System Requirements Specification

Index

For

Matrimony Application

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	UserProfile
UserProfile Module Functionalities	
1	Create a user profile
2	Get a user profile by id
3	Get all user profiles
4	Update a user profile by id
5	Delete a user profile by id
6	Search all user profiles by sex **(should be a custom query)**
7	Get all user profiles by likes **(should be a custom query)**

Overall Application	
1	Actuator support needs to be added in the properties file. Expose all actuator endpoints except beans.
2	In application.properties file expose a property “profile.validate.data” with value as “This is default profile”. Create application-qa.properties file (for QA profile) and expose a property “profile.validate.data” with value as “This is qa profile”.
3	Create an endpoint in UserProfileController with following configurations: 1. Method – GET 2. Endpoint - /profile 3. Return – String **The method for this endpoint must read the “profile.validate.data” property file and return its value based on the active profile.**

2 Assumptions, Dependencies, Risks / Constraints

2.1 Userprofile Constraints

- When fetching a user profile by ID, if the user profile ID does not exist, the service method should throw a ResourceNotFoundException with "User profile 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 profile 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 profile not found" message.

2.2 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 both DTO and Entity classes.
- 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.
- All RestEndpoint methods and Exception Handlers must return data wrapped in **ResponseEntity.**

3 Business Validations

3.1 Business Validations - Userprofile

- Name should not be blank.
- Sex should not be blank.
- Phone Number should not be blank.
- Address 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 UserProfile Controller

URL Exposed		Purpose
1. /api/profiles		Creates a new user profile
Http Method	POST	
Parameter	**The user profile data to be created must be received in the controller using @RequestBody.**	
Return	UserProfileDTO	
2. /api/profiles/{id}		Gets a user profile by id
Http Method	GET	
Parameter 1	Long (id)	
Return	UserProfileDTO	
3. /api/profiles		Fetches list of all user profiles
Http Method	GET	
Parameter	-	
Return	List<UserProfileDTO>	
4. /api/profiles/{id}		Updates a user profile by id
Http Method	PUT	
Parameter 1	Long (id) **The user profile data to be updated must be received in the controller using @RequestBody.**	
Return	UserProfileDTO	

URL Exposed		Purpose
5. /api/profiles/{id}		Delete a user profile by id
Http Method	DELETE	
Parameter 1	Long (id)	
Return	-	

URL Exposed		Purpose
6. /api/profiles/search/sex/{sex}		Searches all user profiles by sex
Http Method	GET	

Request Param	String (sex)	
Return	List<UserProfileDTO>	

7. /api/profiles/search/likes/{likesKeyword}		
Http Method	GET	Fetches list of all user profiles by likes
Parameter 1	String (likesKeyword)	
Return	List<UserProfileDTO>	

8. /api/profiles/profile		
Http Method	GET	Fetches the profile
Parameter 1	-	
Return	String	

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**
UserProfileRepository (interface)	● Repository interface exposing CRUD functionality for UserProfile Entity. ● You can go ahead and add any custom methods as per requirements. ● You need to write a function to find all user profiles by likes. ● You need to write a function to find all profiles by sex.	Partially implemented.

5.3 PACKAGE: COM.MATRIMONYAPPLICATION.SERVICE

Resources

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

5.4 PACKAGE: COM.MATRIMONYAPPLICATION.SERVICE.IMPL

Class/Interface	Description	Status
UserProfileServiceImpl (class)	• Implements UserProfileService. • Contains template method implementation. • Need to provide implementation for user profile 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
UserProfileController (Class)	• Controller class to expose all rest-endpoints for user profile related activities. • May also contain local exception handler methods	To be implemented

5.6 PACKAGE: COM.MATRIMONYAPPLICATION.DTO

Resources

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

5.7 PACKAGE: COM.MATRIMONYAPPLICATION.ENTITY

Resources

Class/Interface	Description	Status
UserProfile (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 profile table. ● Generate the id using the IDENTITY strategy	Partially 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.

5.9 PROPERTIES FILES

Resources

Class/Interface	Description	Status
application.properties	● This file is treated as the default properties file for this application. ● You need to write properties to add actuator support. ● You need to write property to expose all endpoints. ● You need to write property to exclude /beans endpoint. ● Add "profile.validate.data" property with value as "This is default profile".	Partially implemented.
application-qa.properties	● This file is treated as the qa properties file for this application. ● You need to write properties to add actuator support. ● You need to write property to expose all endpoints. ● You need to write property to exclude /beans endpoint. ● Add "profile.validate.data" property with value as "This is qa profile".	Partially implemented.

6 METHOD DESCRIPTIONS

6.1 UserProfileServiceImpl Class - Method Descriptions

- Declare a **private final** variable with name **userProfileRepository** of type **UserProfileRepository** interface.

Method	Task	Implementation Details	Return Value
`UserProfileServiceImpl(UserProfileRepository userProfileRepository)`	Constructor for dependency injection	- Annotated with `@Autowired`. - Assigns the repository instance (parameter) to `this.userProfileRepository`.	None(Used internally by Spring for object creation)
`createUserProfile(UserProfileDTO userProfileDTO)`	Create a new user profile	- Convert the DTO to Entity using BeanUtils as: `BeanUtils.copyProperties()`. - Save entity using `userProfileRepository.save()`. - Convert saved entity back to DTO using `BeanUtils.copyProperties()` and return the `UserProfileDTO` object.	Returns: `UserProfileDTO` (Created profile details).
`getUserProfileById(Long id)`	Get user profile by ID	- Fetch profile using `userProfileRepository.findById(id)`. - If present, convert to DTO using `BeanUtils.copyProperties()` and return `UserProfileDTO` object. - Else, return an exception `ResourceNotFoundException` with message: `"User profile not found"`	Returns: `UserProfileDTO` if found. Else: return an exception `ResourceNotFoundException` with message: `"User profile not found"`

`getAllUserProfiles()`	Get list of all user profiles	- Fetch all profiles using `findAll()`. - Convert each entity to DTO using `stream().map()` and using `BeanUtils.copyProperties()` return `UserProfileDTO` object.	Returns: `List<UserProfileDTO>` (List of all user profiles).
`updateUserProfile(Long id, UserProfileDTO userProfileDTO`	Update an existing user profile	- Fetch profile using `findById(id)`. - If found, convert DTO to entity using `BeanUtils.copyProperties()` and save it. - Convert updated entity to DTO and using `BeanUtils.copyProperties()` return `UserProfileDTO` object. - Else, return an exception `ResourceNotFoundException` with message: `"User profile not found"`	Returns: `UserProfileDTO` (Updated profile details). Else: return an exception `ResourceNotFoundException` with message: `"User profile not found"`
`deleteUserProfile(Long id)`	Delete a user profile by ID	- Fetch profile using `findById(id)`. - If found, delete using `deleteById()` and return true. - Else, return an exception `ResourceNotFoundException` with message: `"User profile not found"`	Returns: `boolean` (True if successfully deleted). Else: return an exception `ResourceNotFoundException` with message: `"User profile not found"`
`searchProfilesBySex(String sex)`	Search profiles based on sex	- Fetch profiles using `findAllBySex(sex)`. - Convert each entity to DTO using `stream().map()` and	Returns: `List<UserProfileDTO>` (Profiles filtered by sex).

		using `BeanUtils.copyProperties()` return `List<UserProfileDTO>` list.	
`searchProfilesByLikesContaining(String likesKeyword)`	Search profiles by keyword in "likes" field	- Fetch profiles using `findByLikesContaining(likesKeyword)`. - Convert each entity to DTO using `stream().map()` and using `BeanUtils.copyProperties()` return `List<UserProfileDTO>` list.	Returns: `List<UserProfileDTO>` (Profiles where "likes" field contains the keyword).

6.2 UserProfileController Class

1. UserProfileController Class – Setup and Constructor:
   - **Declare service variable: userProfileService**
     - ➔ Declare a private final field inside the controller.
     - ➔ Used to interact with the service layer methods.

Method	**Task**	**Implementation Details**
`@Autowired private Environment env;`	Declare environment variable	- Annotated with `@Autowired` to access property values defined in `application.properties`. - Used to retrieve custom config values.
`@Autowired public UserProfileController(UserProfileService userProfileService)`	Constructor-based dependency injection	- Annotated with `@Autowired`. - Injects the service dependency through constructor. - Assigns to the `userProfileService` field.

2. UserProfileController Class – Method Descriptions:

Method	**Task**	**Implementation Details**
`createUserProfile`	To create a new user profile	- The request type should be `POST` with URL `/api/profiles`. - The method name should be `createUserProfile` and it should return `ResponseEntity<UserProfileDTO>`. - Use `@Valid @RequestBody` for accepting and validating the `UserProfileDTO` from the request body. - This method should call `userProfileService.createUserProfile(userProfileDTO)`. - It should return the created profile with HTTP status `201 CREATED`.
`getUserProfileById`	To fetch a user profile by ID	- The request type should be `GET` with URL `/api/profiles/{id}`. - The method name should be `getUserProfileById` and it should return `ResponseEntity<UserProfileDTO>`. - Use `@PathVariable` for accepting the ID from the URL. - This method should call `userProfileService.getUserProfileById(id)`. - It should return the profile with HTTP status `200 OK`.
`getAllUserProfiles`	To fetch all user profiles	- The request type should be `GET` with URL `/api/profiles`. - The method name should be `getAllUserProfiles` and it should return `ResponseEntity<List<UserProfileDTO>>`. - No input arguments are required. - This method should call `userProfileService.getAllUserProfiles()`. - It should return the list of all profiles with HTTP status `200 OK`.
`updateUserProfile`	To update an existing user profile	- The request type should be `PUT` with URL `/api/profiles/{id}`.

		- The method name should be `updateUserProfile` and it should return `ResponseEntity<UserProfileDTO>`. - Use `@PathVariable` for the ID and `@Valid @RequestBody` for the DTO input and validation. - This method should call `userProfileService.updateUserProfile(id, userProfileDTO)`. - It should return the updated profile with HTTP status `200 OK`.
`deleteUserProfile`	To delete a user profile by ID	- The request type should be `DELETE` with URL `/api/profiles/{id}`. - The method name should be `deleteUserProfile` and it should return `ResponseEntity<Void>`. - Use `@PathVariable` to accept the ID. - This method should call `userProfileService.deleteUserProfile(id)`. - It should return an empty response with HTTP status `204 NO_CONTENT`.
`searchProfilesBySex`	To search user profiles by sex	- The request type should be `GET` with URL `/api/profiles/search/sex/{sex}`. - The method name should be `searchProfilesBySex` and it should return `ResponseEntity<List<UserProfileDTO>>`. - Use `@PathVariable` to accept the sex value. - This method should call `userProfileService.searchProfilesBySex(sex)`. - It should return the filtered profiles with HTTP status `200 OK`.
`searchProfilesByLikesContaining`	To search user profiles by keyword in the "likes" field	- The request type should be `GET` with URL `/api/profiles/search/likes/{likesKeyword}`. - The method name should be `searchProfilesByLikesContaining` and it should return `ResponseEntity<List<UserProfileDTO>>`. - Use `@PathVariable` to accept the likes keyword.

		- This method should call `userProfileService.searchProfilesByLikesContaining(likesKeyword)`. - It should return the matching profiles with HTTP status `200 OK`.
`getProfile`	To fetch profile message from properties file	- The request type should be `GET` with URL `/api/profiles/profile`. - The method name should be `getProfile` and it should return `ResponseEntity<String>`. - This method does not take any input arguments. - It should call `env.getProperty("profile.validate.data", "This is default profile")`. - It should return the message with HTTP status `200 OK`.

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:**
    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

       NOTE: After typing any of the above commands you might encounter any warnings.

       >> Please note that this warning is expected and can be disregarded. Proceed to the next step.

    c. mysql -u root -p
       i. The last command will ask for password which is 'pass@word1'
13. Mandatory: Before final submission run the following command:
       i. 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.