

---

# System Requirements Specification Index

For

## REST API for Blog Application

Version 1.0

## TABLE OF CONTENTS

1	Project Abstract	3
2	Assumptions, Dependencies, Risks / Constraints	3
2.1	Blog Constraints:	3
2.2	Common Constraints:	3
3	Business Validations	4
4	Rest Endpoints	4
4.1	BlogController	4
5	Template Code Structure	5
5.1	Package: com.yaksha.assessments.blogs	5
5.2	Package: com.yaksha.assessments.blogs.entity	5
5.3	Package: com.yaksha.assessments.blogs.dto	5
5.4	Package: com.yaksha.assessments.blogs.repository	5
5.5	Package: com.yaksha.assessments.blogs.service	6
5.6	Package: com.yaksha.assessments.blogs.service.impl	6
5.7	Package: com.yaksha.assessments.blogs.exception	7
5.8	Package: com.yaksha.assessments.blogs.controller	7
6	Dependency Requirements	7
7	Execution Steps to Follow	8

# 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 dto 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

---

- 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

URL Exposed		Purpose
1. /api/blogs		Create a new blog
Http Method	POST	
Parameter 1	-	
Return	Blog	
/api/blogs/{id}		Get an blog by id
Http Method	GET	
Parameter 1	Long(id)	
Return	Blog	
/api/blogs/{id}		Updates existing blog by id
Http Method	PUT	
Parameter 1	Long(Id)	
Parameter 2	BlogEntity	
Return	Blog	
/api/blogs/{id}		Deletes a blog by id
Http Method	DELETE	
Parameter 1	Long(Id)	
Return	-	
/api/blogs/all		Fetches all the blogs
Http Method	GET	
Parameter 1	-	
Return	List<Blogs>	

## 5 TEMPLATE CODE STRUCTURE

---

### 5.1 PACKAGE: COM.YAKSHA.ASSESSMENT.BLOGS

#### Resources

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

### 5.2 PACKAGE: COM.YAKSHA.ASSESSMENT.BLOGS.ENTITY

#### Resources

Class/Interface	Description	Status
BlogEntity (class)	<ul style="list-style-type: none"><li>Annotate this class with proper annotation to declare it as an entity. class with <b>id</b> as primary key.</li><li>You can use the javax package for annotations.</li><li>Map this class with a <b>blog</b> table.</li><li>Generate the <b>id</b> using <b>IDENTITY</b> strategy</li></ul>	Partially implemented.

### 5.3 PACKAGE: COM.YAKSHA.ASSESSMENT.BLOGS.DTO

#### Resources

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

### 5.4 PACKAGE: COM.YAKSHA.ASSESSMENT.BLOGS.REPOSITORY

#### Resources

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

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

## 5.5 PACKAGE: COM.YAKSHA.ASSESSMENT.BLOGS.SERVICE

### Resources

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

## 5.6 PACKAGE: COM.YAKSHA.ASSESSMENT.BLOGS.SERVICE.IMPL

### Resources

Class/Interface	Description	Status
<b>BolgServiceImpl (class)</b>	<ul style="list-style-type: none"> <li>• Implements BlogService.</li> <li>• Need to provide implementation for Blog related functionalities</li> <li>• Add required repository dependency</li> <li>• Do not modify, add or delete any method signature</li> </ul>	To be implemented.

## 5.7 PACKAGE: COM.IIHT.TRAINING.BLOGS.EXCEPTION

### Resources

Class/Interface	Description	Status
ResourceNotFoundException (Class)	<ul style="list-style-type: none"><li>• Custom Exception to be thrown when trying to get the blog details.</li><li>• Need to create Exception Handler for same wherever needed (local or global)</li></ul>	Need to be created.

## 5.8 PACKAGE: COM.IIHT.TRAINING.BLOGS.CONTROLLER

### Resources

Class/Interface	Description	Status
BlogController (Class)	<ul style="list-style-type: none"><li>• Controller class to expose all rest-endpoints for Blogs related activities.</li><li>• May also contain local exception handler methods</li></ul>	To be implemented

## 6 DEPENDENCY REQUIREMENTS

---

- In the pom.xml file kindly add the dependency for mysql.
- In the pom.xml file kindly add the dependency for org.hibernate.validator.

## 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**
  - 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.