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

College Management SYSTEM

System Requirements Specification

1.Business-Requirement:

1.1 Problem Statement:

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.

1.2 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 2 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 Http Method: Post Parameter 1: Department department Return: HttpResponse status code	Add new department Details
/api/Department/Update-Department Http Method: PUT Parameter 1: Department department Return: HttpResponse status code	Update existing department Details
/api/Department/Delete-Department Http Method: DELETE Parameter 1: Long departmentId Return: HttpResponse status code	Delete department with given department id.
/api/Department/Get-Department-By-Id Http Method: GET Parameter 1: Long departmentId Return: <Department>	Fetches the department Details with the given department id
/api/Department/Get-All-Departments Http Method: GET Parameter 1: - Return: <<IEnumerable<Department>>>	Fetches all the department Details
/api/Department/Search-Department-By-Name Http Method: GET Parameter 1: - Return: <Department>	Fetches the department Details with the given department name

5.2 StudentController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>/api/Student/Create-Student<table><tr><td>Http Method</td><td>Post</td></tr><tr><td>Parameter 1</td><td>Student student</td></tr><tr><td>Return</td><td>HttpResponse status code</td></tr></table></td><td>Add new student Details</td></tr>
<tr><td>/api/Student/Delete-Student<table><tr><td>Http Method</td><td>DELETE</td></tr><tr><td>Parameter 1</td><td>Long studentId</td></tr><tr><td>Return</td><td>HttpResponse status code</td></tr></table></td><td>Delete student with given department id.</td></tr>
<tr><td>/api/Student/Get-Student-By-Id<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long studentId</td></tr><tr><td>Return</td><td><Student></td></tr></table></td><td>Fetches the student Details with the given student id</td></tr>
<tr><td>/api/Student/Get-All-Students<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>-</td></tr><tr><td>Return</td><td><<IEnumerable<Student>></td></tr></table></td><td>Fetches all the student Details</td></tr>
<tr><td>/api/Student/Search-Student-By-Name<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>-</td></tr><tr><td>Return</td><td><Student></td></tr></table></td><td>Fetches the student Details with the given student name</td></tr>
</table>

5.2 TeacherController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>/api/Teacher/Create-Teacher<table><tr><td>Http Method</td><td>Post</td></tr><tr><td>Parameter 1</td><td>Teacher teacher</td></tr><tr><td>Return</td><td>HttpResponse status code</td></tr></table></td><td>Add new teacher Details</td></tr>
<tr><td>/api/Teacher/Delete-Teacher<table><tr><td>Http Method</td><td>DELETE</td></tr><tr><td>Parameter 1</td><td>Long teacherId</td></tr><tr><td>Return</td><td>HttpResponse status code</td></tr></table></td><td>Delete teacher with given teacher id.</td></tr>
</table>

/api/Teacher/Get-Teacher-By-Id Http Method: GET Parameter 1: Long teacherId Return: <Teacher>	Fetches the teacher Details with the given teacher id
/api/Teacher/Get-All-Teachers Http Method: GET Parameter 1: - Return: <<IEnumerable<Teacher>>>	Fetches all the teacher Details
/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.

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.**
