System Requirements Specification

Index

For

Expense Tracker Application

Version 1.0

TABLE OF CONTENTS

EXPENSE TRACKER APPLICATION

System Requirements Specification

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Expense Tracker Application** is implemented using Spring Boot with a MySQL database. The application aims to provide a comprehensive platform for managing and organizing all expenses.

Following is the requirement specifications:

	Expense Tracker Application
Modules	
1	Expense
Expense Module Functionalities	
1	List all expenses
2	Get expense by id
3	Create expense
4	Update expense by id
5	Delete expense by id

2 Assumptions, Dependencies, Risks / Constraints

2.1 EXPENSE CONSTRAINTS

- When fetching an expense by ID, if the expense ID does not exist, the operation should throw a Not found exception.
- When updating an expense, if the expense ID does not exist, the operation should throw a Not found exception.
- When removing an expense, if the expense ID does not exist, the operation should throw a Not found exception.

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 - Product

- Name is not blank.
- Amount is not blank.
- Category is not blank.
- Date is not blank.

4 REST ENDPOINTS

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

4.1 EXPENSECONTROLLER

URL Exposed		Purpose
1. /api/expenses		Fetches all the expenses
Http Method	GET	
Parameter	-	
Return	List<Expense>	
2. /api/expenses/{id}		Get an expense by id
Http Method	GET	
Parameter 1	Long (id)	
Return	Expense	
3. /api/expenses		Create a new expense
Http Method	POST	
Parameter	-	
Return	Expense	
4. /api/expenses/{id}		Updates existing expense by id
Http Method	PUT	
Parameter 1	Long (id)	
Return	Expense	
5. /api/products/{id}		Deletes a expense by id
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.EXPENSETRACKER

Resources

ExpenseTrackerApplication (Class)	This is the Spring Boot starter class of the application.	Already Implemented

5.2 PACKAGE: COM.EXPENSETRACKER.REPOSITORY

Resources

Class/Interface	Description	Status
ExpenseRepository (interface)	• Repository interface exposing CRUD functionality for Expense Entity. • You can go ahead and add any custom methods as per requirements.	Partially implemented.

5.3 PACKAGE: COM.EXPENSETRACKER.SERVICE

Resources

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

5.4 PACKAGE: COM.EXPENSETRACKER.SERVICE.IMPL

Class/Interface	Description	Status
ExpenseServiceImpl (class)	• Implements ExpenseService. • Contains template method implementation. • Need to provide implementation for expense related functionalities. • Do not modify, add or delete any method signature	To be implemented.

5.5 PACKAGE: COM.EXPENSETRACKER.CONTROLLER

Resources

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

5.6 PACKAGE: COM.EXPENSETRACKER.DTO

Resources

Class/Interface	Description	Status
ExpenseDTO (Class)	Use appropriate annotations from the Java Bean Validation API for validating attributes of this class.	Partially implemented.

5.7 PACKAGE: COM.EXPENSETRACKER.ENTITY

Resources

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

5.8 PACKAGE: COM.EXPENSETRACKER.EXCEPTION

Resources

Class/Interface	Description	Status
NotFoundException (Class)	• Custom Exception to be thrown when trying to fetch or delete the expense info which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already implemented.

6 DEPENDENCY REQUIREMENTS

- In the pom.xml file kindly add the dependency for org.hibernate.validator with version 4.2.0.Final.
- In the pom.xml file kindly add the dependency for org.modelmapper with version 2.3.5.

7 EXECUTION STEPS TO FOLLOW FOR BACKEND

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. **cd into your backend project folder**
4. **To build your project use command:**

   mvn clean package -Dmaven.test.skip
5. **To launch your application, move into the target folder (cd target). Run the following command to run the application:**

   java -jar <your application jar file name>
6. **This editor Auto Saves the code.**
7. **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.**
8. **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.**
9. **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.**
10. **To test any UI based application the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**
11. **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.