Project Proposal

Team Formation Reprise

Team Name: 100-of-100

Repository Link: https://github.com/Alex-XJK/100-of-100-service-team

Language/Environment Selection: Java/MacOS

Team Members:

Yifei Luo (yl5357), Xintong Yu (xy2482), Phoebe Wang (kw3036), Jiakai Xu (ax2155)

Nothing has changed since our initial team formation assignment.

Service Overview

Our proposed service is a Human Resource Management system designed to facilitate staff and employee management across a diverse range of organizations, including hospitals, police stations, fire stations, and various other companies. The target user group includes organizations that require a flexible and efficient solution to manage internal human resources operations, enabling them to efficiently query and update personal data, track team status, and oversee departmental structures.

The service maintains persistent data including organizational structures, employee records, and their roles within teams, departments, and companies. This data will be securely stored in a database to ensure persistence and accessibility, even in the event of an unexpected service termination or restart. To guarantee data separation and security, each client organization will have its own distinct dataset, accessible only by authorized personnel.

We will provide a comprehensive set of API endpoints to perform key operations, such as:

- Managing departments and subgroups within an organization.
- Allocating employees to various teams or departments.
- Querying hierarchical data to analyze workforce distribution and team performance.
- Facilitating employee transfers between departments, following predefined rules or constraints (e.g., requiring more than 3 years of experience for reassignment to another department).

We will add business logic for more complex operations for api endpoints mentioned above. For example, employee allocation or transfers can follow specific rules, like experience-based constraints, ensuring that business requirements are met. Additionally, the service will offer workforce and performance analysis, enabling clients to generate insights into their team dynamics and overall organizational health. For example, workforce metrics could include average team size or skill distribution, while performance metrics could provide insights like average performance levels across departments, top and bottom performers, and trends over time. These analyses can be returned as structured summaries, allowing client applications to integrate them into their dashboards or reporting tools for further use.

Client Overview

1. A Hospital Management System

- User Group: Healthcare workers (doctors, medical technicians, nurses, aides, etc.)
- service would support hospital administrators, department heads, and HR personnel in managing the diverse workforce of healthcare professionals, including doctors, nurses, and support staff.
    - Managing healthcare departments and subgroups.
    - Allocating healthcare staff to teams and departments.
    - Querying hierarchical data to assess workforce distribution and performance.
    - Facilitating employee transfers between departments under predefined rules.

2. A First Responder Management Platform

- User Group: First responders (police, fire, EMT)

  The service would support public sector administrators, emergency unit leaders, and HR personnel in managing the workforce of first responders, including police officers, firefighters, and emergency medical technicians (EMTs).
    - Managing first responder departments and emergency units.
    - Allocating first responders to various teams or emergency assignments.
    - Querying hierarchical data to assess workforce distribution and performance.

- Facilitating personnel transfers between units under predefined rules (e.g., specific training or certifications).

Testing Overview

1. **Unit Testing**

- Tools:
  - JUnit: For writing and executing unit tests in Java.
  - Mockito: For mocking dependencies and isolating units of code.
  - JaCoCo: For measuring code coverage.
- Units to Test:
  - Data Validation Functions: Methods that validate user input, such as employee data (e.g., name, age, role) and organizational structures.
  - Business Logic Components: Functions that implement core business rules, such as checking eligibility for employee transfers based on experience or enforcing team size constraints.
  - Database access: Checks for correct CRUD operations on employee records, departments, and organizations.
  - Authentication Handlers: Components managing client authentication and authorization, ensuring security and proper access for different clients.
  - Utility Functions: Helper methods used throughout the application.
- Develop Test Cases:
  - Positive Tests: Test with valid inputs to ensure units return expected results. (For example, successfully retrieving an employee with all relevant information)
  - Negative Tests: Test with invalid inputs to check error handling and input validation. (For example, attempting to assign an employee to a non-existent department)
  - Boundary Tests: Test with edge case values. (For example, assigning an employee with exactly the required years of experience to a new department)
- Continuous integration:
  - Use GitHub Actions to automatically run unit tests on code commits.

- Code Coverage Analysis:
    - Aim for at least 85% code coverage.

2. **Integration Testing**
    - Tools
        - JUnit: For writing integration tests.
        - Spring Boot Test: Has extensive support for Java integration testing, including the @SpringBootTest annotation
    - Integration Points:
        - Database Interactions: Test database access with a test database to ensure data persistence and retrieval works as expected.
        - Service Layers: Test the interaction between service classes and databases.
        - Authentication: Verify that users with different roles have appropriate access levels, and each client app only has access to its own database.
        - Data Serialization: Ensure data is correctly converted between formats (e.g., JSON to Java objects).
    - Develop Test Cases:
        - End-to-End Scenarios: Simulate workflows that involve multiple components. For example, creating a new department, adding employees to it, and generating a workforce report.
        - Error Handling: Test how the system behaves when one component fails. For example, error handling in case of database connection failure.

3. **API Testing**
    - Endpoints:
        - Employee Endpoints: CRUD operations on employee data.
            - GET /employees/id: Retrieve employee information.
            - POST /employees: Add a new employee.
            - PUT /employees/id: Update employee information.
            - DELETE /employees/id: Remove an employee.
        - Department Endpoints: Manage Departmental structures.
            - GET /departments/id: Retrieve department information.
            - POST /departments: Create a new department.

        - PUT /departments/id: Update department details.
        - DELETE /departments/id: Delete a department.
    - Assignment Endpoints: Assign employees to departments or teams.
        - POST /departments/e_id/d_id: Add an employee to a department.
        - POST /transfers/e_id/d_id: Transfer an employee to a new department.
    - Summary Endpoints: Generate workforce reports.
        - GET /reports/workforce: Get workforce distribution metrics.
        - GET /reports/performance: Get performance analysis.
    - As well as new endpoints we develop in later parts of the project.
- Test Cases:
    - Positive Tests: Valid requests should return successful responses with correct data.
    - Negative Tests: Invalid requests should return appropriate error responses. For example, deleting a department that does not exist.
    - Security Tests: Ensure unauthorized access is correctly blocked.

4. Load Testing

- Stress testing the system beyond its limits to see how it performs under high load.
- Test Cases:
    - Concurrent Users: Simulate multiple users performing actions simultaneously, such as many users updating records at the same time.
    - High Frequency of Requests: Users send rapid requests.
    - Data Volume: Users send/receive large amounts of data, such as importing a large list of employees.
- Performance Metrics:
    - Response times.
    - Throughput.
    - Error rate.