result:** System rejects the file and displays an error message about the incorrect file type.
2. **Tag Validation:**
   * **Test:** Upload an Excel file with all required tags correctly placed.
   * **Expected Result:** File is processed successfully.
   * **Test:** Upload an Excel file with missing or incorrect tags.
   * **Result:** System rejects the file and provides an error message detailing the missing or incorrect tags.
3. **Duplicate User Validation:**
   * **Test:** Upload a file with unique user entries.
   * **Result:** File is added successfully.
   * **Test:** Upload a file with duplicate user entries.
   * **Result:** System identifies duplicates and returns an error indicating which entries are duplicated.