🎓 Capstone Projects for Engineering Grads (React + Spring Boot)

Team Composition

- **Total Members:** 5-6
  - **3-4 Backend Developers** (Java + Spring Boot)
  - **1-2 Frontend Developers** (React + Next.js 15)

Each **assignment is divided among all 5-6 developers**, ensuring every member has one dedicated responsibility per capstone.

Shared Responsibilities

- **Git & Branching:** All developers follow Git workflow with PRs
- **API Contract Design:** Backend devs collaborate using Swagger
- **Documentation:** Rotated weekly (README, Insomnia as replacement of postman, ER diagram)
- **Code Reviews:** Peer review (1 frontend devs with each other and backend dev within themselves)
- **CI/CD (Optional):** Devs can experiment with GitHub Actions and docker
- **Testing:** Backend Dev 3 + frontend dev 2, write relevant test cases

Technologies to be Used

Frontend: Next.js 15 (App Router), Redux Toolkit, Axios, ShadCN UI, Zod, React Testing Library, Jest

Backend: Java Spring Boot, PostgreSQL, MongoDB, Kafka, Apache POI, Apache Camel, Resilience4j, JUnit, Mockito

🎓 Capstone Project 1: **Secure Auth + Resilience + Logging**

Objective: Implement a secure login/logout mechanism using JWT authentication. Include proper session management, error logging, and resilience patterns. Ensure system-level logging and structured exception handling are in place.

Backend Developer 1 & 2

Task: Implement Authentication APIs using JWT (No database)

You are responsible for creating a token-based login system using **Java + Sprint boot + JWT**, **without using any database**. Store valid tokens **in memory**.

APIs to implement:

1. **Login API** (POST /login)
    a. Accepts a hardcoded username and password.
    b. If correct, returns a signed JWT token.
    c. Tokens should be stored in-memory for validation.
2. **Auth API** (GET /auth)
    a. Accepts JWT in the Authorization header.
    b. Validates the token and confirms if the session is still valid.
    c. Returns decoded user info if token is valid.
3. **Logout API** (POST /logout)
    a. Accepts the token and removes it from the in-memory list.
    b. This prevents future use of the token.

Backend Developer 3/4

Task 1: Logging & Error Handling

1. **Setup SLF4J + Logback**
    a. Use Spring Boot or Java backend (or your preferred language stack that supports SLF4J).
    b. Enable logging of incoming requests, responses, and errors using Logback configuration.
2. **Global Exception Handler**
    a. Add a centralized error handler using @ControllerAdvice or middleware.
    b. Return meaningful error messages to the frontend with proper status codes.

Task 2: Add Resilience with Mock External API

- Integrate a **login API call**
- Use **Resilience4j** to:
  - Rate Limiter: Prevent brute-force attacks by limiting the number of logins attempts per user/IP.
  - Circuit Breaker: Open the circuit if the login service is failing consistently, to avoid overwhelming it.
  - Add fallback method when API fails.
- This will simulate real-world external service failure and graceful degradation.

Frontend Developer 1

Task: UI for Login + Session Handling

1. **Build Login and Logout Pages**
   a. Simple UI with input fields for username and password.
   b. Show success/error messages based on API response.
2. **Secure Route Access**
   a. Once the user logs in, it shows protected routes / unaccessible (e.g., a dashboard).
   b. If the user is not authenticated, redirect to login page.
3. **Session Timeout Handling**
   a. If the user is inactive for **more than 5 seconds**, automatically log them out and redirect to login.
   b. Show a message like: "Session expired due to inactivity".
4. **Error Handling**
   a. Show user-friendly error messages for login failure (e.g., “Invalid username or password”).

Frontend Developer 2

Task: Token Storage, API Handling, Global Errors

1. **Manage state**
   a. Store and manage JWT token in local storage or cookie.
   b. Track login status using cookie / local storage state.
2. **Axios Interceptor**
   a. Intercept all outgoing API requests.
   b. Automatically attach the JWT token in Authorization header.
   c. Redirect to login if any 401/403 error occurs.
3. **Error Boundary**

    a. Add a global error boundary to catch UI-level errors.
    b. Display fallback UI in case of rendering issues.

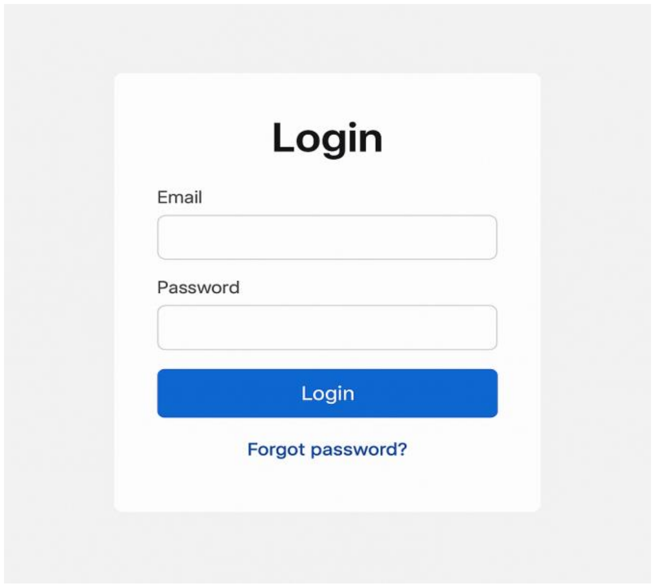
4. **Session Timeout Logic**
    a. Track user activity (mouse movement, keyboard input).
    b. If inactive for **5 seconds**, clear the JWTtoken and navigate to login page.

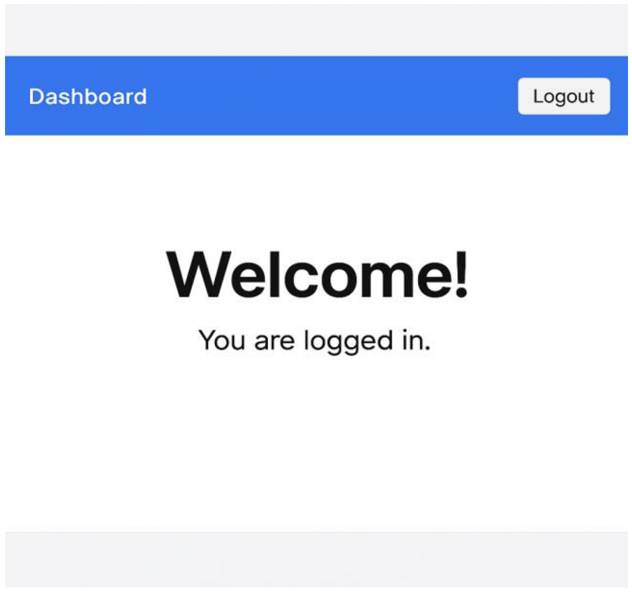
💡 Objective: Learn storage management, secure API access, and global error handling in modern React apps.

Final Output Expectations

- Working login/logout with token flow.
- Tokens stored securely (in-memory backend, Redux frontend).
- Resilient backend for external API.
- UI feedback for all states (success, error, timeout).
- Clean and modular code with basic error logging and handling.

UI Mockups:


Login
Email
Password
Login
Forgot password?


Dashboard
Logout
Welcome!
You are logged in.