**Software Design Document (SDD)**

**1. Introduction**

**1.1 Purpose**This document outlines the design for the UPL Tutorial Platform, an online text-based learning portal supporting instructor contributions (after admin approval), YouTube video embedding, document uploads, and structured learning with real-time notifications and performance metrics.

**1.2 Scope**The UPL Tutorial platform now supports:

* Guest access to explore courses and tutorials.
* Instructor registration and admin approval workflow.
* Instructor dashboard for managing courses and tutorials (CRUD).
* Course document uploads with Apache Tika-based validation.
* Notification system via email (using JavaMailSender and Mailtrap).
* Role-based comment and reply system with moderation.
* Course views tracking and trending chart for admins.
* System observability via Prometheus + Grafana.

**1.3 Audience**This document is intended for developers, project reviewers and team members for the evaluation of the UPL Tutorial Platform.

**2. Project Overview**

**2.1 Project Description**UPL Tutorial enables guest users to access educational content, while instructors (after approval) can contribute structured textual and video-based tutorials. Admins manage the platform and monitor activities.

**2.2 Functional Requirements**

• Instructor registration and approval by Admin

• Instructor dashboard for Course & Tutorial CRUD

• YouTube video embedding

• Guest access without login

• Admin panel for user/content/analytics management

• Course search

• Course view tracking

• Course rating and comments by users

• Replies by instructors/admins; abusive comment deletion

• Real-time system metrics via Grafana

• Email notifications for:

* Instructor registration (approval/rejection)
* Course/tutorial submission/approval/rejection
* New user comments

**2.3 Non-functional Requirements**

* Security: Role-based access control, secure registration, and JWT-based authentication/authorization
* Usability: Simple and clean UI using Angular
* Performance**:** In-memory caching with Redis, observability with Prometheus & Grafana
* Reliability: Apache Tika for content validation

**3. System Architecture**

**3.1 Architectural Design**

The system follows a three-tier architecture:

* Frontend: Angular application
* Backend: Spring Boot REST APIs
* Database: MySQL for persistent storage
* Caching: Redis
* Monitoring: Spring Boot Actuator, Prometheus, Grafana
* Containerization: Docker for all components
* Messaging: JavaMailSender + Mailtrap (SMTP)

**3.2 Technology Stack**

* Frontend: Angular
* Backend: Spring Boot (Java)
* Database: MySQL
* Monitoring: Spring Boot Actuator, Prometheus, and Grafana
* Containerization: Docker, Docker Compose
* Caching: Redis
* Mailtrap (Email testing)

**4. Detailed Design**

**4.1 Component Design**

* User Module: Handles registration, login, role validation
* Course Module: CRUD operations for courses
* Tutorial Module: CRUD operations and video embedding
* Admin Module: Instructor approvals, manage courses/tutorials, and analytics view, performance tracking via Prometheus + Grafana
* Comment System: Ratings, comments, replies, moderation
* Notification Module: Email alerts for approval/comments

**4.2 Data Design**

* User: id, name, email, password, role, status, created\_at
* Course: id, title, description, instructor\_id, created\_at
* Tutorial: id, title, content, youtube\_url, course\_id
* CourseEditHistory: history\_id, course\_id, instructor\_id, changes, modified\_at
* TutorialEditHistory: history\_id, tutorial\_id, instructor\_id, changes, modified\_at
* InstructorApprovalLog: log\_id, user\_id, admin\_id, status, remarks, created\_at
* CourseRating: id, course\_id, rating, comment, user
* CommentReply: id, comment\_id, reply, replied\_by, role, created\_at

**4.3 API Design**Course

* GET /api/course/fetchAllCourses
* POST /api/course/createCourse
* PUT /api/course/updateCourse/{id}
* DELETE /api/course/deleteCourse/{id}

Tutorial

* GET /api/tutorial/fetchAllTutorialsByCourseId/{courseid}
* POST /api/tutorial/createTutorial
* PUT /api/tutorial/updateTutorial/{id}
* DELETE /api/tutorial/deleteTutorial/{id}
* GET /api/tutorial/fetchTutorialById/{id}

Authorization

* POST /api/auth/register
* POST /api/auth/login

Admin

* GET /api/admin/pending-instructors
* GET /api/admin/active-instructors
* POST /api/admin/approve-instructor/{id}
* POST /api/admin/reject-instructor/{id}
* GET /api/admin/review-courses
* GET /api/admin/review-tutorials
* GET /api/admin/analytics

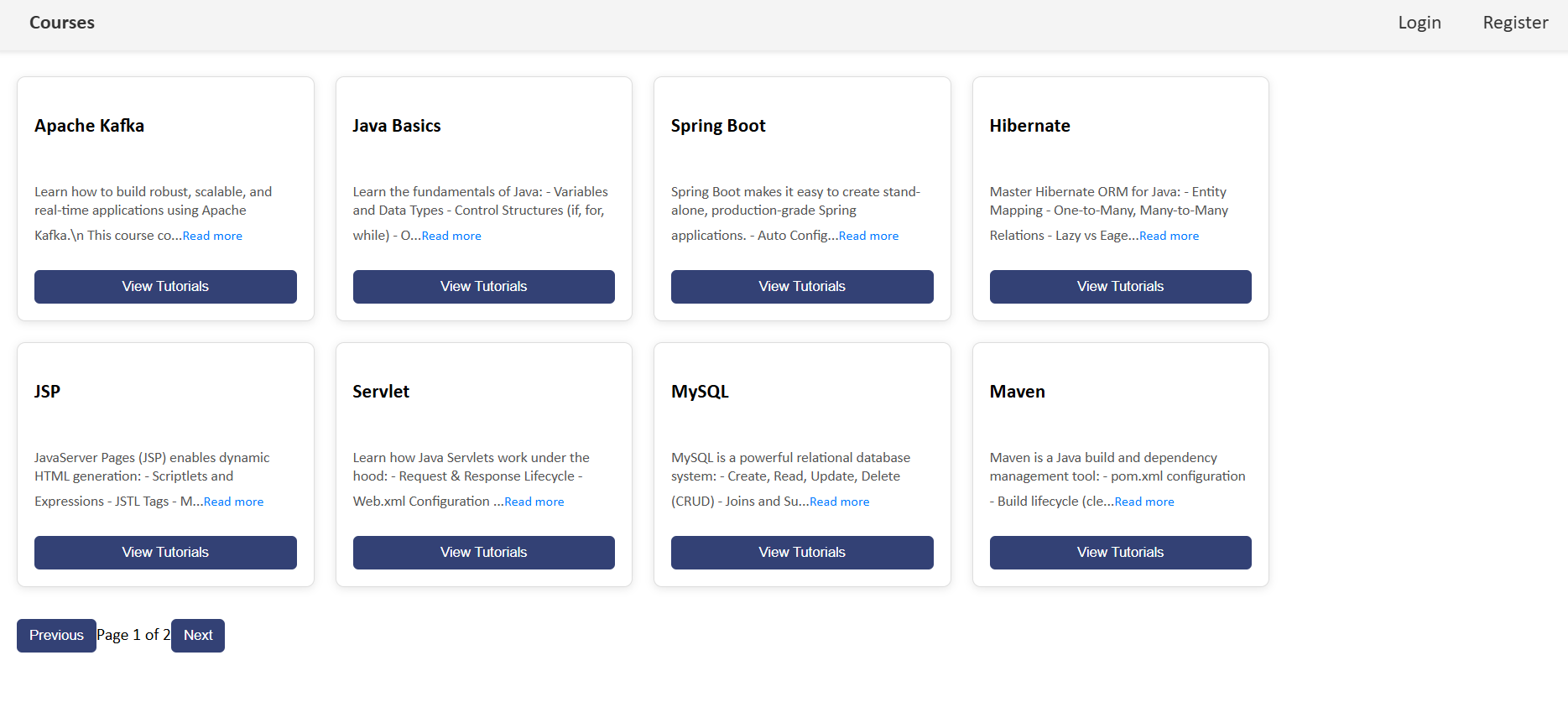
**Notification APIs**

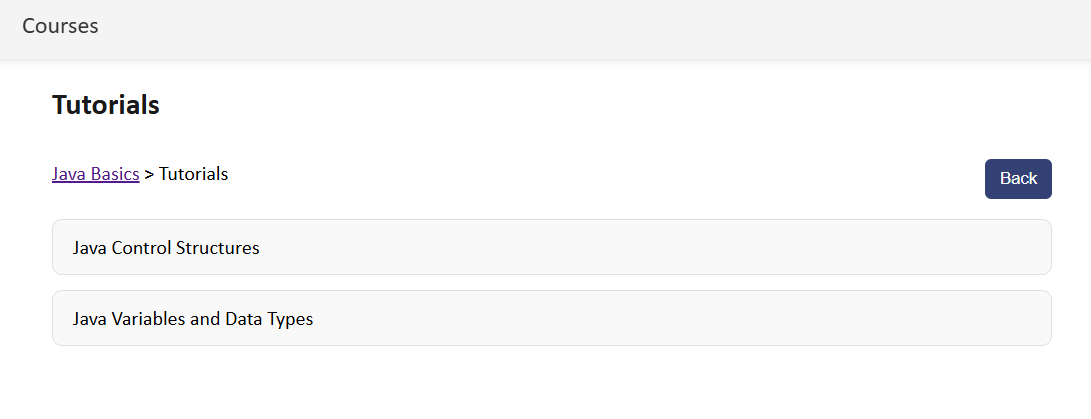
* Integrated via JavaMailSender backend on trigger events.

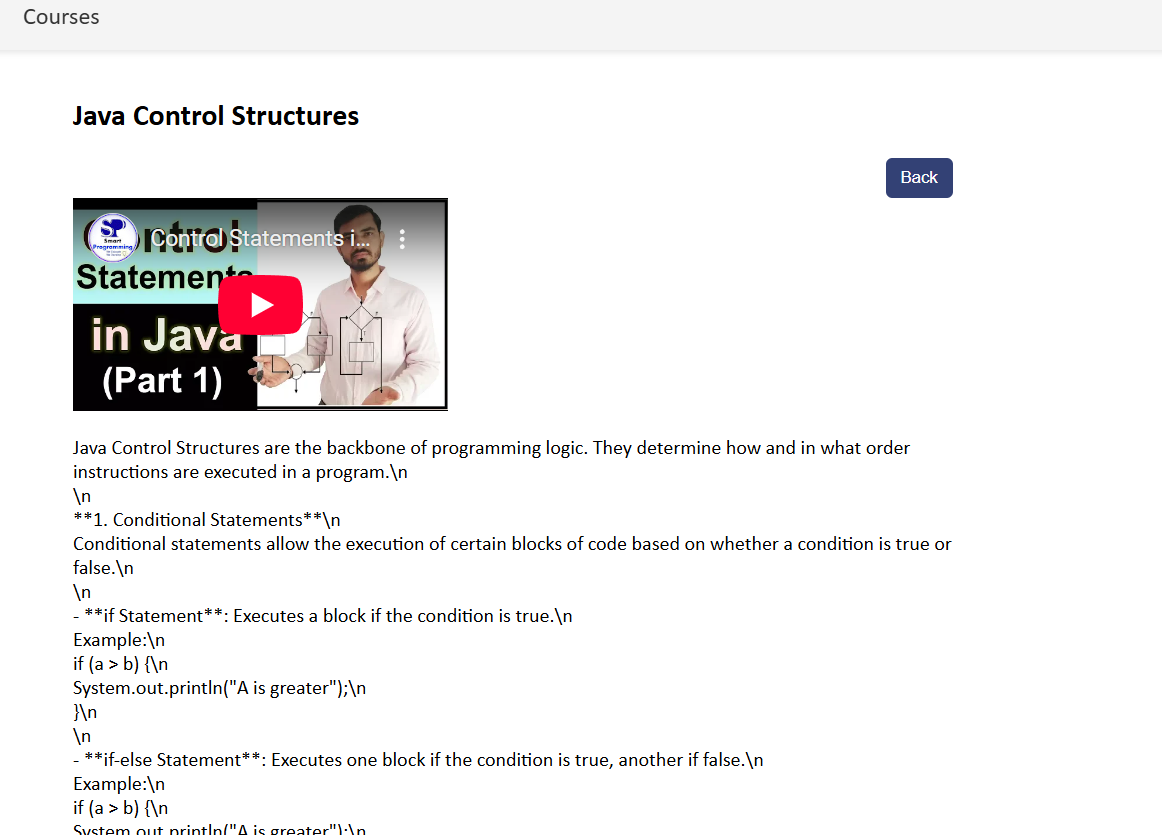
**5. User Interface Design**

**5.1 UI Mockups**

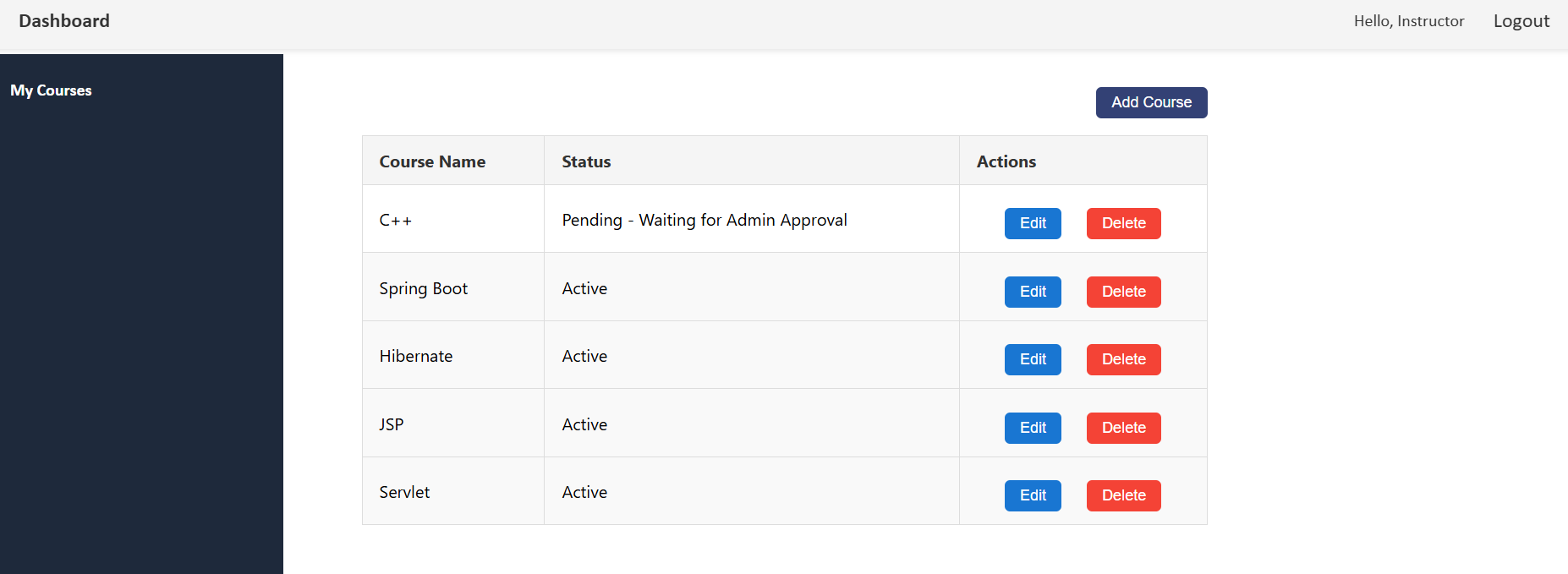
**General User Panel:**

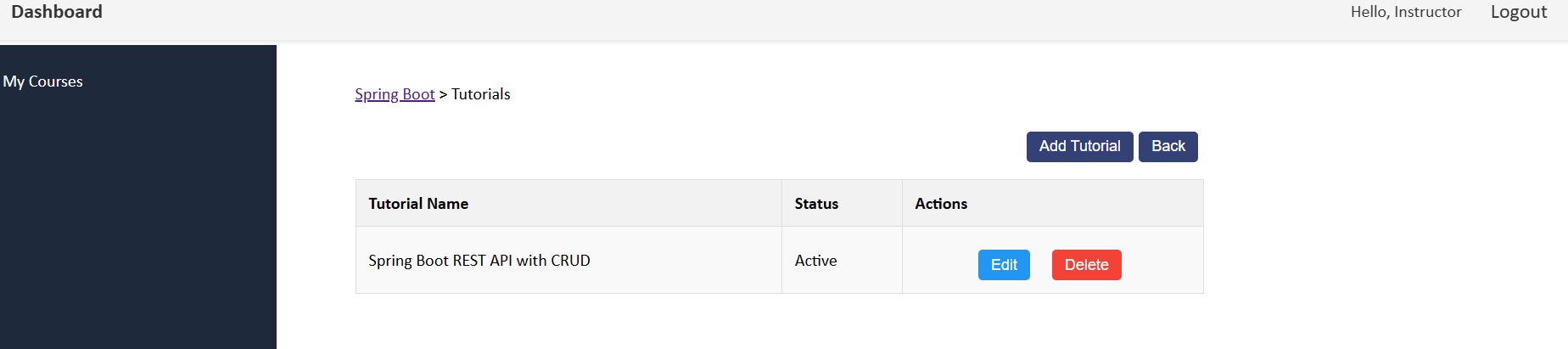


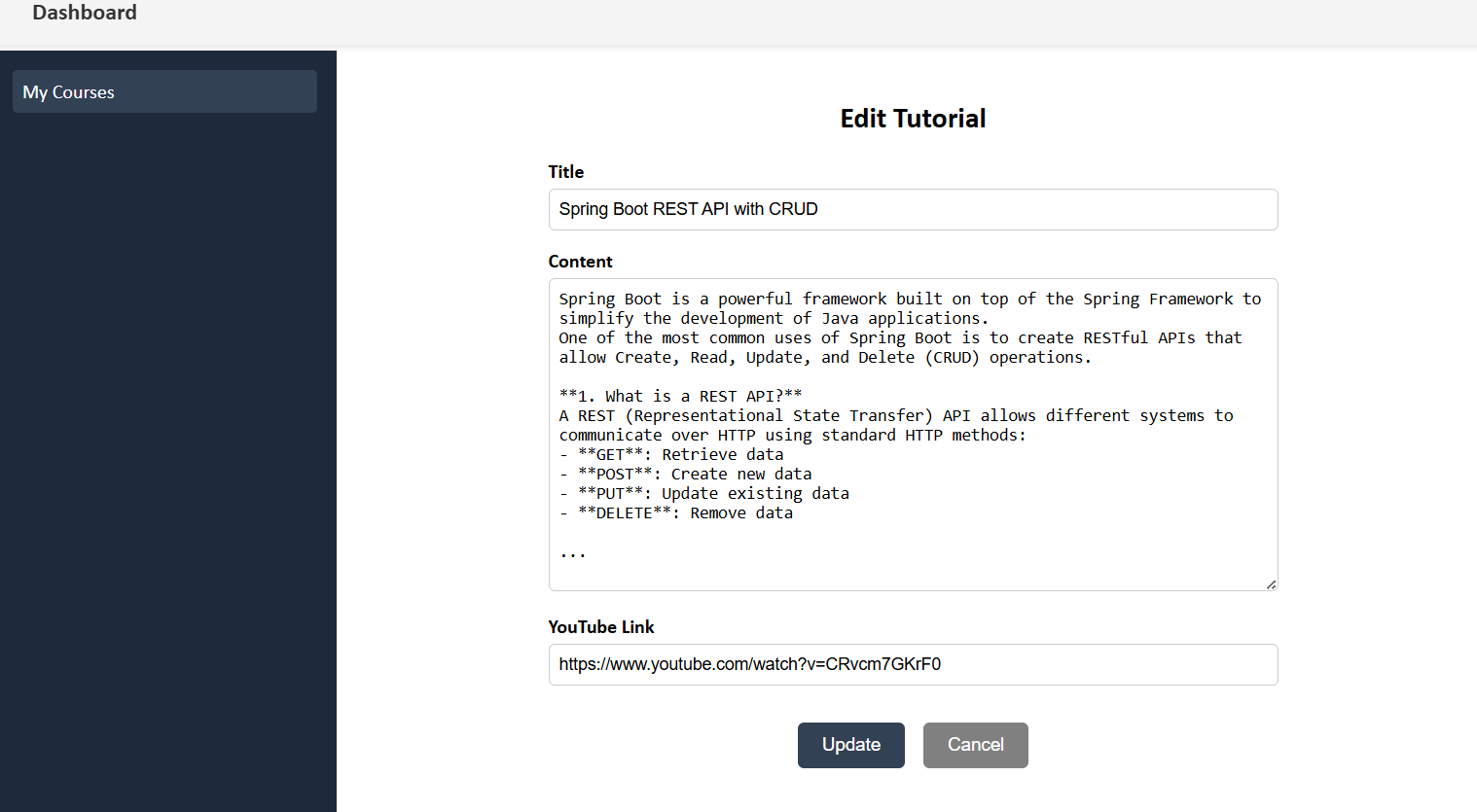


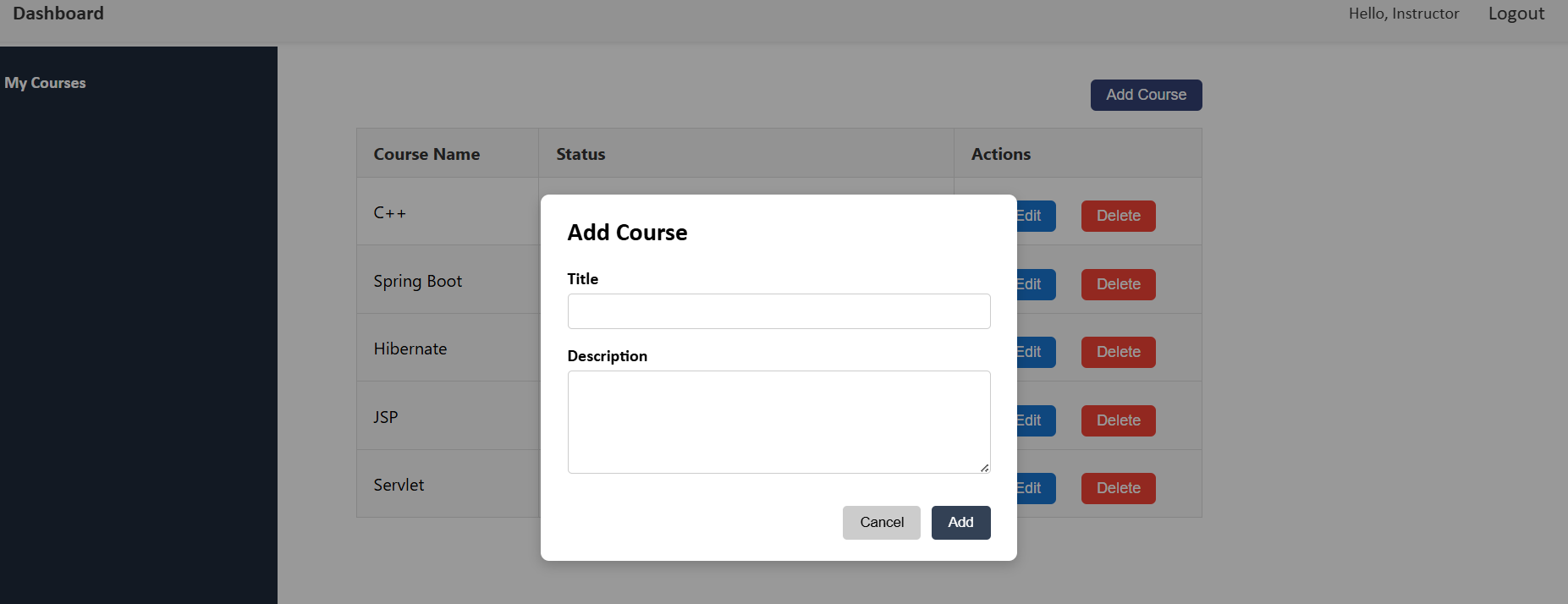


**Instructor Panel:**

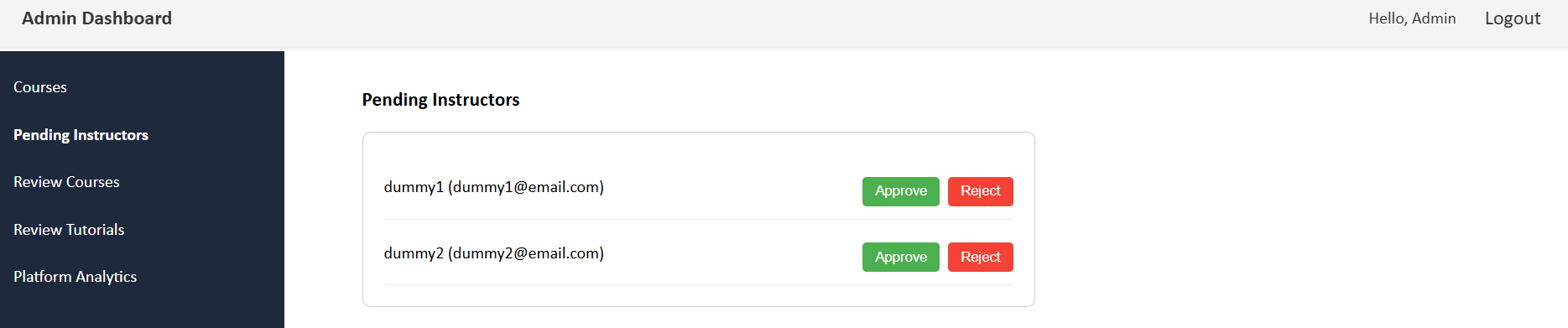


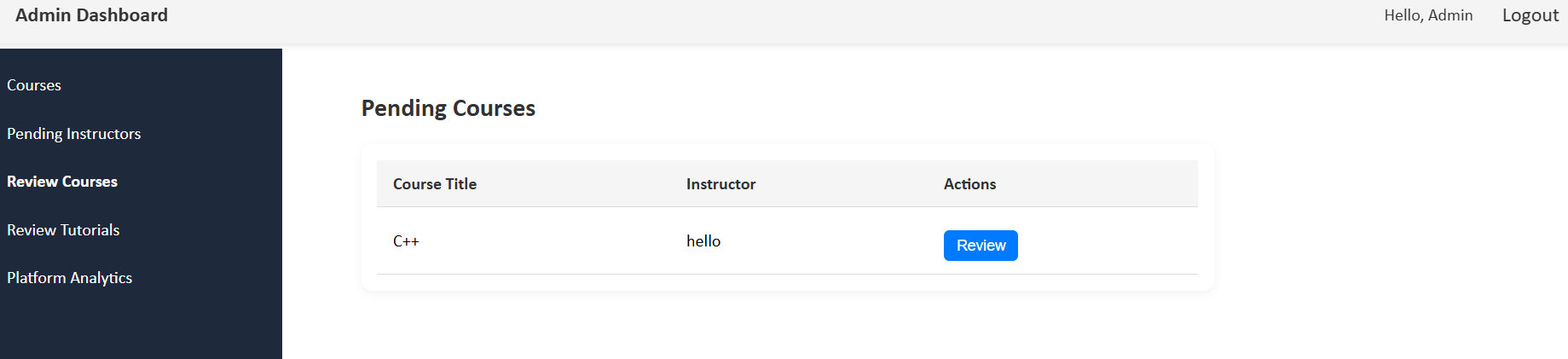


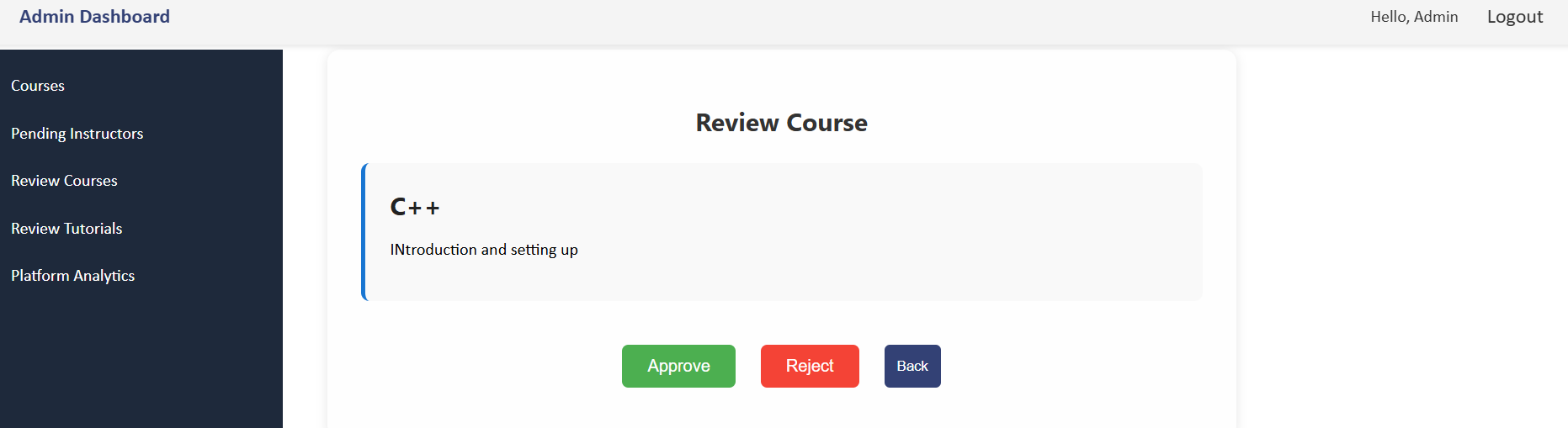


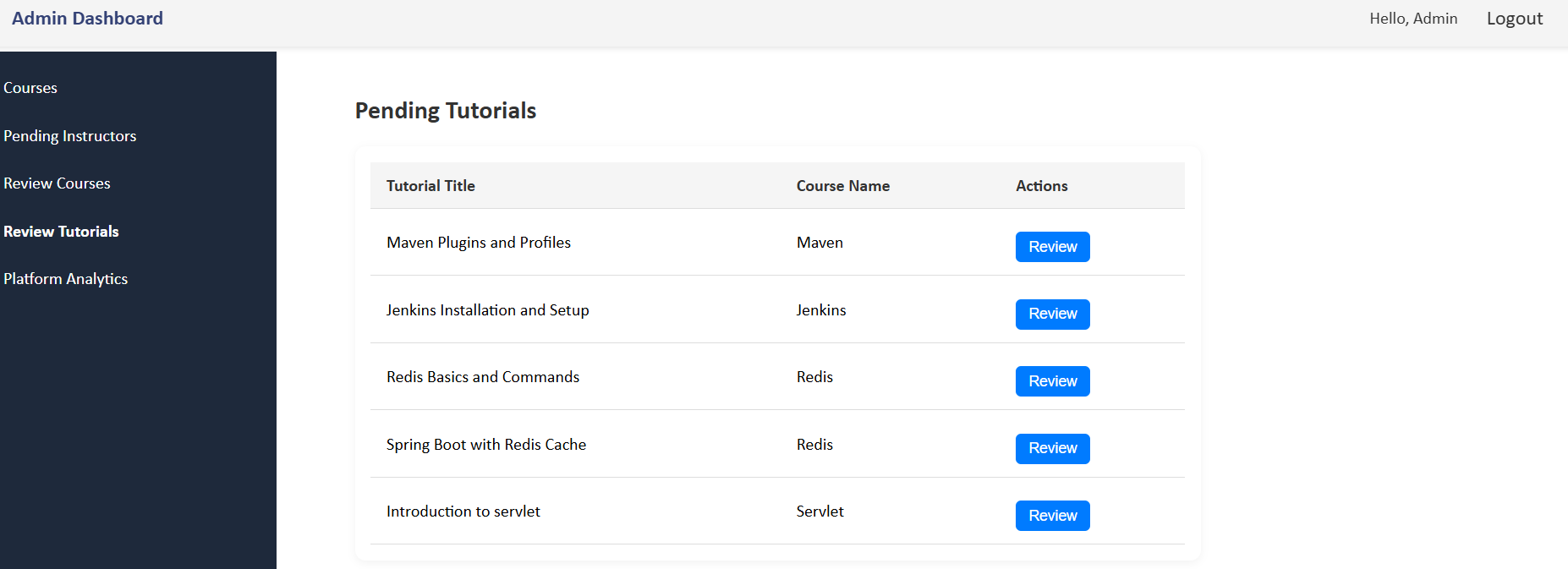


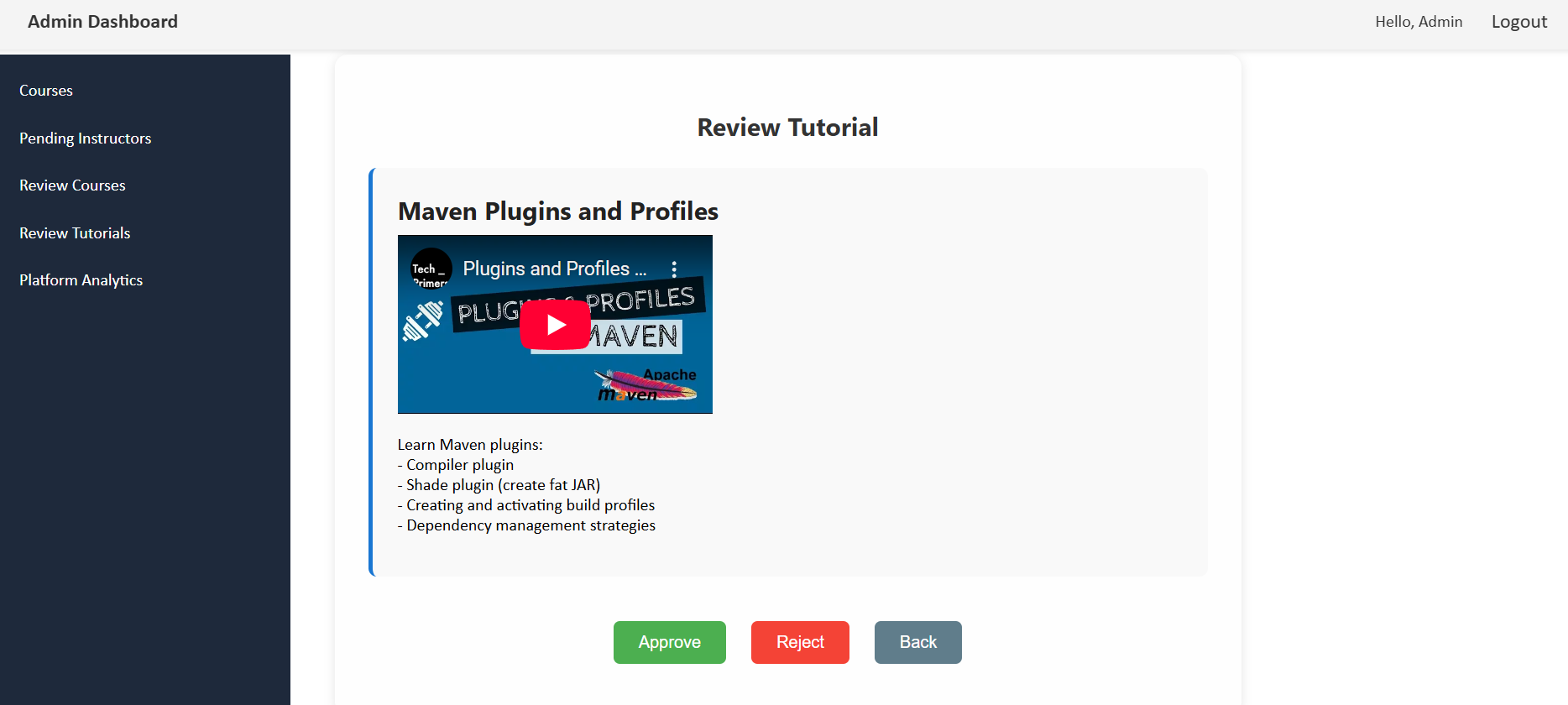
**Admin Panel:**

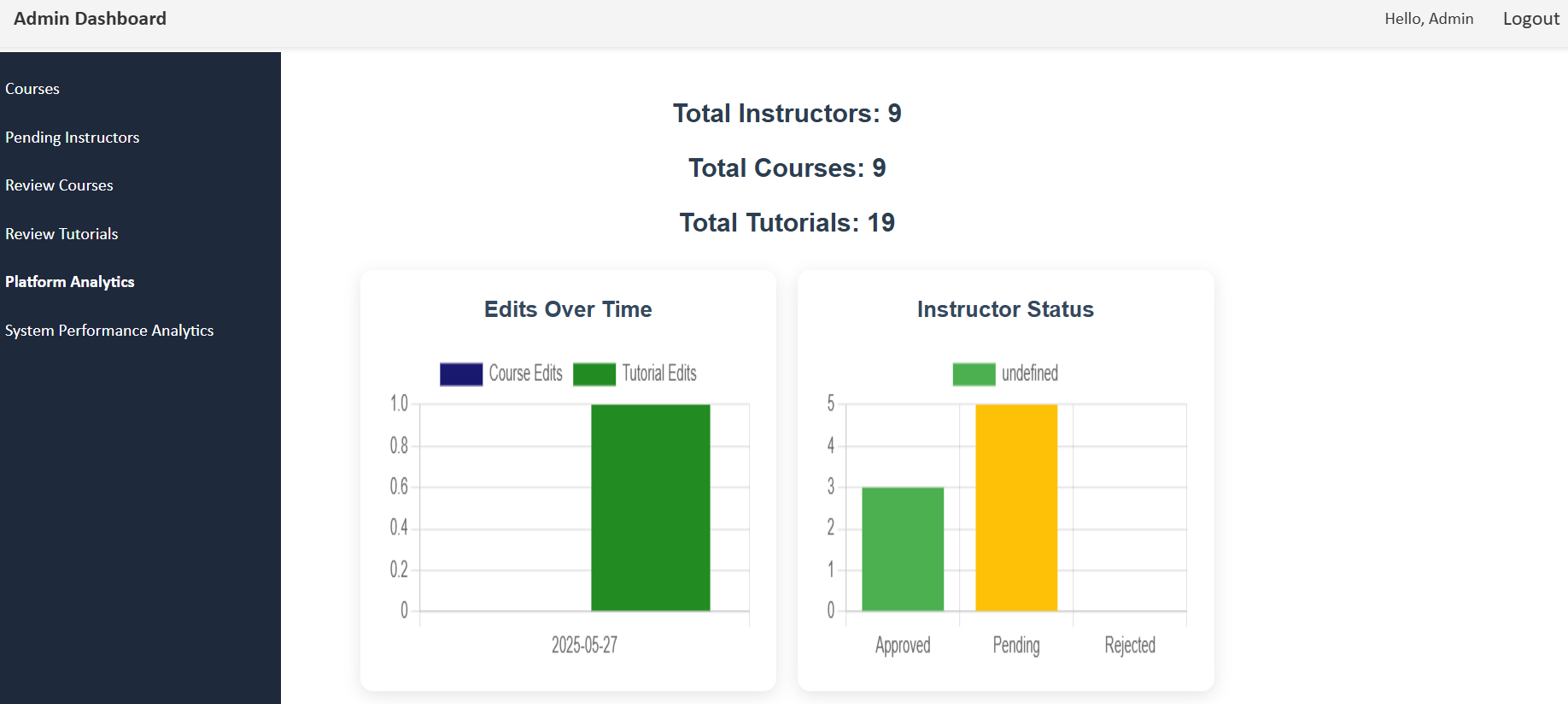












**5.2 User Interaction**

* Guest Users: Browse course/tutorial pages
* Instructors: Login to dashboard, manage content
* Admins: Login to approve instructors, manage content, view analytics
* Real-time updates via backend-driven logic
* Email notifications on course/tutorial/comment actions
* Course trending chart based on views & ratings

**6. Integration and Testing**

**6.1 Integration Plan**

* Integrate Angular frontend with REST APIs from Spring Boot
* Docker Compose integrates all containers for a unified environment

**6.2 Testing Strategy**

* Current Status: Not implemented

**7. Deployment Plan**

**7**.**1 Deployment Process**

* Backend: Spring Boot JAR deployed inside Docker
* Frontend: Angular built & served via Nginx inside Docker
* Database: MySQL Docker container with persistent volumes
* Cache: Redis container
* Monitoring: Prometheus + Grafana with Docker Compose
* Deployment Target: Local via Docker, Future plan for AWS EC2/Kubernetes

**8. Maintenance and Future Work**

**8.1 Maintenance Plan**

* Actuator health endpoints for system monitoring
* Prometheus scrapes metrics for Grafana dashboards
* Redis maintains cache freshness and reduces DB load

**8.2 Future Enhancements**

* WebSocket support for live notifications
* Full test suite
* UI/UX polish
* Real-time performance graphs

**9. Code Structure**

**9.1 Directory Layout**

* **Frontend (Angular)**
  + /src/app/components – Angular components
  + /src/app/services – Angular services
  + /src/app/models – Angular Models
* **Backend (Spring Boot)**
  + /controller – API controllers
  + /service – Business logic
  + /repository – JPA repositories
  + /entity – JPA entities
  + /utility – Utility classes
  + /security – Security configuration
  + /model – DTOs and response models

**9.2 Code Organization**

* Angular for client-side UI
* Spring Boot for backend services
* RESTful architecture with clean separation of concerns

**9.3 Code Files**

* **Main Files**
  + main.ts (Angular): Application entry point
  + UplLearningPlatformApplication.java: Spring Boot entry point
* **Config Files**
  + angular.json, package.json: Angular configs
  + application.properties: Spring Boot configurations

**10. Code Run Instructions**

**10.1 Prerequisites**

* Angular CLI, Java 21, Maven, MySQL, Redis, Docker

**10.2 Setup Instructions  
Frontend Setup**

cd E:\Angular\UPLTutorial

npm install

ng serve

**# Visit** [**http://localhost:4200**](http://localhost:4200)

**Backend Setup**

cd E:\UPL\_Workspace\upl-learning-platform

mvn clean install

java -jar target/upl-learning-platform-0.0.1-SNAPSHOT.jar

**# APIs available at** [**http://localhost:8081**](http://localhost:8081)

**Dockerized Full Stack Setup**

cd E:\SpringBoot Tutorial App

docker-compose up --build

**10.3 Running Tests**

* Frontend: Not implemented yet
* Backend: Not implemented yet

**10.4 Build Instructions**

* **Frontend:** ng build --prod
* **Backend:** mvn clean install

# Output: target/ upl-learning-platform-0.0.1-SNAPSHOT.jar

**10.5 Deployment Instructions**

* Frontend:
* Backend: Deploy Spring Boot JAR
* All-in-One: Use Docker Compose for unified deployment