# System Requirements Specification Index

For

# REST API for Blog Application

Version 4.0

**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

# REST API for Blog APPLICATION

## System Requirements Specification

## 1. Business-Requirement:

### 1.1 Problem Statement:

**Blog Application** is Asp.net Core 3.1 RESTful Web API with MS SQL Server, where it allows users to manage the blogs.

### 1.2 Following is the requirement specification:

| | Blog Application |
|---|---|
| | |
| Modules | |
| 1 | Blogs |
| | |
| Blog Module Functionalities | |
| | |
| 1 | Create a Blog |
| 2 | Update a Blog |
| 3 | Delete a Blog |
| 4 | Get the Blog by Id |
| 5 | Get all Blogs |

## 2. Assumptions, Dependencies, Risks / Constraints

### 2.1 Blog Constraints:

- While fetching the Blog by Id, if Id does not exist then the operation should throw a custom exception.
- While Updating the Blog by Id, if Id does not exist then operation should throw custom exception
- While deleting the Blog by Id, if Id does not exist then operation should throw custom exception

### 2.2 Common Constraints

- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exception if data is invalid
- All the business validations must be implemented in model classes only.
- 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**

# 3. Business Validations

## 3.1 Blog Class Specifications:

- Blog Id(int) is not null,Key attribute.
- Blog title(string) is not null, min 3 and max 100 characters.
- Blog content(string) is not null, min 3, max 200 characters.

# 4. Rest Endpoints

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

## 4.1 BlogController

| URL Exposed | Purpose |
|---|---|
| /blogservice/all<br>Http Method: GET<br>Parameter 1: -<br>Return: <IEnumerable<Blog>> | Fetches all the blogs |
| /blogservice/add<br>Http Method: POST<br>Parameter 1: BlogViewModel<br>Return: HTTP Status Code | Add a new blog |
| /blogservice/delete/{blogId}<br>Http Method: DELETE<br>Parameter 1: Integer (blogId)<br>Return: HTTP Status Code | Delete blog with given blog id |
| /blogservice/get/{blogId}<br>Http Method: GET<br>Parameter 1: Integer (blogId)<br>Return: <Blog> | Fetches the blog with the given id |
| /blogservice/update<br>Http Method: PUT<br>Parameter 1: BlogViewModel<br>Return: HTTP Status Code | Updates blog details |

# 5. Template Code Structure

## 5.1 Package: BlogsApplication

**Resources**

| Names | Resource | Remarks | Status |
|---|---|---|---|
| Package Structure | | | |
| controller | BlogController | 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 file | Contain connection string for database | Already Implemented |

## 5.2 Package: BlogsApplication.BusinessLayer

**Resources**

| Names | Resource | Remarks | Status |
|---|---|---|---|
| Package Structure | | | |
| Interfaces | IBlogService interface | Inside all these interface files contains all business validation logic functions. | Already Implemented |
| Services | Blog Service 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 | IBlog Repository<br>Blog Repository<br>CS file and interface. | All these interfaces and class files contain all CRUD operation code for the database.<br>Need to provide implementation for service related functionalities | Partially Implemented |
| ViewModels | BlogViewModel, | Contain all view Domain entities for show and bind data.<br>All the business validations must be implemented. | Partially Implemented |

### 5.3 Package: BlogsApplication.dataLayer

**Resources**

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

### 5.4 Package: BlogsApplication.Entities

**Resources**

| Names | Resource | Remarks | Status |
|---|---|---|---|
| Package Structure | | | |
| Entities | Blog,Response | All Entities/Domain attribute are used for pass the data in controller<br><br>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 | Partially Implemented |

### 5.5 Package: BlogsApplication.Tests

**Resources**

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

## 6. 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:
   (BlogsApplication/sqlcmd -S localhost -U sa -P pass@word1)
   - To create database from terminal -
     1> Create Database BlogsApp_Db
     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-
     (BlogsApplication/dotnet-ef database update)

6. To check whether migrations are applied from terminal:
   (BlogsApplication/sqlcmd -S localhost -U sa -P pass@word1)
   1> Use BlogsApp_Db
   2> Go
   1> Select * From __EFMigrationsHistory
   2> Go

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

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