KampusLearn

# Problem statement

KampusLearn is a global online institution to learn in-demand technology, design and management skills from anywhere, anytime. The application is committed to transforming millions of lives by providing the best training, online resources, and guidance globally.

The following section will cover aspects related to ShowFlix

* Register Courses.
* Register Trainers.
* Schedule Trainings.
* Fee Collection.
* Register Candidates

**Scope of the System**

The scope of the system is explained through its modules as follows

Register Courses : The system allows admins to register the course so the candidates can view the courses

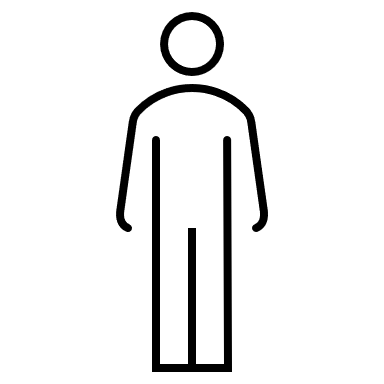
Register Trainers : The system allows admins add the Trainers details so trainers can be mapped for the trainings

TrainingSchedule -- Create training schedule as requested by candidates.

Register Candidates—Save the candidate details in the system.

Fee Collection --- Add a Fee collection module to the application.

**Use Case Diagram**



**Flow Diagram**

# Project Development Guidelines

The project to be developed based on the below design considerations

|  |  |
| --- | --- |
| **Backend Development** | * Use Rest APIs (Springboot/ASP.Net Core WebAPI to develop the services * Use Java/C# latest features * Use ORM with database * Use Swagger to invoke APIs * Use JWT for authentication in SpringBoot/WebApi. A Token must be generated using JWT. Tokens must expire after a definite time interval, and authorization must be handled accordingly based on token expiry * Implement Logging * Implement API Versioning * Implement security to allow/disallow CRUD operations * Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant. * Any error message or exception should be logged and should be user-readable (not technical) * Database connections and web service URLs should be configurable * Implement Unit Test Project for testing the API * Follow Coding Standards |
| **Frontend Development** | * Use Angular/React to develop the UI * Implement Forms, databinding, validations * Implement Routing and navigations * Use JavaScript to enhance functionalities * Implement External and Custom JavaScript files * Implement Typescript for Functions, Operators. * Any error message or exception should be logged and should be user-readable (and not technical) * Follow coding standards * Follow Standard project structure |

# Good to have implementation features

* Generate a SonarQube report and fix the required vulnerability
* Use the Moq framework as applicable
* Create a Docker image for the frontend and backend of the application
* Implement design patterns
* Deploy the docker image in AWS EC2 or Azure VM
* Build the application using the AWS/Azure CI/CD pipeline. Trigger a CI/CD pipeline when code is checked-in to GIT. The check-in process should trigger unit tests with mocked dependencies
* Use AWS RDS or Azure SQL DB to store the data