STDISCM P4
Distributed Fault Tolerance

DIMAGIBA, Rafael
GARCIA, Aurelio
PARK, Sehyun
SILLONA, Eugene

Task Description

We are tasked to create an **online enrollment system** with different services distributed across multiple nodes.

The system should have the following bare-minimum features:

1. **Login/Logout:** Track sessions across different nodes. (Use OAuth / JWT)
2. **View** available courses.
3. Students to **enroll** to open courses.
4. Students to **view previous grades**.
5. Faculty to be able to **upload grades**.

The application should be web based using MVC.
The view will be a node on its own.
The rest of the features / API / controllers will also be on a separate node.

Use of networked virtual machines/bare-metal computers is recommended.
When a node is down, the features supported by that node should stop working, but the rest of the application should still work.

Key Implementations

Login / Logout

- The user enters their username and password into the **login.html page**.
- The **api.js file** sends these credentials to the server's “/auth/login” endpoint.
- The **AuthController.java** uses **Spring Boot** to verify the submitted credentials against the database.
- If valid, **JwtUtil.java** generates a JWT, and the server sends it back to the client.
- The **api.js** files keeps the JWT in localStorage and redirects the user to the appropriate dashboard.
- **WebConfig.java** enables CORS to allow communication between the frontend and backend.

View Courses

- When a user accesses their dashboard, the **dashboard.js** script runs.
  - Students: **dashboard.html**
  - Faculty: **faculty-dashboard.html**
- The **dashboard.js** file retrieves the JWT to authenticate the user.
- The **dashboard.js** fetches the list of courses from the /courses endpoint using a GET request.
- **CourseController.java** uses **Spring Boot** to handle the /courses request, retrieves course data from the database using **CourseRepository.java**, and returns it as a JSON response.
- The **dashboard.js** file receives the JSON data and dynamically generates the list of courses on the page.
- The user sees the list of available courses with enrollment status and options.

Course Enrollment

- The student views the list of courses on **dashboard.html**, which is populated by dashboard.js.
- If a course has available slots and the student isn't already enrolled, an "Enroll" button is displayed.
- Clicking "Enroll" on a listed course on **dashboard.js** sends a POST request to the /enrollments endpoint with the courseId and the JWT for authentication.
- **EnrollmentController.java** handles the request, validates the enrollment, creates an Enrollment record, and updates the course's available slots.
- **EnrollmentController.java** uses **Spring Boot**, various functions from **EnrollmentRepository.java**, and
- The server responds with a success or error message, and dashboard.js updates the UI accordingly.

Grade Uploads

- When the faculty navigates to the **faculty-dashboard.html**, they can upload grades.
- When a faculty user uploads a grade, the **faculty.js** file sends a POST request to the /grades/upload endpoint with the target student's enrollmentId and the grade in the request body.
  - The JWT is included in the Authorization header for authentication.
- **GradeController.java** uses **Spring Boot** to process this request and verify that the user is a faculty member by extracting the username from the JWT.
- **GradeController.java** accesses the **UserRepository.java** to retrieve the corresponding student's Enrollment record from the database.
- The server updates the grade for that enrollment and saves the changes, then sends a response indicating success or failure.

Fault Tolerance

Fault Tolerance

- Basic Fault Tolerance is implemented throughout the backend of the application.
- Architecture is split into controllers, services and repositories so that they are not reliant with each other.
- ResponseEntity allows us to return clear error message just in case the client wants to know what went wrong.
- JwtUtil is used to embed the userId in a signed JWT token

- In AuthController:
  - Fault tolerance for invalid credentials by checking the token of the user.
- In EnrollmentController:
  - Checking the course if:
    - The course is not found
    - The course is full
    - The user has already enrolled
- Checks for data validation such as if the user actually inputs or not.
- GradeController:
  - Checks if the user can edit the grades
  - Checks if the user has valid enrollment

Separation of Nodes (Micro-services)

- Each microservice is independently deployed and run, allowing individual features to operate independently without affecting the entire system.
- By separating the enrollment logic from grade upload logic, failures can be localized. For example, failure of grade upload microservice will not disrupt other part of the application such as login
- The frontend is designed to detect when a node (micro service) is down and displays a fallback messages, updating the user with information about disruption.
- API calls from the frontend to microservices are explicitly routed to their corresponding ports (In our case the Micro service listens to port 8080 and 8082 respectively). Adding more micro services is very scalable to the application.
- Testing was conducted by shutting down one service (grade upload node was killed) to confirm that the rest of the application remains responsive and intractable.
- This mirrors real world fault tolerance system, where critical services are distributed and load-balanced to minimize disruption from individual service failures.