* Part A - Design a Test Plan and test cases
  + Include an introduction and entry criteria

Testing a web-based application that stores employee records and departments is crucial to ensure its functionality, usability, and data integrity. The application serves as a central repository for managing employee data, including their identification details, such as ID, name, phone number, and the department to which they belong. This document outlines the entry criteria for testing the web-based application, ensuring that the testing process is comprehensive and effective.

**Entry Criteria for Testing the Web-Based Application**:

1. Application Environment:

• The web-based application should be deployed and accessible in the test environment.

• The necessary infrastructure, including servers, databases, and networking components, should be set up and configured correctly.

• All required software dependencies and libraries should be installed and functional.

2. Test Data Setup:

• A representative set of employee records should be available in the application.

• Employee records should contain valid and diverse data, including different departments, names, phone numbers, and IDs.

• The department records should be populated with the correct department names, such as Information Technology, Human Resources, Finance, Marketing, and Legal.

3. User Access and Authentication:

• User accounts with appropriate access levels should be created for testing purposes.

• User roles and permissions should be properly configured, ensuring that different user types can perform their designated tasks.

• Authentication mechanisms, such as login credentials or Single Sign-On (SSO), should be operational and validated.

4. Application Functionality:

• All features related to employee record management and department association should be implemented and accessible.

• Basic operations, including creating, updating, retrieving, and deleting employee records, should be functional.

• The application should accurately link employees to their respective departments.

• Any additional functionality specific to the application, such as search, filtering, or reporting, should be available and operational.

5. User Interface:

• The user interface should be responsive, intuitive, and visually appealing.

• All forms, fields, buttons, and navigation elements should be correctly rendered and interactive.

• The user interface should adhere to design and usability standards, providing a seamless user experience.

6. Data Integrity and Security:

• The application should ensure the confidentiality, integrity, and availability of employee records and department information.

• Data encryption, access controls, and proper handling of sensitive information should be implemented and tested.

• Backup and recovery mechanisms should be in place to protect against data loss or corruption.

7. Performance and Scalability:

• The application should perform optimally under expected user load and concurrent usage.

• Load testing should be conducted to verify the application's stability and responsiveness.

• The application should be scalable, allowing for future growth and an increasing number of users and records.

By meeting the entry criteria outlined above, the testing team can ensure a solid foundation for conducting comprehensive and effective tests on the web-based application storing employee records and departments

List WebUI related test cases

1. Functional Testing:

a. Verify that the user can successfully log in to the application using valid credentials.

b. Create a new employee record and ensure that all mandatory fields (ID, name, phone, department) are correctly saved.

c. Update an existing employee record and verify that the changes are reflected accurately.

d. Delete an employee record and confirm that it is no longer present in the database.

e. Retrieve employee records by searching for specific criteria (e.g., department, name) and validate the search results. f. Verify that the application correctly associates employees with their respective departments.

2. Usability Testing:

a. Ensure that the user interface is intuitive and easy to navigate.

b. Validate that all form fields are labeled correctly and provide appropriate hints or tooltips.

c. Verify that the user can easily access and navigate between different sections of the application.

d. Test the responsiveness of the user interface across different devices and screen resolutions.

e. Check for consistency in design elements, such as fonts, colors, and layouts.

3. Compatibility Testing:

a. Test the application on different web browsers (e.g., Chrome, Firefox, Safari, Edge) to ensure consistent functionality and display.

b. Verify that the application is compatible with different operating systems (e.g., Windows, macOS, Linux).

c. Validate the application's responsiveness on various devices, including desktops, laptops, tablets, and mobile phones.

4. Security Testing:

a. Ensure that the application enforces proper authentication and authorization mechanisms.

b. Test for vulnerabilities such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

c. Validate that sensitive employee data is properly encrypted during transmission and storage.

d. Verify that only authorized users can access and modify employee records and department information.

5. Performance Testing:

a. Conduct load testing to determine the application's performance under heavy user loads.

b. Measure the response time for common actions, such as creating, updating, and retrieving employee records.

c. Test the application's scalability by gradually increasing the number of records and users.

d. Monitor resource utilization (CPU, memory, disk) during peak usage to identify any bottlenecks.

6. Accessibility Testing:

a. Ensure that the application complies with accessibility standards, such as WCAG (Web Content Accessibility Guidelines).

b. Test the application using screen readers to verify compatibility and proper reading of content.

c. Validate that all form fields have proper labels and can be navigated using keyboard-only controls.

These are just a few examples of WebUI test cases, and the actual number and scope of test cases may vary depending on the specific requirements and complexity of the web-based application.

* + List possible functional web app test cases and tag each test case with ‘To automate’ or ‘Manual’

1. User Authentication(Not present in application : future implementation): a. Verify that a user with valid credentials can log in successfully. (To automate) b. Verify that a user with invalid credentials cannot log in and receives an appropriate error message. (To automate) c. Verify that the application enforces password complexity requirements during user registration. (To automate)

2. Employee Record Management: a. Create a new employee record with all mandatory fields filled. (To automate) b. Create a new employee record with missing mandatory fields and verify that an error message is displayed. (To automate) c. Update an existing employee record and ensure that the changes are accurately reflected. (To automate) d. Delete an employee record and confirm that it is removed from the system. (To automate) e. Retrieve an employee record using various search criteria (e.g., ID, name, department) and validate the search results. (To automate) f. Verify that the application allows assigning an employee to a specific department. (To automate)

3. Department Management: a. Create a new department with a unique name. (To automate) b. Create a new department with a name that already exists and verify that an error message is displayed. (To automate) c. Update an existing department's name and ensure that the change is reflected in the application. (To automate) d. Delete a department and confirm that it is no longer available for selection. (To automate) e. Verify that the application displays the list of departments correctly. (To automate)

4. Data Validation: a. Validate that the employee ID field accepts only numeric values. (To automate) b. Verify that the phone number field accepts valid phone number formats. (To automate) c. Test the application's handling of special characters and input lengths in different fields. (To automate) d. Verify that appropriate error messages are displayed when invalid data is entered. (To automate)

5. Record Navigation and Pagination: a. Verify that the application displays a specified number of employee records per page. (To automate) b. Test the navigation buttons (e.g., Next, Previous, First, Last) and confirm that they work as expected. (To automate)

6. Integration Testing: a. Test the integration between employee records and department information to ensure accurate association. (To automate)

7. Error Handling: a. Verify that the application gracefully handles unexpected errors and displays informative error messages. (To automate)

8. Cross-Browser Testing: a. Test the application on different web browsers (e.g., Chrome, Firefox, Safari, Edge) and verify consistent functionality. (Manual)

9. Usability Testing: a. Validate the user interface's ease of use, intuitiveness, and responsiveness. (Manual)

10. Accessibility Testing: a. Test the application's compliance with accessibility standards, such as keyboard navigation and screen reader compatibility. (Manual)

Note: The categorization of "To automate" or "Manual" is based on general assumptions. The actual feasibility of automation may vary depending on the testing framework, tools, and specific implementation details.

* + List Rest API test cases

1. Employee Endpoint:

a. Verify that a GET request to retrieve all employee records returns a list of employees. (Automated) b. Verify that a GET request to retrieve a specific employee by ID returns the correct employee record. (Automated)

c. Verify that a POST request with valid data creates a new employee record successfully. (Automated)

d. Verify that a POST request with missing mandatory fields returns an appropriate error response. (Automated)

e. Verify that a PUT request with valid data updates an existing employee record successfully. (Automated)

f. Verify that a DELETE request to delete a specific employee removes the employee record from the system. (Automated)

g. Verify that a GET request with search parameters (e.g., department) returns the relevant employee records. (Automated)

2. Department Endpoint:

a. Verify that a GET request to retrieve all departments returns a list of departments. (Automated)

b. Verify that a GET request to retrieve a specific department by ID returns the correct department record. (Automated)

c. Verify that a POST request with valid data creates a new department successfully. (Automated)

d. Verify that a POST request with missing mandatory fields returns an appropriate error response. (Automated)

e. Verify that a PUT request with valid data updates an existing department successfully. (Automated) f. Verify that a DELETE request to delete a specific department removes the department record from the system. (Automated)

3. Data Validation:

a. Verify that the API returns appropriate error responses when invalid or incomplete data is provided. (Automated)

b. Validate that the API enforces data type constraints for fields such as employee ID, phone number, etc. (Automated)

c. Verify that the API handles and returns errors for duplicate entries or conflicts. (Automated)

4. Authentication and Authorization: (Future functionality not included in the application)

a. Verify that the API endpoints require authentication or authorization where necessary. (Automated)

b. Test the authentication process by providing valid credentials and ensuring successful access to protected endpoints. (Automated)

c. Verify that unauthorized access to protected endpoints results in appropriate error responses. (Automated)

5. Error Handling:

a. Test error scenarios by sending requests with invalid endpoints or incorrect request methods and validate the response codes and error messages. (Automated)

6. Pagination and Filtering:

a. Verify that the API supports pagination for retrieving a specified number of records per page. (Automated)

b. Test filtering capabilities by using query parameters (e.g., department) and validating the returned results. (Automated)

7. Performance and Load Testing:

a. Test the API's performance under various load conditions to ensure it can handle concurrent requests efficiently. (Automated)

8. Security Testing:

a. Validate that the API endpoints are protected against common security vulnerabilities, such as SQL injection and cross-site scripting (XSS). (Automated)

9. API Versioning:

a. Verify that the API supports versioning, allowing backward compatibility and smooth transitions between versions. (Automated)

Note: The categorization of "Automated" assumes the use of automated testing frameworks and tools for API testing. However, it's important to note that the feasibility of automation depends on the specific tools, framework

* Part B - Test automation
  + Create from scratch simple automation framework(s) in Java/Maven;
    - Cucumber/Selenium/TestNG;
    - Selenium/TestNG;
  + Implement 2 or more automation test cases on the framework(s) you created
  + In Postman develop 4+ Rest-API automation tests for Department’s APIs;

MZ- Department id generated is not in sequence, for example for every new add the number gets incremented by 1, but when deleted the sequence number changes

Part - C

* User Experience Improvement Report
  + Prepare a UX testing report using the template below
    - Issue: a short description of behaviour identified
    - Location: which page was it found
    - Level of Effect on UX: is it a blocking issue, user experience improvement, specific flow bug, etc
    - Recommendation: what can be improved in the system
  + Feel free to add more categories

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Issue | Location | Level of Effect on UX | Recommended Improvement |
| Website objectives | Unclear website objectives | Homepage Level of Effect on UX | User experience improvement | Clearly communicate the purpose and goals of the website on the homepage. This can be achieved by providing a concise and engaging tagline or heading that immediately conveys the value proposition and intended audience. |
| Navigation | Confusing navigation Location | Main navigation menu | Blocking issue Recommendation | Simplify the navigation menu by grouping related sections and using clear and descriptive labels for each menu item. Consider implementing visual cues, such as highlighting the current page or providing a breadcrumb trail, to improve orientation within the website |
| UI Design | Lack of responsive design Location | Mobile and tablet devices | User experience improvement | Establish a consistent visual design language throughout the website, including color schemes, typography, and button styles. This will enhance usability and create a cohesive and professional appearance |
| User Flows | Navigation from Employees to department or vice versa | Add of employees or departments | Increases overall usability and user engagement | Provide an option to add or delete employees from a particular department  Add multiple employees in bulk via excel or csv upload |

By addressing the identified issues and implementing the recommended improvements, the website can enhance the overall user experience, increase user engagement, and improve conversion rates.