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

- Username should not be blank.
- User email should not be blank and must be of type email.
- User password should not be blank.
- User gender 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 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</td></tr></table></td><td>Updates an user by id</td></tr>
</table>

	updated must be received in the controller using @RequestBody.	
Return	UserDTO	

5. /api/users		Delete an user by id
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	

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

4.2 PARTNERPREFERENCES CONTROLLER

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

Parameter 1	Long (userId) **The user profile data to be updated must be received in the controller using @RequestBody.**	Updates a partner preferences by user id
Return	PartnerPreferencesDTO	

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.MATRIMONYAPPLICATION

Resources

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

5.2 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

5.3 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.

5.4 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 method signature.	To be implemented.

5.5 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

5.6 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.

5.7 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 partnerpreferences **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.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.

5.9 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.

5.10 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 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:**
    i. **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:**
    i. **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**
12. **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'
13. **Mandatory: Before final submission run the following command:**

    mvn test
14. **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.**