IIT Bombay Internship Assignment

**Courses API - Design and Implementation Documentation**

|  |  |
| --- | --- |
| Sr.no | Table of Contents |
|  | [Project Overview](#_Project_Overview) |
|  | [Architecture](#_Architecture) |
|  | [Design Decisions and Justifications](#_Design_Decisions_and) |
|  | [Frontend Implementation](#_Frontend_Implementation) |
|  | [Backend Implementation](#_Backend_Implementation) |
|  | [Database Design](#_Database_Design) |
|  | [API Design](#_API_Design) |
|  | [Deployment Strategy](#_Deployment_Strategy) |
|  | [Conclusion](#_Conclusion) |
|  | [Useful Links](#_Useful_Links) |

## Project Overview

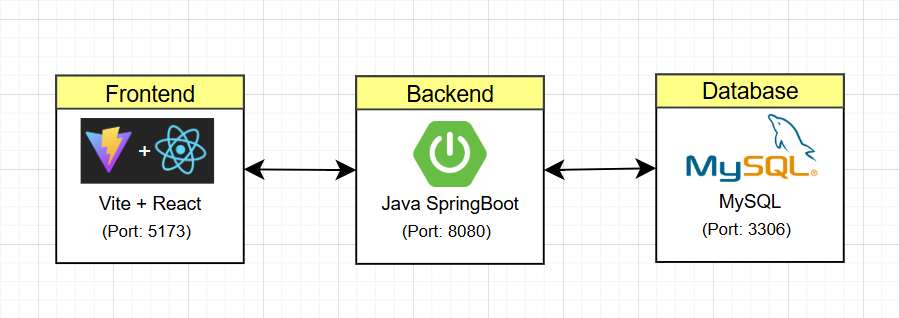
The Courses API system is a full-stack web application designed to manage course offerings and their delivery instances. The system provides comprehensive CRUD operations for courses, handles prerequisite relationships, and manages course delivery instances across different years and semesters.

**Key Features**

* Course management with prerequisite handling
* Course delivery instance management
* RESTful API with proper HTTP status codes
* User-friendly web interface
* Docker containerization

## Architecture

#### System Architecture

Fig.1. System Architecture

#### Technology Stack

1. **Backend**
   * Java
   * Spring Boot
   * Spring Data JPA
   * Spring Web
   * Maven for build management
   * MySQL for database
2. **Frontend**
   * React Vite
   * Tailwind CSS for styling the web page
3. **DevOps**
   * Docker for containerization
   * GitHub Actions for CI/CD pipeline

## Design Decisions and Justification

#### Spring Boot for Backend

**Justification:**

* Mature ecosystem with extensive documentation
* Built-in features for REST API development Excellent Spring Data JPA integration
* Robust security and validation framework
* Easy testing with Spring Boot Test

#### React for Frontend

**Justification:**

* Component-based architecture for reusability
* Large ecosystem and community support
* Excellent developer experience with modern tooling
* Easy state management and API integration

#### MySQL for Database

**Justification:**

* ACID compliance ensuring data integrity and consistency
* Mature and stable database
* Excellent performance for both read and write operations
* Strong support for complex relationships with foreign key constraints
* Easy integration with Spring Boot through Spring Data JPA
* Reliable replication and backup capabilities

## Frontend Implementation

#### Project Folder Structure

src/

|-- components/

| |-- Course.jsx

| |-- Instance.jsx

|-- pages/

| |-- CourseApp.jsx

|-- App.jsx

|-- main.jsx

#### Key Components

1. **Course** **Management**

* **CourseForm**: Multiple selection for prerequisites
* **CourseList**: Displays courses with prerequisites
* **CourseDetails**: Shows detailed course information

1. **Instance** **Management**

* **InstanceForm**: Creates course delivery instances with dropdown selection of a course
* **InstanceList**: Filters by year and semester
* **InstanceDetails**: Shows instance-specific information

#### State Management

* React hooks (useState, useEffect) for local state
* Context API for global state management
* Custom hooks for API integration

## Backend Implementation

#### Project Folder structure

src/

|-- main/

| |-- java/com/iitbombay/courses-api/

| |-- controller

| |-- CourseController.java

| |-- dto

| |-- CourseDto.java

| |-- CourseInstanceDto.java

| |-- entity

| |-- Course.java

| |-- CourseInstance.java

| |-- exception

| |-- CourseDependencyException.java

| |-- CourseInstanceNotFoundException.java

| |-- GloabalExceptionHandler.java

| |-- InvalidPrerequisiteException.java

| |-- repository

| |-- CourseInstanceRepository.java

| |-- CourseRepository.java

| |-- service

| |-- CourseService.java

| |-- CoursesApiApplicationTests.java

| |-- resources/

| |-- application.properties

|-- test/

|-- java/com/iitbombay/courses-api/

|-- CoursesApiApplicationTests.java

|-- resources/

|-- application-test.properties

#### Key Components

1. **Course Entity**
2. **Course Instance Entity**

#### Service Layer Design

1. **Course Service**

* Validates prerequisite existence before course creation
* Checks dependency constraints before deletion
* Handles prerequisite relationship management

1. **Instance Service**

* Manages course delivery instances
* Validates course existence before instance creation
* Handles year and semester validation

#### Control Layer

1. **200 OK** - for successful operations
2. **201 Created** - for resource creation
3. **400 Bad Request** - for validation errors
4. **204 No Content** – for not finding any records
5. **500 Internal Server Error** – for server issue or network problems

## Database Design

#### Entity Relationship Diagram

Fig.2. ER Diagram of Database

#### Tables

1. **courses**
   * + id (Primary Key)
     + title (VARCHAR, NOT NULL)
     + course\_id (VARCHAR, UNIQUE, NOT NULL)
     + description (TEXT)
     + course\_prerequisites
2. **course\_instance**

* id (Primary Key)
* course\_id (Foreign Key to courses)
* year (INTEGER, NOT NULL)
* semester (INTEGER, NOT NULL)

1. **course\_prerequisites**

* course\_id (Foreign Key to courses)
* prerequisite\_id (Foreign Key to courses)

#### Database Constrains

* Unique constraint on course\_id
* Check constraint on semester (1-2)
* Foreign key constraints on respective fields
* Composite unique constraint on (course\_id, year, semester) for instances

## API Design

#### Course Endpoints

1. **POST /api/courses -** Create a new course
2. **GET /api/courses -** Retrieve all courses with array of courses including prerequisites
3. **GET /api/courses/{id} -** Retrieve specific course with course object
4. **DELETE /api/courses/{id} -** Delete a course or if dependency exist(is prerequisite for other course or having instance for specific year and semester) doesn’t allow to delete that course

#### Instance Endpoints

* 1. **POST /api/instances –** create new instance in particular year of semester 1 or 2
  2. **GET /api/instances/{year}/{semester} -** List course instances for specific year/semester
  3. **GET /api/instances/{year}/{semester}/{courseId} -** Get specific course instance details
  4. **DELETE /api/instances/{year}/{semester}/{courseId} -** Delete specific course instance

## Deployment Strategy

#### Docker Configuration

1. **Backend Dockerfile**

FROM openjdk:17-jdk-slim as build

WORKDIR /app

COPY mvnw .

COPY .mvn .mvn

COPY pom.xml .

RUN chmod +x ./mvnw

RUN ./mvnw dependency:resolve

COPY src src

RUN ./mvnw clean package -DskipTests

FROM openjdk:17-jdk-slim

RUN addgroup --system spring && adduser --system spring --ingroup spring

WORKDIR /app

COPY --from=build /app/target/\*.jar app.jar

RUN chown spring:spring app.jar

USER spring

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]

1. **Frontend Dockerfile**

FROM node:20-alpine as build

WORKDIR /app

COPY package\*.json ./

RUN npm ci --only=production

COPY . .

RUN npm run build

FROM nginx:alpine

COPY --from=build /app/dist /usr/share/nginx/html

EXPOSE 5173

CMD ["nginx", "-g", "daemon off;"]

1. **Docker Compose**

services:

  # MySQL Database

  mysql:

    image: mysql:8.0

    container\_name: mysql-db

    restart: unless-stopped

    environment:

      MYSQL\_ROOT\_PASSWORD: Sakshi@2004

      MYSQL\_DATABASE: courses\_db

    ports:

      - "3306:3306"

    volumes:

      - mysql\_data:/var/lib/mysql

    networks:

      - app-network

    healthcheck:

      test: ["CMD", "mysqladmin", "ping", "-h", "localhost", "-u", "root", "-pSakshi@2004"]

      timeout: 10s

      retries: 10

  # Backend Service

  backend:

    image: saksy20/backend-app:latest

    container\_name: spring-backend

    restart: unless-stopped

    environment:

      SPRING\_DATASOURCE\_URL: jdbc:mysql://mysql:3306/courses\_db

      SPRING\_DATASOURCE\_USERNAME: root

      SPRING\_DATASOURCE\_PASSWORD: Sakshi@2004

      SPRING\_JPA\_HIBERNATE\_DDL\_AUTO: update

    ports:

      - "8080:8080"

    depends\_on:

      mysql:

        condition: service\_healthy

    networks:

      - app-network

  # Frontend Service

  frontend:

    image: saksy20/frontend-app:latest

    container\_name: react-frontend

    restart: unless-stopped

    ports:

      - "5173:80"

    depends\_on:

      - backend

    networks:

      - app-network

volumes:

  mysql\_data:

networks:

  app-network:

    driver: bridge

#### CI/CD Pipeline

GitHub Actions workflow for automated deployment:

* Code checkout
* Run tests
* Build Docker images
* Push to DockerHub
* Deploy to staging/production

## Conclusion

The Courses API is a well-designed full-stack app that follows best practices for clean structure, validation, and user experience. It handles course prerequisites, errors, and user input effectively. Docker and CI/CD make deployment smooth and consistent across environments.

## Useful Links

* [Frontend Github Repository Link](https://github.com/saksy25/frontendApp-iitb)
* [Backend Github Repository Link](https://github.com/saksy25/backendApp-iitb)
* [Linkedin Profile Link](http://www.linkedin.com/in/sakshi-salunke-758014273)