System Requirements Specification Index

For

REST API for Blog Application

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

Table of Contents

REST API for Blog APPLICATION

System Requirements Specification

1 Project Abstract

Blog Application is Spring boot RESTful application with MySQL, where it allows users to manage the blogs.

Following is the requirement specifications:

	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 by creating a class ResourceNotFoundException with message “Blog not found”.
- While Updating the Blog by Id, if Id does not exist then the operation should throw a custom exception by creating a class ResourceNotFoundException with message “Blog not found”.
- While deleting the Blog by Id, if Id does not exist then the operation should throw a custom exception by creating a class ResourceNotFoundException with message “Blog not found”.

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 both DTO and entity classes.
- 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

- Blog title should not be null and min 3 and max 100 characters.
- Blog content should not be null and 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

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/blogs<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>-</td></tr><tr><td>Return</td><td>BlogEntity</td></tr></table></td><td>Create a new blog</td></tr>
<tr><td>/api/blogs/{id}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long(id)</td></tr><tr><td>Return</td><td>BlogEntity</td></tr></table></td><td>Get an blog by id</td></tr>
<tr><td>/api/blogs<table><tr><td>Http Method</td><td>PUT</td></tr><tr><td>Parameter 1</td><td>BlogEntity</td></tr><tr><td>Return</td><td>BlogEntity</td></tr></table></td><td>Updates existing blog</td></tr>
<tr><td>/api/blogs/{id}<table><tr><td>Http Method</td><td>DELETE</td></tr><tr><td>Parameter 1</td><td>Long(Id)</td></tr><tr><td>Return</td><td>Boolean</td></tr></table></td><td>Deletes a blog by id</td></tr>
<tr><td>/api/blogs<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>-</td></tr><tr><td>Return</td><td>List<BlogEntity></td></tr></table></td><td>Fetches all the blogs</td></tr>
</table>

5 Template Code Structure

5.1 Package: com.yaksha.assessments.blogs

Resources

SpringbootBlogsServiceApplication (Class)	This is the Spring Boot starter class of the application.	Already Implemented. **You are free to add any bean in this class.**

5.2 Package: com.yaksha.assessments.blogs.entity

Resources

Class/Interface	Description	Status
BlogEntity (class)	• Annotate this class with proper annotation to declare it as an entity. class with **id** as primary key. • Map this class with a **blogs** table. • Generate the **id** using **IDENTITY** strategy	Partially implemented.

5.3 Package: com.yaksha.assessments.blogs.dto

Resources

Class/Interface	Description	Status
BlogDto (class)	Use appropriate annotations for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.

5.4 Package: com.yaksha.assessments.blogs.repository

Resources

Class/Interface	Description	Status
BlogRepository (interface)	1. Repository interface exposing CRUD functionality for **Blog** Entity.	Already implemented

	2. You can go ahead and add any custom methods as per requirements	

5.5 PACKAGE: COM.YAKSHA.ASSESSMENTS.BLOGS.SERVICE

Resources

Class/Interface	Description	Status
BlogService (interface)	Interface to expose method signatures for Blog related functionality. Do not modify, add or delete any method	Already implemented.

5.6 PACKAGE: COM.YAKSHA.ASSESSMENTS.BLOGS.SERVICE.IMPL

Resources

Class/Interface	Description	Status
BolgServiceImpl (class)	• Implements BlogService. • Need to provide implementation for Blog related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	To be implemented.

5.7 Package: com.yaksha.assessments.blogs.exception

Resources

Class/Interface	Description	Status
ResourceNotFoundException (Class)	• Custom Exception to be thrown when trying to get the blog details. • Need to create Exception Handler for same wherever needed (local or global)	Need to be created.

5.8 Package: com.yaksha.assessments.blogs.controller

Resources

Class/Interface	Description	Status
BlogController (Class)	• Controller class to expose all rest-endpoints for Blogs related activities. • May also contain local exception handler methods	To be implemented

6 Method Descriptions

6.1 BlogServiceImpl Class - Method Descriptions

-> Add blogRepository as dependency.

Method	Task	Implementation Details
createBlog	To create and save a new blog post	- You should call `blogRepository.save(blogEntity)` to store the new blog in the database. - You need to return the saved `BlogEntity` object.
getBlogById	To fetch a blog post by its ID	- You should call `blogRepository.findById(id)` to find the blog. - If the blog is not found, you must return an exception `ResourceNotFoundException` with the message `"Blog not found"`. - You need to return the found `BlogEntity`.
updateBlog	To update an existing blog post	- You should call `blogRepository.findById(blogEntity.getId())` to find the existing blog. - If the blog is not found, return an exception `ResourceNotFoundException` with the message `"Blog not found"`. - You should set the updated values for title and content using setters. - Call `blogRepository.save(updatedEntity)` to save changes. - Return the updated `BlogEntity`.
deleteBlog	To delete a blog post by ID	- You should call `blogRepository.findById(id)` to locate the blog. - If not found, throw `ResourceNotFoundException` with message `"Blog not found"`. - You should delete the blog using `blogRepository.delete(blog)`. - Return `true` after successful deletion.

findAll	To retrieve all blog posts	- You should call `blogRepository.findAll()` to get all blog records. - Return the result as a `List<BlogEntity>`.

6.2 BlogController Class - Method Descriptions

-> Add blogService as dependency

Method	Task	Implementation Details
createBlog	To implement logic to handle blog creation via API	- The request type should be `POST` with URL `/api/blogs`. - The method name should be `createBlog` and it should return `ResponseEntity<BlogEntity>`. - Use `@Valid @RequestBody` to accept and validate the `BlogEntity` from the request. - This method should call `blogService.createBlog(blogEntity)`. - It should return the created blog with status `HttpStatus.CREATED`.
getBlogById	To implement logic to get a blog post by ID	- The request type should be `GET` with URL `/api/blogs/{id}`. - The method name should be `getBlogById` and it should return `ResponseEntity<BlogEntity>`. - Use `@PathVariable` to accept the blog `id` from the path. - This method should call `blogService.getBlogById(id)`. - It should return the blog with status `HttpStatus.OK`.
updateBlog	To implement logic to update a blog post	- The request type should be `PUT` with URL `/api/blogs`. - The method name should be `updateBlog` and it should return `ResponseEntity<BlogEntity>`. - Use `@RequestBody` to accept the updated blog data. - This method should call `blogService.updateBlog(blogEntity)`.

		- It should return the updated blog with status `HttpStatus.OK`.
`deleteBlogById`	To implement logic to delete a blog by ID	- The request type should be `DELETE` with URL `/api/blogs/{id}`. - The method name should be `deleteBlogById` and it should return `ResponseEntity<Boolean>`. - Use `@PathVariable` to accept the blog `id`. - This method should call `blogService.deleteBlog(id)`. - It should return `true / false` with status `HttpStatus.OK` on success.
`getAllBlogs`	To implement logic to retrieve all blog posts	- The request type should be `GET` with URL `/api/blogs`. - The method name should be `getAllBlogs` and it should return `ResponseEntity<List<BlogEntity>>`. - This method does not accept any input parameters. - It should call `blogService.findAll()`. - It should return the list of all blogs with status `HttpStatus.OK`.

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 Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal**
3. **To build your project use command:**
   mvn clean package -Dmaven.test.skip
4. **To launch your application, move into the target folder (cd target). Run the following command to run the application:**
   java -jar springboot-blogs-service-0.0.1-SNAPSHOT.jar

5.This editor Auto Saves the code

6. 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.

7. 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.
8. 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.
9. This is a web-based application, to run the application 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

10. Default credentials for MySQL:
    a. Username: root
    b. Password: pass@word1
11. To login to mysql instance: Open new terminal and use following command:
    a. sudo systemctl enable mysql
    b. sudo systemctl start mysql

    NOTE: After typing the second sql command (sudo systemctl start mysql), you may encounter a warning message like :

    > System has not been booted with systemd as init system (PID 1). Can't operate. Failed to connect to bus: Host is down

    >> Please note that this warning is expected and can be disregarded. Proceed to the next step.

    c. mysql -u root -p
       The last command will ask for password which is 'pass@word1'

12. Mandatory: Before final submission run the following command:
    mvn test

13. 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.