System Requirements Specification Index

For

College Management Application

Version 1.0

IIHT Pvt. Ltd.
IIHT Ltd, No: 15, 2nd Floor, Sri Lakshmi Complex, Off MG Road, Near SBI LHO,
Bangalore, Karnataka – 560001, India
fullstack@iiht.com

Table of Contents

College Management SYSTEM
System Requirements Specification

1 Project Abstract

College Management Application is a simple .Net Core 3.1 RESTful Web API application with MS Sql server, where it managing students, departments and teachers
Every student details should have the properties like student id, name, department.
Every teacher details should have the properties like teacher id, name, department.

Following is the requirement specification:

	College Management Application
1	Department
2	Teacher
3	Student
Department Module Functionalities	
	1.Create a department
	2.Can delete a department
	3.Get department Info by department name
	4.Fetch all departments
	5.Get department info by department id
Student Module Functionalities	
	1.Create a student
	2.Can delete a student
	3.Get student Info by student name
	4.Fetch all students
	5.Get student info by student id
Teacher Module Functionalities	
	1.Create a teacher
	2.Can delete a teacher
	3.Get teacher Info by teacher name

	4.Fetch all teachers
	5.Get teacher info by teacher id

2. Assumptions, Dependencies, Risks / Constraints

2.1 Student Constraints

- While adding a student details, if student is already existing, it should throw a custom exception
- While deleting the student, ensure that student already exists, if not, the operation should throw a custom exception

2.2 Teacher Constraints

- The teacher and student Details are connected through a field department name – applying integrity constraint

2.3 Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exception if data is invalid
- All the database operations must be implemented on entity object only
- Do not change, add, remove any existing methods in service layer
- In Repository interfaces, custom methods can be added as per requirements.
- All RestEndpoint methods and Exception Handlers must return data wrapped in ResponseEntity

- **All business logic CRUD operations under repository class and write your business logic validation in Services class and related validation use proper user defined exceptions mentioned in above document.**
- **Controller must validate before processing any logic on the database.**

2.4 Visitors can perform the follow actions

- Allows to add a new department/student/teacher details
- Allows to delete an existing department/student/teacher
- Allows to search the department/student/teacher on the basis of name
- Allows to display all department/student/teacher

2.5 ToolChain

- .NET Core 3.1, RESTful Web API, MS SQL Server.

3. Business Validations

3.1 Department Class Entities

- Department Id (long) must be not null and unique, key attribute
- Department Name (string) is not null, min 3 and max 100 characters.

3.2 Student Class Entities

- Student Id (long) is not null, key attribute
- Student Name (String) is not null, min 3 and max 100 characters.
- Department Name (String) is not null, min 3 and max 100 characters.

3.2 Teacher Class Entities

- Teacher Id (long) is not null, key attribute
- Teacher Name (String) is not null, min 3 and max 100 characters.
- Department Name (String) is not null, min 3 and max 100 characters.

4. Considerations

- For Role of application users 3 possible values must be used: -
  1. Department
  2. Student
  3. Teacher

5. Rest Endpoints

Rest End-points to be exposed in the controller along with method details for the same to be created

5.1 DepartmentController

URL Exposed		Purpose
/api/Department/Create-Department		Add new department Details
Http Method	Post	

	Parameter 1	Department department	
	Return	HttpResponse status code	
/api/Department/Update-Department			Update existing department Details
	Http Method	PUT	
	Parameter 1	Department department	
	Return	HttpResponse status code	
/api/Department/Delete-Department			Delete department with given department id.
	Http Method	DELETE	
	Parameter 1	Long departmentId	
	Return	HttpResponse status code	
/api/Department/Get-Department-By-Id			Fetches the department Details with the given department id
	Http Method	GET	
	Parameter 1	Long departmentId	
	Return	<Department>	
/api/Department/Get-All-Departments			Fetches all the department Details
	Http Method	GET	
	Parameter 1	-	
	Return	<<IEnumerable<Department>>>	
/api/Department/Search-Department-By-Name			Fetches the department Details with the given department name
	Http Method	GET	
	Parameter 1	-	
	Return	<Department>	

5.2 StudentController

URL Exposed			Purpose
/api/Student/Create-Student			Add new student Details
	Http Method	Post	
	Parameter 1	Student student	
	Return	HttpResponse status code	

/api/Student/Delete-Student		Delete student with given department id.
Http Method	DELETE	
Parameter 1	Long studentId	
Return	HttpResponse status code	
/api/Student/Get-Student-By-Id		Fetches the student Details with the given student id
Http Method	GET	
Parameter 1	Long studentId	
Return	<Student>	
/api/Student/Get-All-Students		Fetches all the student Details
Http Method	GET	
Parameter 1	-	
Return	<<IEnumerable<Student> >>	
/api/Student/Search-Student-By-Name		Fetches the student Details with the given student name
Http Method	GET	
Parameter 1	-	
Return	<Student>	

5.3 TeacherController

URL Exposed		Purpose
/api/Teacher/Create-Teacher		Add new teacher Details
Http Method	Post	
Parameter 1	Teacher teacher	
Return	HttpResponse status code	
/api/Teacher/Delete-Teacher		Delete teacher with given teacher id.
Http Method	DELETE	
Parameter 1	Long teacherId	
Return	HttpResponse status code	
/api/Teacher/Get-Teacher-By-Id		Fetches the teacher Details with the given teacher id
Http Method	GET	
Parameter 1	Long teacherId	
Return	<Teacher>	
/api/Teacher/Get-All-Teachers		Fetches all the teacher Details

Http Method	GET
Parameter 1	-
Return	`<<IEnumerable<Teacher> >>`

/api/Teacher/Search-Teacher-By-Name

Http Method	GET
Parameter 1	-
Return	`<Teacher>`

Fetches the teacher Details with the given teacher name

6. Template Code Structure

6.1 Package: CollegeManagement

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	Department Controller Student Controller Teacher Controller	Controller class to expose all rest-endpoints for auction related activities.	Partially implemented
Startup.cs	Startup CS file	Contain all Services settings and SQL server Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented
	appsettings.json	Contain connection string for database	Already Implemented

6.2 Package: CollegeManagement.BusinessLayer

Resources

Names	Resource	Remarks	Status
Package Structure			

Interface	ITeacherService, IStudentService, IDepartmentService interface	Inside all these interface files contains all business validation logic functions.	Already Implemented
Service	TeacherService CS file StudentService CS file DepartmentService CS file	Using this all class we are calling the Repository method and use it in the program and on the controller.	Partially Implemented
Repository	ITeacherRepository TeacherRepository IStudentRepository StudentRepository IDepartmentRepository DepartmentRepository CS file and interface.	All these interfaces and class files contain all CRUD operation code for the database. Need to provide implementation for service related functionalities	Partially Implemented
ViewModels	TeacherViewModel, StudentViewModel, DepartmentViewModel	Contain all view Domain entities for show and bind data. All the business validations must be implemented.	Already Implemented

6.3 Package: CollegeManagement.DataLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
DataLayer	CollegeDbContext cs file	All database Connection and collection setting class	Already Implemented

6.4 Package: CollegeManagement.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Entities	Teacher, Student, Department CS file	All Entities/Domain attribute are used for pass the data in controller. Annotate this class with proper annotation to declare	

		it as an entity class with **Id** as primary key. Generate the **Id** using the **IDENTITY** strategy	Already Implemented

6.5 Package: CollegeManagement.Tests

Resources

The CollegeManagement.Tests project contains all test case classes and functions for code evaluation. Don’t edit or change anything inside this project.

FRONTEND-ANGULAR SPA

1 PROBLEM STATEMENT

College management is SPA (Single Page Application) for maintaining all information related to students, teachers and departments in college. It performs all CRUD operations along with searching functionality for all 3 modules.

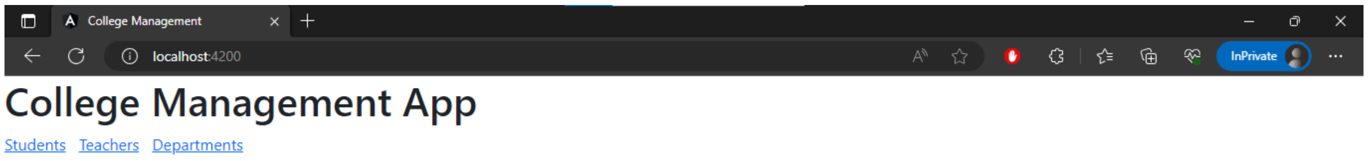
The core modules of College management app are:

1. Home Page
2. Students Page
3. Teachers Page
4. Departments Page

2 PROPOSED COLLEGE MANAGEMENT WIREFRAME

UI needs improvisation and modification as per given use case and to make test cases passed.

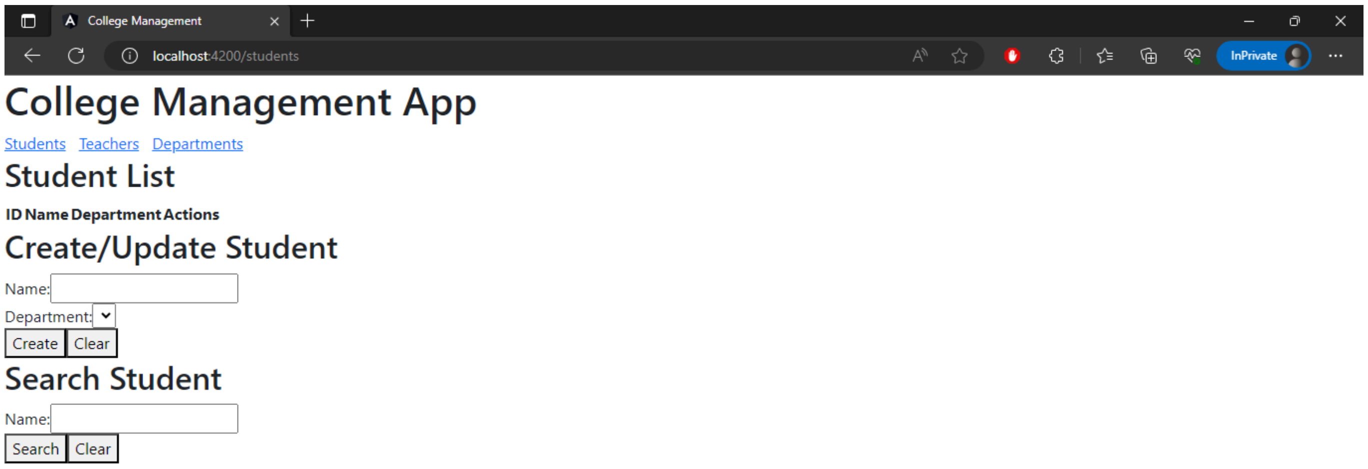
2.1 HOME PAGE



College Management App

Students Teachers Departments

2.2 STUDENTS PAGE



College Management App

Students Teachers Departments

Student List

ID Name Department Actions

Create/Update Student

Name:

Department:

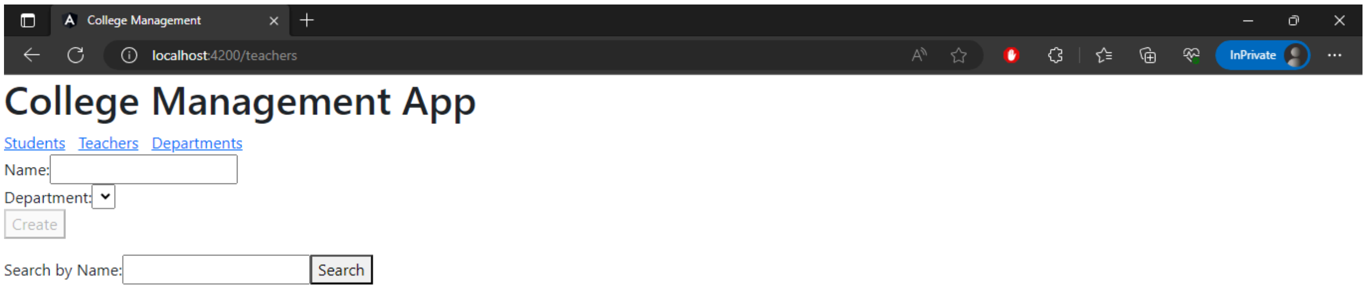
Create Clear

Search Student

Name:

Search Clear

2.3 Teachers Page



College Management App

Students Teachers Departments

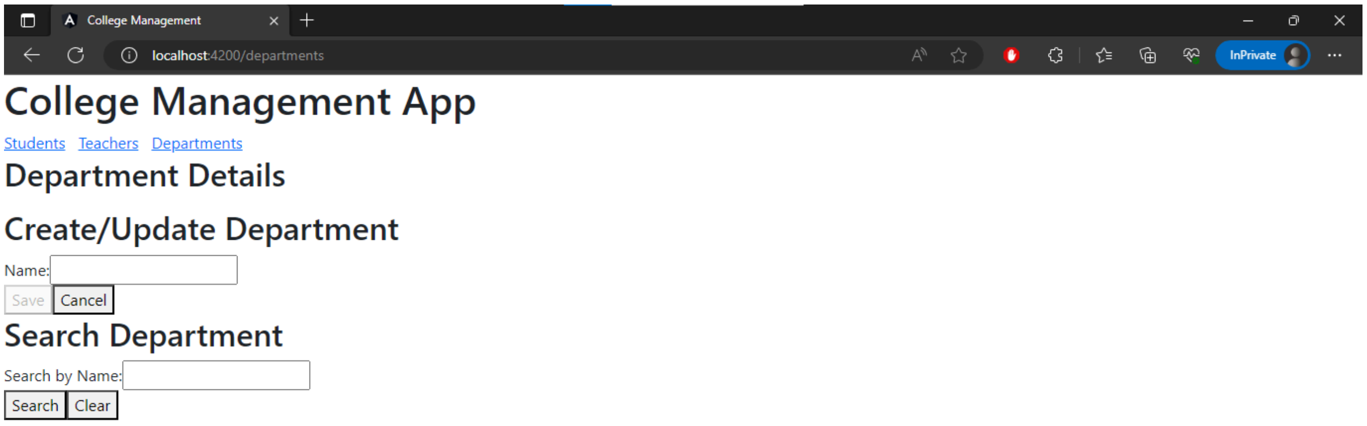
Name:

Department:

Create

Search by Name: Search

2.4 Departments Page



College Management App

Students Teachers Departments

Department Details

Create/Update Department

Name:

Save Cancel

Search Department

Search by Name:

Search Clear

3 BUSINESS-REQUIREMENT:

As an application developer, develop the College management App (Single Page App) with below guidelines:

User Story #	User Story Name	User Story
US_01	Home Page	As a user I should be able to visit the welcome page as default page. Acceptance criteria: 1. User can click any links shown at homepage.
US_02	Students Page	As a user I should be able to see Students page and perform all CRUD operations: Acceptance criteria: 1. As a user I should be able to furnish the following details at the time of creating an ngo. 1.1 Name 1.2 Department Name 2. Create button should be disabled until all fields are validated. 3. Cancel button should clear all fields. 4. Search button should search all students on a name basis and update the list accordingly.

US_03	Teachers Page	As a user I should be able to see teachers page and perform all CRUD operations: Acceptance criteria: 1. As a user I should be able to furnish the following details at the time of creating a teacher. 1.2 Name 1.3 Department Name 2 Create button should be disabled until all fields are validated. 3 Search button should search all teachers on a name basis and update the list accordingly.

US_04	Departments Page	As a user I should be able to see department page and perform all CRUD operations: Acceptance criteria: 1. As a user I should be able to furnish the following details at the time of creating a department. 1.2 Name Save button should be disabled until all fields are validated. 2. Search button should search all departments on a name basis and update the list accordingly.

4 CONSTRAINTS

1. On the page load, input focus must come to the first name input field.
2. You should be able to press the “TAB” key and “SHIFT + TAB” to navigate from top field to bottom field and vice-versa.

7. Execution Steps to Follow

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**

2. **To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) Terminal → New Terminal.**

3. **On command prompt, cd into your project folder (cd <Your-Project-folder>).**

4. **To connect SQL server from terminal:**
   (CollegeManagement /sqlcmd -S localhost -U sa -P pass@word1)
   - **To create database from terminal -**

   ```
   1>  Create Database CollegeManagementDb
   2>  Go
   ```

5. **Steps to Apply Migration(Code first approach):**
   - **Press Ctrl+C to get back to command prompt**
   - **Run following command to apply migration-**
     (CollegeManagement /dotnet-ef database update)

6. **To check whether migrations are applied from terminal:**
   (CollegeManagement /sqlcmd -S localhost -U sa -P pass@word1)

   ```
   1> Use CollegeManagementDb
   2> Go
   1> Select * From __EFMigrationsHistory
   2> Go
   ```

7. **To build your project use command:**
   (CollegeManagement /dotnet build)

8. **To launch your application, Run the following command to run the application:**
   (CollegeManagement/dotnet run)

9. **This editor Auto Saves the code.**

10. **To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.**

11. **To test web-based applications on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**

    Note: The application will not run in the local browser

12. **To run the test cases in CMD, Run the following command to test the application: (CollegeManagement.Tests/dotnet test --logger "console;verbosity=detailed")**
    (You can run this command multiple times to identify the test case status,and refactor code to make maximum test cases passed before final submission)

13. **If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B - command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.**

14. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**

15. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**

8. Execution Steps to Follow For Frontend

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers, need to go to**

   Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.

3. **This is a web-based application, to run the application on a browser, use the internal browser in the environment.**
4. **Follow the steps below to install and use Node.js version 18.20.3 using nvm:**
   a. **Install nvm:**
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash
   b. **Set up nvm environment:**
   export NVM_DIR="$([-z "${XDG_CONFIG_HOME-}"] && printf %s "${HOME}/.nvm" || printf %s "${XDG_CONFIG_HOME}/nvm")" && [-s "$NVM_DIR/nvm.sh"] && \. "$NVM_DIR/nvm.sh"

   c. **Verify nvm Installation:**

   command -v nvm

   d. **Install Node.js Version 18.20.3:**

   nvm install 18.20.3

   e. **Set the installed Node.js version as active:**

   nvm use 18.20.3

5. **You can follow series of command to setup Angular environment once you are in your project-name folder:**
   a. **npm install -> Will install all dependencies -> takes 10 to 15 min**

   b. **npm run start -> To compile and deploy the project in browser. You can press <Ctrl> key while clicking on localhost:4200 to open project in browser -> takes 2 to 3 min**

   c. **npm run test -> to run all test cases. It is mandatory to run this command**

before submission of workspace -> takes 5 to 6 min

6. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**