System Requirements Specification

Index

For

Expense Tracker Application - JWT

Version 1.0

TABLE OF CONTENTS

BACKEND-SPRING BOOT RESTFUL APPLICATION 3

1 Project Abstract 3

2 Assumptions, Dependencies, Risks / Constraints 4

2.1 Expense Constraints 4

2.2 Common Constraints 4

3 Business Validations 4

4 Rest Endpoints 5

4.1 ExpenseController 5

4.2 UserController 6

5 Template Code Structure 6

5.1 Package: com.expensetracker 6

5.2 Package: com.expensetracker.repository 6

5.3 Package: com.expensetracker.service 7

5.4 Package: com.expensetracker.service.impl 7

5.5 Package: com.expensetracker.controller 8

5.6 Package: com.expensetracker.dto 9

5.7 Package: com.expensetracker.entity 9

5.8 Package: com.expensetracker.exception 10

5.9 Package: com.expensetracker.config 10

5.10 Package: com.expensetracker.filter 11

6 METHOD DESCRIPTIONS 12

6.1 Controller Class - Method Descriptions 12

6.2 ServiceImpl Class - Method Descriptions 13

6.3 Config Class - Method Descriptions 17

6.4 Filter Class - Method Descriptions 17

7 Execution Steps to Follow for Backend 18

EXPENSE TRACKER APPLICATION

System Requirements Specification

BACKEND-SPRING BOOT RESTFUL APPLICATION

- ### 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

● Assumptions, Dependencies, Risks / Constraints

●.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 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**

● 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.

● Database Operations

- Expense class must be linked with the “expenses” table in the database.
- Id must be treated as primary key and should be generated using IDENTITY technique.
- Name column should not accept null values.
- Amount column should not accept null values.

- Category column should not accept null values.
- Date should be Temporal.DATE.
- User field in Expense should be linked with “user_id”.
- Id field in User must be treated as primary key and should be generated using IDENTITY technique.

Rest Endpoints

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

●.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>	

●.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	

● Template Code Structure

●.1 Package: com.expensetracker

Resources

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

●.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.

●.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.

●.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 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

●.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.

●.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.

●.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.	Already implemented.

	● Use this as a reference for creating exception handlers for other custom exception classes.	

●.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.

●.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.

● METHOD DESCRIPTIONS

●.1 Controller Class - Method Descriptions:

1. ExpenseController - Implementation Guidelines

Method	**Task**	**Implementation Details**
`getExpenseById`	To fetch a specific expense by ID	- Request type: GET with URL `/api/expenses/{expenseId}` - Method name: `getExpenseById` returning `ResponseEntity<Expense>` - Use `@PathVariable` to get `expenseId` - Call `expenseService.getExpenseById(expenseId)` - If found, return with `HttpStatus.OK` - If not, catch `ResourceNotFoundException` and return `HttpStatus.NOT_FOUND`

`createExpense`	To create a new expense for a given user	- Request type: POST with URL `/api/expenses/{userId}` - Method name: `createExpense` returning `ResponseEntity<Expense>` - Use `@RequestBody` for `Expense` input and `@PathVariable` for `userId` - Call `expenseService.createExpense(expense, userId)` - Return created expense with `HttpStatus.CREATED` - If user not found, catch `ResourceNotFoundException` and return `HttpStatus.NOT_FOUND`
`updateExpense`	To update an existing expense	- Request type: PUT with URL `/api/expenses/{expenseId}` - Method name: `updateExpense` returning `ResponseEntity<Expense>` - Use `@PathVariable` for `expenseId` and `@RequestBody` for new data - Call `expenseService.updateExpense(expenseId, expenseDetails)` - Return updated expense with `HttpStatus.OK` - If not found, catch `ResourceNotFoundException` and return `HttpStatus.NOT_FOUND`
`deleteExpense`	To delete an expense by its ID	- Request type: DELETE with URL `/api/expenses/{expenseId}` - Method name: `deleteExpense` returning `ResponseEntity<Void>` - Use `@PathVariable` to get `expenseId` - Call `expenseService.deleteExpense(expenseId)` - Return `HttpStatus.NO_CONTENT` if successful - Catch `ResourceNotFoundException` and return `HttpStatus.NOT_FOUND`
`getAllExpensesForUser`	To fetch all expenses related to a particular user	- Request type: GET with URL `/api/expenses/user/{userId}` - Method name: `getAllExpensesForUser` returning `ResponseEntity<List<Expense>>` - Use `@PathVariable` to get `userId` - Call `expenseService.getAllExpensesForUser(userId)` - Return the list of expenses with `HttpStatus.OK`

2. UserController - Implementation Guidelines

Method	**Task**	**Implementation Details**
`authenticateAndGetToken`	To authenticate a user and generate JWT token	- Request type: POST with URL `/api/auth/generateToken` - Method name: `authenticateAndGetToken` returning `String` - Accept `AuthRequest` using `@RequestBody` - Use `authenticationManager.authenticate(...)` with username and password

		- If authentication is successful, generate token using `jwtService.generateToken(...)` - If authentication fails, throw `UsernameNotFoundException` with message 'invalid user request !'
addNewUser	To register a new user in the system	- Request type: POST with URL `/api/auth/addNewUser` - Method name: `addNewUser` returning `String` - Accept `User` entity using `@Valid @RequestBody` - Call `service.addUser(user)` to create a new user - Return the result message as response

●.2 ServiceImpl Class - Method Descriptions

1. ExpenseServiceImpl - Implementation Guidelines

Method	**Task**	**Implementation Details**
createExpense	To create a new expense entry for a user	- Accepts an `Expense` object and `userId` - Use `userRepository.findById(userId)` to check if the user exists - If present, set user to expense and call `expenseRepository.save(expense)` - If user not found, throw `ResourceNotFoundException` with message 'User not found'
getExpenseById	To retrieve a specific expense by ID	- Use `expenseRepository.findById(expenseId)` to find the expense - If not found, throw `ResourceNotFoundException` with message 'Expense not found' - If found, return the expense object
updateExpense	To update an existing expense record	- Retrieve expense using `expenseRepository.findById(expenseId)` - If not found, throw `ResourceNotFoundException` with message 'Expense not found' - Update fields: name, amount, category, date, and note - Save updated entity using `expenseRepository.save(expense)` - Return the updated expense

deleteExpense	To delete an expense by ID	- Retrieve expense using `expenseRepository.findById(expenseId)` - If not found, throw `ResourceNotFoundException` with message 'Expense not found' - Call `expenseRepository.delete(expense)` to delete the entry
getAllExpensesForUser	To fetch all expenses for a specified user	- Accepts `userId` as input - Use `expenseRepository.findByUserId(userId)` to fetch expenses - Return the list of expenses

2. JwtService - Implementation Guidelines

Method	Task	Implementation Details
generateToken	To generate a JWT token for a given username	- Method name: `generateToken` returning `String` - Accepts `userName` as input - Creates an empty `claims` map - Calls `createToken(claims, userName)` and returns the result
createToken	To create a JWT token using claims and username	- Method name: `createToken` (private) - Sets subject and issue/expiration date - Signs token using `HS256` with signing key - Returns the compact JWT string
getSignKey	To retrieve the signing key for JWT	- Method name: `getSignKey` (private) - Decodes the `SECRET` key using Base64 - Uses `Keys.hmacShaKeyFor(keyBytes)` to return `Key` object
extractUsername	To extract username from a JWT token	- Calls `extractClaim(token, Claims::getSubject)` - Returns the subject (username) from the token
extractExpiration	To extract expiration date from a JWT token	- Calls `extractClaim(token, Claims::getExpiration)` - Returns expiration date from token
extractClaim	To extract a specific claim from the JWT token	- Generic method using a `claimsResolver` function - Retrieves all claims via `extractAllClaims` - Applies resolver and returns result
extractAllClaims	To get all claims from the token body	- Parses the token using `Jwts.parserBuilder()` with signing key - Returns the `Claims` object from token body

isTokenExpired	To check if token is expired	- Calls `extractExpiration(token)` - Compares with current date - Returns true if token is expired
validateToken	To validate JWT token against user details	- Extracts username from token using `extractUsername` - Checks if username matches and token is not expired - Returns true if valid, false otherwise

3. UserInfoDetails - Implementation Guidelines

Method	Task	Implementation Details
UserInfoDetails	Constructor to initialize user details	- Accepts a `User` object as parameter - Initializes `name` and `password` from the user - Splits roles string and maps to `SimpleGrantedAuthority` - Collects into `authorities` list
getAuthorities	To return the list of granted authorities for user	- Returns the list of `GrantedAuthority` initialized in constructor
getPassword	To get the user's password	- Returns the user's password
getUsername	To get the user's name/username	- Returns the user's name
isAccountNonExpired	To check if the account is not expired	- Always returns `true` (account never expires)
isAccountNonLocked	To check if the account is not locked	- Always returns `true` (account never locked)
isCredentialsNonExpired	To check if credentials are not expired	- Always returns `true` (credentials never expire)
isEnabled	To check if account is enabled	- Always returns `true` (account always enabled)

4. UserInfoService - Implementation Guidelines

Method	Task	Implementation Details

`loadUserByUsername`	To load user details by username for authentication	- Method from `UserDetailsService` interface - Accepts `username` as input - Calls `repository.findByName(username)` to fetch user - If user is present, wraps it in `UserInfoDetails` and returns - If not, throws `UsernameNotFoundException` with message 'User not found'
`addUser`	To add a new user to the system with encrypted password	- Accepts a `User` object - Encodes the user's password using `PasswordEncoder` - Saves the user using `repository.save(user)` - Returns success message: 'User Added Successfully'

●.3 Config Class - Method Descriptions

1. SecurityConfig - Implementation Guidelines

Method	**Task**	**Implementation Details**
`userDetailsService`	To provide user detail service bean for authentication	- Method returns a new instance of `UserInfoService` - Used for loading user-specific data during authentication
`securityFilterChain`	To configure HTTP security and JWT filter chain	- Disables CSRF protection - Permits access to `/api/auth/addNewUser` and `/api/auth/generateToken` - Requires authentication for `/api/expenses/**` - Configures stateless session policy - Sets custom `AuthenticationProvider` - Adds `JwtAuthFilter` before `UsernamePasswordAuthenticationFilter`
`passwordEncoder`	To provide password encoding bean	- Returns a new instance of `BCryptPasswordEncoder` - Used to encode and verify user passwords securely
`authenticationProvider`	To set up the authentication provider using DAO	- Creates `DaoAuthenticationProvider` - Sets userDetailsService and passwordEncoder - Returns the configured provider
`authenticationManager`	To provide an authentication manager bean	- Accepts `AuthenticationConfiguration` - Returns `AuthenticationManager` from the config

●.4 Filter Class - Method Descriptions

1. JwtAuthFilter - Implementation Guidelines

Method	Task	Implementation Details
`doFilterInternal`	To intercept HTTP requests and apply JWT-based authentication	- Override method from `OncePerRequestFilter` - Extract `Authorization` header from the request - Check if header starts with `Bearer ` and extract JWT token - Use `jwtService.extractUsername(token)` to get username - Check if user is not already authenticated - Load user details using `userDetailsService.loadUserByUsername(username)` - Validate token with `jwtService.validateToken(token, userDetails)` - If valid, create `UsernamePasswordAuthenticationToken` and set it in `SecurityContextHolder` - Call `filterChain.doFilter(request, response)` to continue request processing

- ## 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