TechQuest

Context:

RIWI TechQuest has decided to build a modern and secure platform to manage missions and training for its students (heroes in training). On this platform, teachers will be able to assign missions and skills to students according to their progress and abilities, while students can participate in missions and view their assigned skills. The platform must be secure, role-based, and provide a detailed record of all key actions to ensure transparency and control.

Given the critical nature of the information, the system must ensure secure authentication and authorization using JWT, role management (enum), and an auditing system that automatically tracks which users created, modified, or deleted key entities within the system.

Objective:

Develop a system that allows RIWI to manage students, teachers, missions, and special skills securely, efficiently, and at scale. The system must include automatic auditing, robust exception handling, and difficulty levels for missions. Additionally, optional points will be awarded for implementing features like granular permissions, multi-database support, and integration with an external API for multimedia file storage.

Scope:

For the first version of the application, developers are expected to implement the following functionalities:

1. **Roles and Authentication with JWT:**
    a. The system must implement authentication using JWT.
    b. Users should be categorized into three roles: ADMIN, TEACHER, and STUDENT, defined as an enum.
        i. **ADMIN**: Can manage the entire system (users, missions, skills).
        ii. **TEACHER**: Can manage their own classes, missions, and assigned students.
        iii. **STUDENT**: Can only view their assigned missions and skills.

2. **Skills and Mission Management:**
    a. Teachers must be able to assign skills to students. Each student can have multiple assigned skills, categorized by type (e.g., SUPER_STRENGTH, FLIGHT, INVISIBILITY...).
    b. Teachers must be able to create missions and assign them to students. Missions must be classified by difficulty level (EASY, MEDIUM, HARD), and associated with the skills needed to complete them.

3. **Automatic Auditing System:**
    a. The system must automatically track who creates, modifies, or deletes missions, students, and skills.
    b. This logging must be implemented using Spring Data JPA's auditing capabilities (@CreatedBy, @LastModifiedBy).

4. **Exception Handling:**
    a. Implement a robust exception-handling system using @ControllerAdvice to capture and return clear and consistent error messages. The system must handle:
        i. Unauthorized access.
        ii. Resource not found.
        iii. Validation errors.

5. **Swagger Documentation:**
    a. All API endpoints must be documented using Swagger, making it easy to test and consult the API.

6. **Mission Difficulty Levels:**
    a. Each mission must have a difficulty level (EASY, MEDIUM, HARD), assigned by the teacher when creating the mission.
    b. Only students with the necessary skills can participate in higher-difficulty missions.

7. **Production:**
    a. Deploy the application on a free server such as Railway or Render.

Optional Features

1. **Multi-Database Support (MySQL or PostgreSQL):**
    a. The application should support both databases using Spring profiles (application-mysql.properties and application-postgresql.properties).
    b. **Bonus Points**: Additional points will be awarded if the system supports both databases without requiring changes to the codebase.

2. **Granular Permissions:**
    a. Implement a more detailed permission system, where roles not only define allowed actions but also support assigning specific permissions such as CAN_CREATE_MISSIONS or CAN_ASSIGN_ABILITIES.
    b. **Bonus Points**: Additional points will be awarded for developers who implement this feature.

3. **Integration with External API for File Storage:**
    a. Implement the ability to store multimedia files (images, videos, documents) associated with missions using an external API.
    b. Recommended options for Java include **Cloudinary**, which offers an easy-to-use API for cloud-based multimedia file storage.
    c. Other alternatives include **Amazon S3** and **Google Cloud Storage**, both scalable and Java-compatible.
    d. **Bonus Points**: Additional points will be awarded for integrating an external API for file storage.

Requirements:

1. **Task ID: task-dev-01 - Entity: Auth**
    a. **Objective**: Authenticate users via JWT.
    b. **Description**: The system must authenticate users by receiving their email and password and returning a valid JWT if the credentials are correct.
    c. **Endpoint**: /api/v1/auth/login
    d. **HTTP Method**: POST
    e. **Acceptance Criteria**: The system must accept an email and password, and return a JWT upon successful authentication.

2. **Task ID: task-dev-02 - Entity: Student**
    a. **Objective**: Assign skills to students.
    b. **Description**: Teachers should be able to assign special skills to students via a dedicated endpoint. Each student can have multiple skills associated with them.
    c. **Endpoint**: /api/v1/students/{id}/abilities
    d. **HTTP Method**: POST
    e. **Acceptance Criteria**: The system must validate that the provided skills exist, and correctly assign them to the corresponding student.

3. **Task ID: task-dev-03 - Entity: Mission**
    a. **Objective**: Create missions and assign them to students.
    b. **Description**: Teachers should be able to create missions and assign them to specific students. Each mission should be associated with the skills required to complete it.
    c. **Endpoint**: /api/v1/missions
    d. **HTTP Method**: POST
    e. **Acceptance Criteria**: The system must validate that the students assigned to the mission possess the necessary skills to complete it.

4. **Task ID: task-dev-04 - Entity: Mission**
    a. **Objective**: Evaluate student performance in missions.
    b. **Description**: After a student completes a mission, the teacher should be able to evaluate their performance, and the system should store this data.
    c. **Endpoint**: /api/v1/missions/{id}/evaluate
    d. **HTTP Method**: PATCH
    e. Acceptance Criteria: The system must record the student's progress and performance for the mission, and update their record accordingly.

Deliverables:

1. Public GitHub repository link where the source code is hosted, including the respective branches and Pull Requests.
2. A compressed zip file containing the finalized code, Swagger documentation, and configuration details.
3. Detailed instructions on how to configure and run the application, including setting up database profiles (MySQL or PostgreSQL) and integrating the external API for multimedia file storage (if implemented).

Additional Notes:

- All sensitive information (e.g., JWT credentials or API keys for external storage) must be handled via environment variables.
- Data validation and security must be strict to prevent unauthorized access or malicious inputs.

"The only way to do great work is to love what you do."

Steve Jobs