System Requirements Specification

Index

For

Expense Tracker Application - JWT

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
2	User
Expense Module Functionalities	
1	Get an expense by id for user (data to be retrieved from jwt token)
2	Create an expense by user id
3	Update an expense by id
4	Delete an expense by id
5	List all expenses for user by user id
User Module Functionalities	
1	Generate token
2	Create new user

2 Assumptions, Dependencies, Risks / Constraints

2.1 Expense constraints

- When fetching an expense by ID, if the expense ID does not exist, the service method should throw a ResourceNotFoundException with “Expense not found.” message.
- When updating an expense, if the expense ID does not exist, the service method should throw a ResourceNotFoundException with “Expense not found.” message.
- When removing an expense, if the expense ID does not exist, the service method should throw a ResourceNotFoundException with “Expense not found.” message.

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

- Expense Name should not be blank.
- Expense Amount should not be null and must be positive value.
- Expense Category should not be blank.
- User ID in expense should not be null.
- Username should not be blank.
- User email should not be blank.
- User password should not be 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/{expenseId}		Fetch expense by id
Http Method	GET	
Path variable	Long (expenseId)	
Return	Expense	
2. /api/expenses/{userId}		Create a new expense for user
Http Method	POST	
Path variable	Long (userId) **The expense data to be created must be received in the controller using @RequestBody.**	
Return	Expense	
3. /api/expenses/{expenseId}		Updates an existing expense by id
Http Method	PUT	
Path variable	Long (expenseId) **The expense data to be updated must be received in the controller using @RequestBody.**	
Return	Expense	
4. /api/expenses/{expenseId}		Delete an existing expense by id
Http Method	DELETE	
Path variable	Long (expenseId)	
Return	-	
5. /api/expenses/user/{userId}		Fetches all expenses by user id
Http Method	GET	
Path variable	Long (userId)	
Return	List <Expense>	

4.2 USERCONTROLLER

URL Exposed		Purpose
1./api/auth/generateToken		Generate a new token
Http Method	POST	

Parameter 1	AuthRequest {
	username
	password
	}	

Return	String (token)	
2./api/auth/addNewUser		Create a new user
Http Method	POST	
Parameter 1	User	
Return	String	

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.

UserInfoRepository (interface)	● Repository interface exposing CRUD functionality for User 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.
JwtService (class)	● Contains template method implementation to jwt utilities. ● Need to provide implementation for all functionalities. ● Do not modify, add or delete any method signature.	To be implemented.

UserInfoDetails (class)	● Implements UserDetails. ● Contains template method implementation. ● Need to provide implementation for user info details related functionalities. ● Do not modify, add or delete any method signature	To be implemented.
UserInfoService (class)	● Implements UserDetailsService. ● Contains template method implementation. ● Need to provide implementation for all undefined 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
UserController (Class)	● Controller class to expose all rest-endpoints for user 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 for validating attributes of this class.	Partially implemented.
UserDTO (Class)	● Use appropriate annotations 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 expenses **table**. ● Generate the **id** using the IDENTITY strategy	Partially implemented.
User (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 a user **table**. ● Generate the **id** using the IDENTITY strategy	Partially implemented.
AuthRequest(Class)	● This class is already implemented. ● This should be used for taking input for auth requests.	Already implemented.

5.8 PACKAGE: COM.EXPENSETRACKER.EXCEPTION

Resources

Class/Interface	Description	Status
ResourceNotFoundException (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.
ErrorResponse (Class)	● RestControllerAdvice Class for defining global exception handlers. ● Contains Exception Handler for **InvalidDataException** class. ● Use this as a reference for creating exception handlers for other custom exception classes.	Already implemented.
RestExceptionHandler (Class)	● RestControllerAdvice Class for defining rest exception handlers. ● Contains Exception Handler for **ResourceNotFoundException** class. ● Use this as a reference for creating exception handlers for other custom exception classes.	Already implemented.

5.9 PACKAGE: COM.EXPENSETRACKER.CONFIG

Resources

Class/Interface	Description	Status
SecurityConfig (Class)	● Provides a filter that intercepts the request and authenticates the user.	Need to be implemented.

5.10 Package: com.expensetracker.filter

Resources

Class/Interface	**Description**	**Status**
JwtAuthFilter (Class)	● Responsible for processing incoming requests by inspecting the "Authorization" header to identify and validate a Bearer token.	Partially implemented.

6 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. **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. **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.**
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