System Requirements Specification

Index

For

Matrimony Application - JWT

Version 1.0

TABLE OF CONTENTS

MATRIMONY APPLICATION

System Requirements Specification

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

The **Matrimony Application** is implemented using Spring Boot with a MySQL database. The application aims to provide a comprehensive platform for managing and registering different types of volunteers for different types of programs.

Following is the requirement specifications:

	Matrimony Application
Modules	
1	User
1	Partner Preferences
User Module Functionalities	
1	Login (to send jwt token)
2	Register an User
3	Get an user profile by id
4	Get all matches
5	Update an user profile by id
6	Delete an user profile by id
Partner Preferences Module Functionalities	
1	Create partner preferences by user id
2	Get partner preferences by user id
3	Update partner preferences by user id
4	Delete preferences by user id

2 Assumptions, Dependencies, Risks / Constraints

2.1 User Constraints

- When fetching a user in loadUserByUsername, if the username does not exist, the method should throw a UsernameNotFoundException with “User not found” message.
- When fetching a user profile by ID, if the user ID does not exist, the service method should throw a RuntimeException with “User not found.” message.
- When updating a user profile by ID, if the user profile ID does not exist, the service method should throw a RuntimeException with “User not found.” message.

2.2 Partner Preferences Constraints

- When creating a partner preference by user ID, if the user ID does not exist, the service method should throw a RunTimeException with “User not found.” message.
- When fetching a partner preference by user ID, if the preference does not exist, the service method should throw a RunTimeException with “Partner Preferences not found.” message.
- When updating a partner preference by user ID, if the preference does not exist, the service method should throw a RunTimeException with “Partner Preferences not found.” message.

2.3 Common Constraints

- For all rest endpoints receiving @RequestBody, validation checks must be done and must throw custom exceptions 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 the service layer.
- In Repository interfaces, custom methods can be added as per requirements.

3 BUSINESS VALIDATIONS (UserDTO)

- Name should not be blank.
- Email should not be blank and must be of type email.
- Password should not be blank.
- Gender should not be blank.

4 DATABASE OPERATIONS

- User Entity File

  - User entity class should have “users” as table name.
  - Id should be considered as Primary key and generated using IDENTITY technique.
  - Name should not accept null value.
  - Email should not accept null value.
  - Password should not accept null value.
  - Roles should not accept null value.
  - Dob should not accept null value.
  - Gender should not accept null value.

- PartnerPreference Entity File

  - PartnerPreference entity class should have “partner_preferences” as table name.
  - Id should be considered as Primary key and generated using IDENTITY technique.
  - PreferredAgeStart should have the column name as “preferred_age_start”.
  - PreferredAgeEnd should have the column name as “preferred_age_end”.
  - PreferredGender should have the column name as “preferred_gender”.
  - User should have the column name “user_id”.

5 REST ENDPOINTS

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

5.1 USER CONTROLLER

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /api/users/login
<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter</td><td>AuthRequest {
email
password
}</td></tr><tr><td>Return</td><td>String (token)</td></tr></table></td><td>Login the user and return token</td></tr>
<tr><td>2. /api/users/register
<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>The user data to be created must be received in the controller using @RequestBody.</td></tr><tr><td>Return</td><td>UserDTO</td></tr></table></td><td>Creates a new user</td></tr>
<tr><td>3. /api/users/profile/{userId}
<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Path vaiable</td><td>Long userId</td></tr><tr><td>Return</td><td>UserDTO</td></tr></table></td><td>Fetches the user profile by id</td></tr>
<tr><td>4./api/users/profile
<table><tr><td>Http Method</td><td>PUT</td></tr><tr><td>Parameter 1</td><td>Long (id)

The user data to be updated must be received in the controller using @RequestBody.</td></tr><tr><td>Return</td><td>UserDTO</td></tr></table></td><td>Updates an user by id</td></tr>
<tr><td>5. /api/users
<table><tr><td>Http Method</td><td>DELETE</td></tr><tr><td>Parameter 1</td><td>Long (id)</td></tr><tr><td>Return</td><td>-</td></tr></table></td><td>Delete an user by id</td></tr>
</table>

6. /api/users/matches Http Method Parameter Return	 GET Long (userId) List <UserDTO>	Find all matches for user by id

5.2 PARTNERPREFERENCES CONTROLLER

URL Exposed		Purpose
1. /api/preferences Http Method Parameter Return	 POST **The partner preference data to be created must be received in the controller using @RequestBody.** PartnerPreferencesDTO	Creates a new partner preferences
2. /api/preferences Http Method Parameter 1 Return	 GET Long (userId) PartnerPreferencesDTO	Gets a partner preferences by user id
3. /api/profiles Http Method Parameter 1 Return	 DELETE Long (userId) -	Deletes a partner preferences by id
4. /api/profiles Http Method Parameter 1 Return	 PUT Long (userId) **The user profile data to be updated must be received in the controller using @RequestBody.** PartnerPreferencesDTO	Updates a partner preferences by user id

6 TEMPLATE CODE STRUCTURE

6.1 PACKAGE: COM.MATRIMONYAPPLICATION

Resources

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

6.2 PACKAGE: COM.MATRIMONYAPPLICATION.REPOSITORY

Resources

Class/Interface	Description	Status
UserRepository (interface)	• Repository interface exposing CRUD functionality for User Entity. • You can go ahead and add any custom methods as per requirements.	Already Implemented
PartnerPreferencesRepository (interface)	• Repository interface exposing CRUD functionality for PartnerPreferences Entity. • You can go ahead and add any custom methods as per requirements.	Already Implemented

6.3 PACKAGE: COM.MATRIMONYAPPLICATION.SERVICE

Resources

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

6.4 PACKAGE: COM.MATRIMONYAPPLICATION.SERVICE.IMPL

Class/Interface	Description	Status
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.
PartnerPreferencesServiceImpl (class)	• Implements PartnerPreferencesService. • Contains template method implementation. • Need to provide implementation for partner preferences related 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.
UserServiceImpl (class)	• Implements UserDetailsService. • Contains template method implementation. • Need to provide implementation for user info service related functionalities. • Do not modify, add or delete any	To be implemented.

	method signature.	

6.5 Package: Com.Matrimonyapplication.Controller

Resources

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

6.6 Package: Com.Matrimonyapplication.Dto

Resources

Class/Interface	Description	Status
UserDTO (Class)	• Use appropriate annotations for validating attributes/fields of this class.	Partially implemented.
PartnerPreferencesDTO (Class)	• Use appropriate annotations for validating attributes/fields of this class.	Partially implemented.

6.7 Package: Com.Matrimonyapplication.Entity

Resources

Class/Interface	Description	Status

PartnerPreferences (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 partner_preferences **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 users **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.

6.8 PACKAGE: COM.MATRIMONYAPPLICATION.EXCEPTION

Class/Interface	Description	Status
ResourceNotFoundException (Class)	• Custom Exception to be thrown when trying to fetch, update or delete the user profile 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 handler 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 handler for other custom exception classes	Already implemented.

6.9 PACKAGE:COM.MATRIMONYAPPLICATION.CONFIG

Resources

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

6.10 PACKAGE: COM.MATRIMONYAPPLICATION.FILTER

Class/Interface	Description	Status
JwtAuthFilter (Class)	• Responsible for processing incoming requests by inspecting the “Authorization” header to identify and validate a Bearer	Partially implemented.

	token.	

7 METHOD DESCRIPTIONS

7.1 Controller Class - Method Descriptions:

1. UserController - Implementation Guidelines

Method	Task	Implementation Details
`authenticateAndGetToken`	To authenticate user and return JWT token	- Request type: POST with URL `/api/users/login` - Method name: `authenticateAndGetToken` returning `String` - Use `@RequestBody` to accept `AuthRequest` - Call `authenticationManager.authenticate()` and `jwtService.generateToken()` - If authenticated, return JWT token - Else, throw `UsernameNotFoundException` with the message: "invalid user request !"
`registerUser`	To register a new user	- Request type: POST with URL `/api/users/register` - Method name: `registerUser` returning `ResponseEntity<UserDto>` - Use `@RequestBody` to accept `UserDto` - Call `userService.registerUser(userDto)` - Return registered user with `HttpStatus.OK`
`getUserProfile`	To retrieve user profile by user ID	- Request type: GET with URL `/api/users/profile/{userId}` - Method name: `getUserProfile` returning `ResponseEntity<UserDto>` - Use `@PathVariable` for `userId` - Call `userService.getUserProfile(userId)` - Return user profile with `ResponseEntity.ok`
`updateUserProfile`	To update user profile	- Request type: PUT with URL `/api/users/profile` - Method name: `updateUserProfile` returning `ResponseEntity<UserDto>` - Use `@RequestParam` for `userId` and `@RequestBody` for `UserDto` - Call `userService.updateUserProfile(userId, userDto)` - Return updated profile with `ResponseEntity.ok`
`deleteUser`	To delete a user by user ID	- Request type: DELETE with URL `/api/users` - Method name: `deleteUser` returning `ResponseEntity<?>` - Use `@RequestParam` for `userId`

		- Call `userService.deleteUser(userId)` - Return `ResponseEntity.ok().build()`
`findMatches`	To find matches for a user	- Request type: GET with URL `/api/users/matches` - Method name: `findMatches` returning `ResponseEntity<List<UserDto>>` - Use `@RequestParam` for `userId` - Call `userService.findMatches(userId)` - Return list of matches with `ResponseEntity.ok`

2. PartnerPreferencesController - Implementation Guidelines

Method	Task	Implementation Details
`createPreferences`	To create partner preferences for a user	- Request type: POST with URL `/api/preferences` - Method name: `createPreferences` returning `ResponseEntity<PartnerPreferencesDto>` - Use `@RequestBody` for input and `@RequestParam` for `userId` - Call `partnerPreferencesService.createPartnerPreferences(userId, partnerPreferencesDto)` - Return the created PartnerPreferencesDto with `ResponseEntity.ok`
`getPreferences`	To fetch the partner preferences for a specific user	- Request type: GET with URL `/api/preferences` - Method name: `getPreferences` returning `ResponseEntity<PartnerPreferencesDto>` - Use `@RequestParam` to get `userId` - Call `partnerPreferencesService.getPartnerPreferences(userId)` - Return the retrieved PartnerPreferencesDto with `ResponseEntity.ok`
`updatePreferences`	To update existing partner preferences for a user	- Request type: PUT with URL `/api/preferences` - Method name: `updatePreferences` returning `ResponseEntity<PartnerPreferencesDto>` - Use `@RequestBody` for input and `@RequestParam` for `userId` - Call `partnerPreferencesService.updatePartnerPreferences(userId, partnerPreferencesDto)` - Return the updated PartnerPreferencesDto with `ResponseEntity.ok`

`deletePreferences`	To delete partner preferences of a user	- Request type: DELETE with URL `/api/preferences` - Method name: `deletePreferences` returning `ResponseEntity<?>` - Use `@RequestParam` to get `userId` - Call `partnerPreferencesService.deletePartnerPreferences(userId)` - Return with `ResponseEntity.ok` after deletion

7.2 ServiceImpl Class - Method Descriptions

1. PartnerPreferencesServiceImpl - Implementation Guidelines

Method	Task	Implementation Details
`createPartnerPreferences`	To create partner preferences for a given user	- Find the user using userId with userRepository.findById(userId). - Create a new PartnerPreferences object and populate it using values from PartnerPreferencesDto. - Set the user to the preferences entity. - Save the PartnerPreferences using partnerPreferencesRepository.save(). - Set the generated ID back into the DTO and return it. - Throws RuntimeException if user is not found with the message:"User not found.".
`getPartnerPreferences`	To fetch partner preferences for a specific user	- Call partnerPreferencesRepository.findByUserId(userId) to get preferences. - If null, throw new RuntimeException("Partner Preferences not found."). - If found, convert the entity to DTO using convertToDto() and return it.
`updatePartnerPreferences`	To update existing partner preferences for a user	- Call partnerPreferencesRepository.findByUserId(userId). - If null, throw RuntimeException("Partner Preferences not found."). - Update the entity fields using values from PartnerPreferencesDto. - Save the updated entity using

		partnerPreferencesRepository.save(). - Convert the updated entity to DTO and return it.
`deletePartnerPreferences`	To delete partner preferences for a given user	- Call partnerPreferencesRepository.findByUserId(userId). - If not null, call partnerPreferencesRepository.delete(preferences).

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	- Method name: `extractUsername` - Calls `extractClaim(token, Claims::getSubject)` - Returns the subject (username) from the token
`extractExpiration`	To extract expiration date from a JWT token	- Method name: `extractExpiration` - Calls `extractClaim(token, Claims::getExpiration)` - Returns the expiration date from the token
`extractClaim`	To extract a specific claim using a resolver	- Method name: `extractClaim` - Accepts JWT token and claimsResolver function - Calls `extractAllClaims` to get all claims - Applies resolver and returns the extracted value
`extractAllClaims`	To extract all claims from JWT token	- Method name: `extractAllClaims` (private) - Uses `Jwts.parserBuilder().setSigningKey(...).build().parseClaimsJws(token)` - Returns all claims from token
`isTokenExpired`	To check if the JWT token is expired	- Method name: `isTokenExpired` (private) - Calls `extractExpiration(token).before(new Date())` - Returns boolean indicating if the token is expired

`validateToken`	To validate a token against a user's username	- Method name: `validateToken` - Accepts JWT token and `UserDetails` object - Compares token username with `userDetails.getUsername()` - Checks if 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. UserServiceImpl - Implementation Guidelines

Method	Task	Implementation Details

`loadUserByUsername`	To load a user by username for authentication	- Method name: `loadUserByUsername` returning `UserDetails` - Accepts `username` as input - Calls `userRepository.findByName(username)` to find the user - Returns `UserInfoDetails` if user is found - If user not found, throws `UsernameNotFoundException` with the message: User not found
`authenticateUser`	To authenticate a user using email and password	- Method name: `authenticateUser` returning `UserDto` - Accepts `email` and `password` as input - Currently throws `UnsupportedOperationException` with the message: Authentication logic needs to be implemented
`getUserProfile`	To retrieve a user's profile by userId	- Method name: `getUserProfile` returning `UserDto` - Accepts `userId` as input - Calls `userRepository.findById(userId)` - If user found, returns the user as `UserDto` - If not found, throws `RuntimeException` with the message: User not found.
`updateUserProfile`	To update user profile information	- Method name: `updateUserProfile` returning `UserDto` - Accepts `userId` and `UserDto` as input - Calls `userRepository.findById(userId)` and updates name, dob, gender - Saves updated user and returns updated `UserDto` - If not found, throws `RuntimeException` with the message: User not found.
`deleteUser`	To delete a user by userId	- Method name: `deleteUser` returning `void` - Accepts `userId` as input - Calls `userRepository.deleteById(userId)`
`findMatches`	To find potential matches for a user	- Method name: `findMatches` returning `List<UserDto>` - Accepts `userId` as input - Currently throws `UnsupportedOperationException` with the message: Matching logic needs to be implemented.

7.3 Config Class - Method Descriptions

1. SecurityConfig - Implementation Guidelines

Method	Task	Implementation Details
`userDetailsService`	Creates UserDetailsService Bean	- Method name: `userDetailsService` returning `UserDetailsService` - Returns an instance of `UserServiceImpl`
`securityFilterChain`	Configures HTTP security and JWT authentication	- Method name: `securityFilterChain` returning `SecurityFilterChain` - Accepts `HttpSecurity` as input

		- Disables CSRF protection - Permits access to `/api/users/register` and `/api/users/login` - Secures `/api/preferences/**` endpoints to authenticated users only - Configures stateless session management - Registers custom authentication provider and JWT filter
`passwordEncoder`	Defines password encoding strategy	- Method name: `passwordEncoder` returning `PasswordEncoder` - Returns a `BCryptPasswordEncoder` instance for password hashing
`authenticationProvider`	Sets authentication provider details	- Method name: `authenticationProvider` returning `AuthenticationProvider` - Creates `DaoAuthenticationProvider` instance - Sets userDetailsService to `userDetailsService()` - Sets password encoder to `passwordEncoder()`
`authenticationManager`	Exposes authentication manager bean	- Method name: `authenticationManager` returning `AuthenticationManager` - Accepts `AuthenticationConfiguration` as input - Returns the configured `AuthenticationManager`

7.4 Filter Class - Method Descriptions

1. JwtAuthFilter - Implementation Guidelines

Method	Task	Implementation Details
`doFilterInternal (Overridden from OncePerRequestFilter)`	Filters each HTTP request to validate JWT token and authenticate user.	- Method name: `doFilterInternal` returning `void` - Accepts `HttpServletRequest`, `HttpServletResponse`, and `FilterChain` - Extracts the `Authorization` header from the HTTP request - Verifies it starts with `Bearer ` and extracts the token - Calls `jwtService.extractUsername(token)` to get the username - If username is not null and authentication is not set in `SecurityContextHolder` - Loads user details using `userDetailsService.loadUserByUsername(username)` - Validates token with `jwtService.validateToken(token, userDetails)` - If valid, creates and sets `UsernamePasswordAuthenticationToken` - Sets authentication in `SecurityContextHolder`

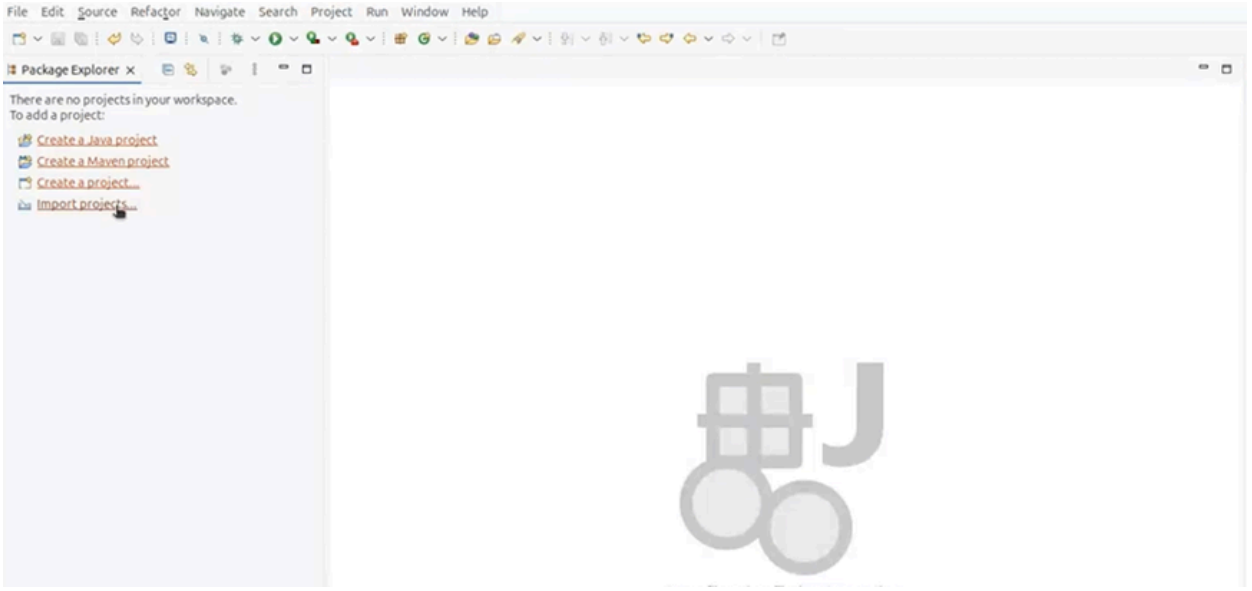
		- Proceeds with filter chain using `filterChain.doFilter(request, response)`

8 EXECUTION STEPS TO FOLLOW FOR BACKEND

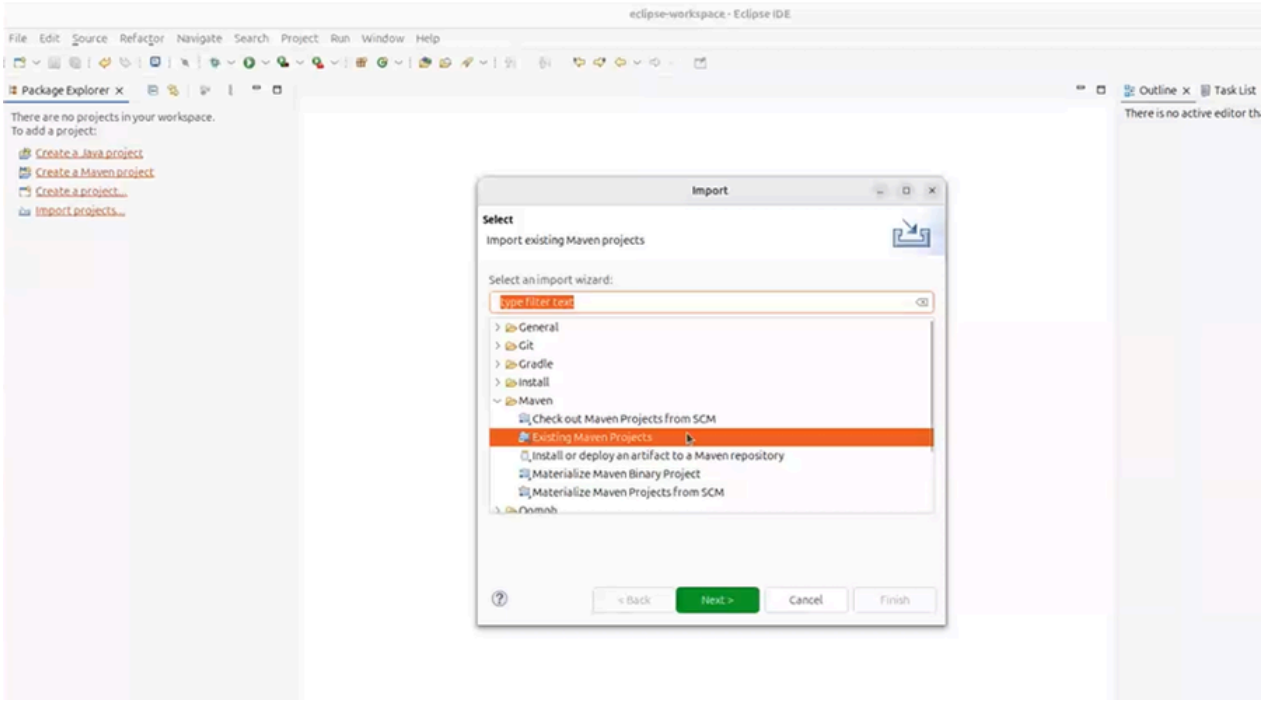
Execution Steps:

Steps for Execution:

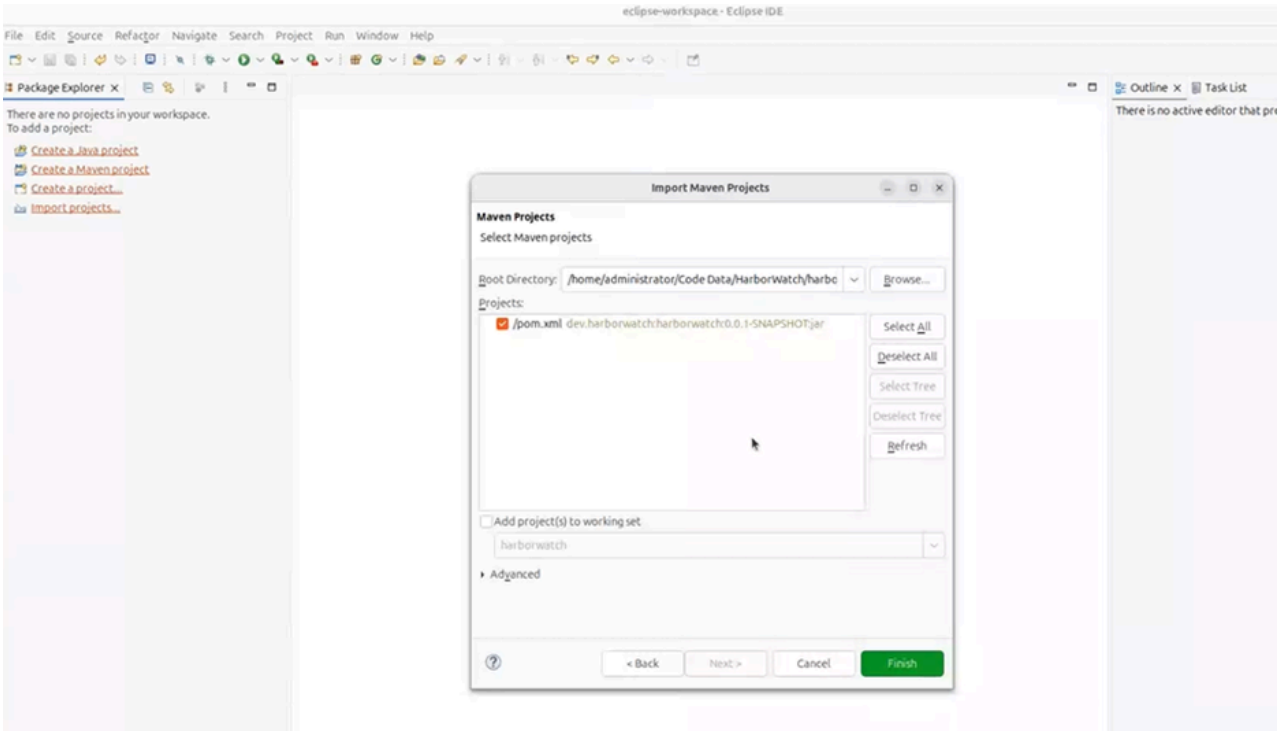
1. Launch the Eclipse IDE on your system. In the **Package Explorer**, click **Import projects…**.



2. In the **Import Wizard**, expand the **Maven** folder. Choose **Existing Maven Projects** → click **Next**.

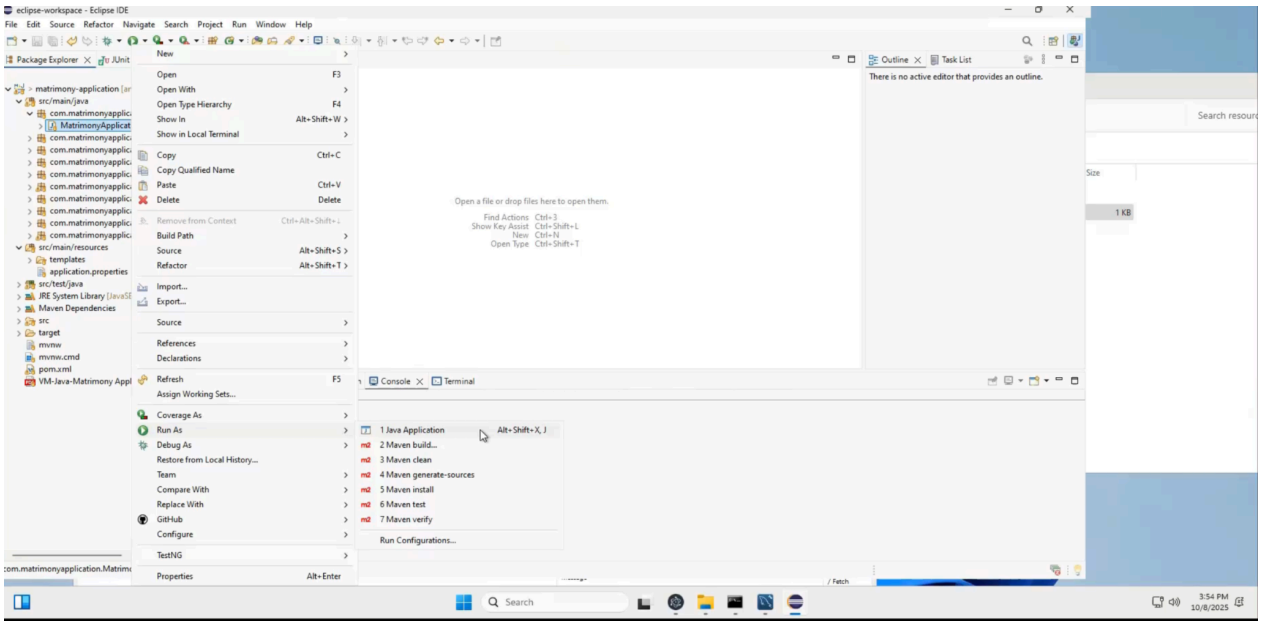


3. In the **Root Directory**, browse to your project folder (the one that contains `pom.xml`). Select the project → click **Finish**.



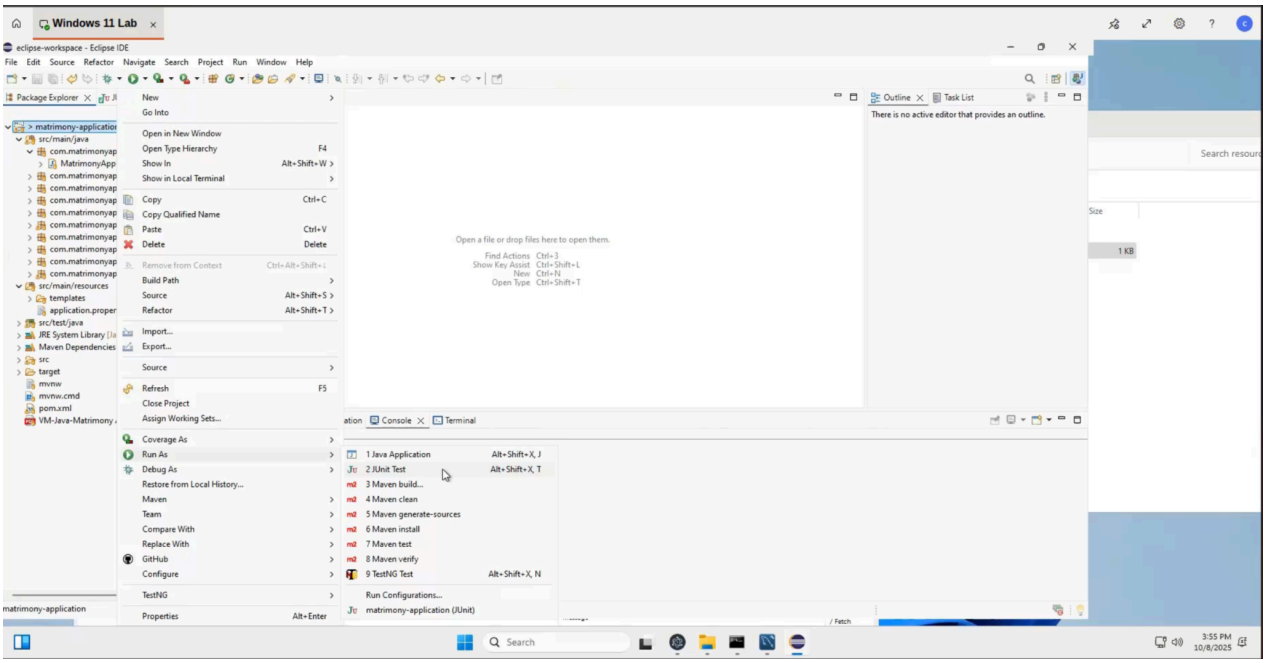
4. After importing the project into Eclipse:

   - Open the main application class (for example, `MatrimonyApplication.java`)
   - **Right-click** on that main class file in the **Project Explorer** (as shown in the image below).
   - Select **Run As** → **Java Application** to start the Spring Boot application.



5. To execute the test cases, **right-click on the project folder** (for example, `matrimony-application`), then

select **Run As → JUnit Test**.



This will run all the JUnit test cases inside the project, and the results will be displayed in the **JUnit console** at the bottom of Eclipse.

If you need to access the MySQL database, Workbench is already installed and available for use.

==========