Test Plan for Concierge

Change Log

Version	Change Date	By	Description
1.0.0	October 6th, 2024	All Members	Initial Testing Plan

Introduction

Scope

This is our testing scope for Sprint 1:

1. Amenities
    a. View all Amenities
    b. View Specified Amenity
    c. Create Amenity
    d. Update Amenity
    e. Delete Amenity
2. Incident Reports
    a. View all Incident Reports
        i. With filters
        ii. With no filter
    b. View Specified Incident Reports
    c. Create Incident Reports
    d. Update Incident Reports
    e. Delete Incident Reports
3. Account Management
    a. Server
        i. Verify get /index
        ii. Verify account login
        iii. Verify account creation
    b. Auth Manager
        i. Validate hashed passwords
        ii. Authenticate user login
        iii. Validate genuine hashes
    c. Database controller
        i. Test user retrieval
        ii. Test user addition
        iii. Test user deletion
    d. User dto
        i. Verify creation
        ii. Verify equality test
    e. User Service
        i. User creation
        ii. User deletion
    f. Validation Manager
        i. Validate password
        ii. Validate username

Roles and Responsibilities

Name	Net ID	Github Username	Role
Nhat Anh	nguyen74	nateng98	Full-stack Developer, Staff Front-end Supervisor
		rainclouded	Full-stack Developer, Database Supervisor
Mykola		mykolabesarab	Full-stack Developer, Guest Front-end Supervisor
Josh			Full-stack Developer, Back-end Supervisor, DevOps Supervisor
Leeroy	dilimI1	LeeroyDilim	Full-stack Developer, Back-end Supervisor

Role Details

1. Full-stack Developer
    - A developer focused on implementing features across the presentation, logic, and data layer
2. Guest Front-end Supervisor
    - An analyst responsible for code reviewing pull requests of changes relating to the guest front end, ensuring additions meet requirements and follow good practices.
3. Staff Front-end Supervisor
    - An analyst responsible for code reviewing pull requests of changes relating to the staff front end, ensuring additions meet requirements and follow good practices.
4. Database Supervisor
    - An analyst responsible for code reviewing pull requests of changes relating to the SQL/Mongo Databases, ensuring additions meet requirements and follow good practices.
5. Back-end Supervisor
    - An analyst responsible for code reviewing pull requests of changes relating to the microservice architecture, ensuring additions meet requirements and follow good practices.
6. DevOps Supervisor
    - An analyst responsible for supervising the CI/CD pipeline.

Test Methodology

Test Levels

Core Feature: Amenities

Unit Tests

1. The test ensures that the GetAmenities endpoint returns a response with a 200 OK status code and valid data when called.
2. When provided with a valid amenity ID, the GetAmenityByID endpoint returns a response with a 200 OK status code.
3. The test confirms that when an invalid amenity ID is used, the GetAmenityByID endpoint returns a 404 Not Found response.
4. When valid data is provided, the AddAmenity endpoint returns a 201 Created response, indicating that the amenity has been successfully added.
5. After successfully adding an amenity, the test verifies that the amenity can be fetched, and the response contains a 200 OK status code.
6. The test ensures that if invalid data is submitted to the AddAmenity endpoint, the response returns a 400 Bad Request status code.
7. After attempting to add an invalid amenity, the test checks that the amenity cannot be fetched, resulting in a 404 Not Found response.
8. The test confirms that trying to add a duplicate amenity returns a 400 Bad Request response, indicating a failure due to duplication.
9. If a null amenity is submitted, the AddAmenity endpoint returns a 400 Bad Request response, indicating invalid input.
10. When valid data is submitted to update an amenity, the test ensures that the response contains a 200 OK status code, indicating successful modification.
11. After updating an amenity, the test confirms that the updated data can be retrieved with a 200 OK response.
12. The test verifies that attempting to update an amenity with invalid data returns a 404 Not Found response.
13. After attempting an invalid update, the test ensures that the amenity cannot be fetched, resulting in a 404 Not Found response.
14. The test confirms that attempting to update a non-existing amenity returns a 404 Not Found response.
15. When a null value is submitted for updating an amenity, the endpoint returns a 404 Not Found response, indicating a failure.
16. The test ensures that when a valid amenity is deleted, the response contains a 200 OK status code.
17. After successfully deleting an amenity, the test checks that attempting to fetch it returns a 404 Not Found response.

18. When attempting to delete an amenity with an invalid ID, the response is a 404 Not Found.
19. After attempting to delete an invalid amenity, the test confirms that trying to fetch it results in a 404 Not Found response.

Integration Tests

1. **Get and View All Amenities**
   - This test verifies that the application can successfully retrieve and display a list of all amenities. The test checks the API response to ensure that the correct data is being returned in an expected format, and the list is rendered properly on the frontend.
2. **Create New Amenity**
   - This test ensures that a new amenity can be created by sending a POST request to the server with the appropriate data. It validates the API's response to confirm the amenity was created successfully and checks if the newly added amenity appears in the list of amenities on the frontend.
3. **Delete Amenity**
   - This test confirms the ability to delete an existing amenity. It checks the API response for the deletion request and validates that the amenity is removed from the server's data as well as the frontend display.
4. **Edit Amenity**
   - This test ensures that an existing amenity can be edited. It sends a PUT request to update the amenity's information, validates the API response to confirm the changes were applied, and verifies that the updated amenity is reflected on the frontend.

Acceptance Tests

1. **Scenario:** Guest views hotel amenity information.
   - Upon opening the amenities dashboard, the guest should see a comprehensive list of all amenities offered by the hotel.
   - Each amenity listed must display the following information:
   - Title of the amenity
   - Description of the amenity
   - Operating hours for the amenity
   - The list must be presented in a clear and user-friendly format for easy navigation.
2. **Scenario:** Staff updates an amenity.
   - The staff member must be logged into the staff dashboard.
   - The staff member can access the amenity details for editing.

- When the staff member clicks the "Save" button after updating the amenity details:
- The system processes the update in real-time.
- The updated amenity details are successfully saved in the system.
- A confirmation message is displayed to the staff member indicating the update was successful.

Core Feature: Incident Reports

Unit Tests

1. When filtering incident reports by a specified severity, the system should respond appropriately.
2. When filtering incident reports by a specific status, the system should respond appropriately.
3. When filtering incident reports by a range of dates, the system should respond appropriately.
4. When no filters are applied to the incident reports request, the system should respond appropriately.
5. When an invalid severity is used for filtering incident reports, the system should respond with an error.
6. When an invalid status is used for filtering incident reports, the system should respond with an error.
7. When invalid dates are provided for filtering incident reports, the system should respond with an error.
8. When retrieving an incident report by a valid identifier, the system should respond with the correct data.
9. When attempting to retrieve an incident report by an invalid identifier, the system should respond with an error.
10. When a valid incident report is submitted, the system should confirm successful creation.
11. After submitting a valid incident report, it should be possible to retrieve it successfully.
12. When submitting an incident report with a missing title, the system should respond with an error.
13. When submitting a null incident report, the system should respond with an error.
14. When submitting an incident report with a missing description, the system should respond with an error.
15. When submitting an incident report with an invalid filing person ID, the system should respond with an error.

16. When submitting an incident report with an invalid reviewer ID, the system should respond with an error.
17. When updating a valid incident report, the system should confirm the update.
18. After updating a valid incident report, it should be retrievable successfully.
19. When updating an incident report with a missing title, the system should respond with an error.
20. When attempting to update a non-existing incident report, the system should respond with an error.
21. When attempting to update an incident report with a null value, the system should respond with an error.
22. When deleting a valid incident report, the system should confirm the deletion.
23. After deleting a valid incident report, it should not be retrievable.
24. When attempting to delete an invalid incident report, the system should respond with an error.
25. After attempting to delete an invalid incident report, it should still be untraceable.

Integration Tests

1. **Get and View All Reports**
   - This test ensures that the system can successfully retrieve and display a list of all incident reports. It checks that a GET request to the server fetches the correct data and verifies that the reports are properly rendered in the user interface.
2. **Edit Incident Report**
   - This test verifies the ability to edit an existing incident report. It sends a PUT request with updated data, checks the server's response to confirm that the changes were applied, and confirms that the edited report is correctly reflected on the frontend.
3. **Delete Incident Report**
   - This test ensures that an incident report can be successfully deleted. It sends a DELETE request to the server, verifies the correct response from the backend, and checks that the deleted report is no longer visible in the list on the frontend.

Acceptance Tests

1. **Scenario:** Hotel Manager Views Past and Current Incident Reports
   - Given the hotel manager is logged in,
   - When the hotel manager accesses the incident reports dashboard,
   - Then the system displays a list of past and current incident reports.

    - And each report includes details such as severity, status, and description.
2. **Scenario:** Hotel Manager Updates an Incident Report
    - Given the hotel manager is viewing an incident report,
    - When the hotel manager updates the incident details and clicks "Save,"
    - Then the system updates the report in real-time.
    - And the system displays a confirmation message indicating the update was successful.

Core Feature: Accounts

Unit Tests

Integration Tests

Acceptance Tests

1. **Scenario:** User Logs In
    - Given the user has valid credentials,
    - When the user enters their username and password and clicks the "Login" button,
    - Then the system directs the user to their role-specific dashboard (guest, staff, or manager).
2. **Scenario:** Creating a New Account
    - Given the staff member has permission to create accounts,
    - When the staff member fills in the necessary fields for user creation and submits the form,
    - Then a new account is created successfully.
3. **Scenario:** Editing Permissions
    - Given the staff member is logged in and has permission to edit permissions,
    - When the staff member creates a new group, configures its permissions, and assigns accounts to it,
    - Then those accounts gain access to the specified tools.
4. **Scenario:** Modifying Preferences
    - Given the staff member is logged in,
    - When the staff member updates settings (e.g., default dashboard views, notification preferences, initial landing pages),
    - Then the system saves the changes and reflects them in the staff member's next login.