System Requirements Specification Index

For

Notes-App

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

Table of Contents

1 Project Abstract

Note App is Spring boot application with MongoDB (Embedded), where it allows any unregistered users (visitors) to manage the notes like create, view, modify and delete.

Visitors can perform the follow actions:

1. Allows to add a note
2. Allows to delete an existing note
3. Allows to update the status on go
4. Allows to search the notes on the basis of Author and Status
5. Allows to search any note based on the id.
6. Allows to display all the notes

2 Assumptions, Dependencies, Risks / Constraints

- While fetching the note by ID, if note id does not exist then the operation should throw a custom exception.
- While deleting the note by ID, if note id does not exist then the operation should throw a custom exception.
- While updating the status of note, if note id does not exist then the operation should throw a custom exception.
- 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 document object only
- Do not change, add, remove any existing methods in service layer
- In Repository interfaces, custom methods can be added as per requirements.
- Must not go and touch the test resources, as they will be used for Auto-Evaluation
- All RestEndpoint methods and Exception Handlers must return data wrapped in **ResponseEntity**

3 Business Validations

1. Note description is not null, min 5 and max 200 characters.
2. Note status is not null and it should be either “completed” or “pending”.
3. Note title is not null and min 5 and max 20 characters.
4. Note author is not null and min 5 and max 20 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 NoteController

URL Exposed	Purpose
/noteservice/all Http Method: GET Parameter 1: - Return: List<NotesDto>	Fetches all the notes by using Stream API and Lambda expression
/noteservice/add Http Method: POST Parameter 1: NotesDto Return: NotesDto	Add a new note
/noteservice/delete/{id} Http Method: DELETE Parameter 1: String (id) Return: NotesDto	Delete note with given note id
/noteservice/get/{id} Http Method: GET Parameter 1: String (id) Return: NotesDto	Fetches the note with the given id
/noteservice/update/{id}/{status} Http Method: PUT Parameter 1: String (id) Parameter 2: String (status) Return: NotesDto	Updates new status for the note with the given id
/noteservice/findByAuthor/{author} Http Method: GET Parameter 1: String (author) Return: List<NotesDto>	Fetches all the notes for the given author
/noteservice/findbyStatus/{status} Http Method: GET Parameter 1: String (status) Return: List<NotesDto>	Fetches all the notes for the given status

5 Template Code Structure

5.1 Package: com.example.notesservice

Resources

NotesserviceApplication (Class)	This is the SpringBoot starter class of the application.	Already Implemented

5.2 Package: com.example.notesservice.document

Resources

Class/Interface	Description	Status
Note (class)	○ Annotate this class with proper annotation to declare it as a document class with **Id** as primary key.	Partially implemented.

5.3 Package: com.example.notesservice.dto

Resources

Class/Interface	Description	Status
NotesDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attribute of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.
NotesExceptionResponse (class)	Object of this class is supposed to be returned in case of exception through exception handlers	Already implemented.

5.4 Package: com.example.notesservice.repo

Resources

Class/Interface	Description	Status
NoteRepository (interface)	1. Repository interface exposing CRUD functionality for **Note** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented

5.5 Package: com.example.notesservice.service

Resources

Class/Interface	Description	Status
NoteService (interface)	Interface to expose method signatures for note related functionality. Do not modify, add or delete any method	Already implemented.
NoteServiceImpl (class)	• Implements **NoteService**. Contains template method implementation. • Need to provide implementation for note related functionalities • Do not modify, add or delete any method signature	To be implemented.

5.6 PACKAGE: COM.EXAMPLE.NOTESSERVICE.EXCEPTION

Resources

Class/Interface	Description	Status
GlobalExceptionHandler (class)	• RestControllerAdvice Class for defining global exception handlers. • Contains Exception Handler for **NotesException** class. • Use this as a reference for creating exception handler for other custom exception classes	Partially implemented.

Class/Interface	Description	Status
InvalidNoteDataException (Class)	• Custom Exception to be thrown when note data received by client does not • Need to create Exception Handler for same wherever needed (local or global)	Already created.
NoteIdNotFoundException (Class)	• Custom Exception to be thrown when trying to perform any activity based on note id, which is not present • Need to create Exception Handler for same wherever needed (local or global)	Already created.

NotesException (Class)	• Custom Exception to be thrown for any other type of exception • Exception Handler for same is already created in global exception handler class	Already created.

5.7 PACKAGE: COM.EXAMPLE.NOTESSERVICE.CONTROLLER

Resources

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

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 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 noteservice-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. Mandatory: Before final submission run the following command:
    mvn test
11. 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.