System Requirements Specification Index

For

Dating Application

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

Table of Contents

Dating APPLICATION
System Requirements Specification

1 Project Abstract

Dating Application is Spring boot RESTful application with MySQL, where it allows Users to find matches, likes and dislikes the profiles based on the interests of the other users.

Following is the requirement specifications:

	Dating Application
Modules	
1	User
2	Interests
3	Match
4	Like
5	Dislike
User Module Functionalities	
1	Can register Itself
2	Can Update Itself
3	Can Delete Itself
4	Can get the User by Id
5	Can Get all the Users
Interests Module Functionalities	
1	User can create Interests
2	User can update Interests
3	User can delete the Interests
4	User can get the interests-by-Interests Id
5	User can get the interests by User id
Match Module Functionalities	
1	User can get all the matches
2	User can create a match
3	User can get Potential matches based on Age, Gender, City and Country
Likes Module Functionalities	
1	User can get all the Likes

2	User add a like
Dislikes Module Functionalities	
1	User can get all the dislikes
2	User can add a dislike

2 Assumptions, Dependencies, Risks / Constraints

2.1 User Constraints:

- While deleting the user, if the user id does not exist then the operation should throw a custom exception.
- While getting the user by id, if user id does not exist then the operation should throw a custom exception.

2.2 Interests Constraints

- While deleting the interests by user, if interest id does not exist then operation should throw custom exception.
- While getting the interests by user, if interest does not exist then operation should throw custom exception.

2.3 Match Constraints

- While matching the profile, if the match id does not exist then the operation should throw a custom exception.

2.4 Like Constraints

- While getting all the likes by user, if user id does not exist then operation should throw custom exception.

2.5 Dislike Constraints

- While getting all the dislikes by user, if user id does not exist then operation should throw custom exception

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 model 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

- User name is not null, min 3 and max 100 characters.
- User age is not null, min age is 18 and max age is 60.
- User email is not null, min 3 and max 100 characters and in proper email format.
- User Phone Number is not null, min 10 and max 10 digits.
- User gender is not null
- User city is not null
- User country is not null
- Interests interestedIn is not null, min 3 and max 100 characters
- Interests notInterestedIn is not null, min 3 and max 100 characters
- Interests hobbies are a List and is not null
- Interests profileUrl is not null
- Interests about is not null, min 3 and max 100 characters

4 Rest Endpoints

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

4.1 UserRestController

URL Exposed	Purpose
1. /users Http Method: POST Parameter 1: UserDto Return: UserDto	Register a User
/users Http Method: PUT Parameter 1: UserDto Return: UserDto	Update a User
/users Http Method: GET Parameter 1: - Return: List<UserDto>	Fetches the list of all registered Users
/users/{userId} Http Method: GET Parameter 1: Long (userId) Return: UserDto	Fetches the details of a user
/users /{userId} Http Method: DELETE Parameter 1: Long (userId) Return: UserDto	Delete a User

4.2 InterestsRestController

URL Exposed	Purpose
/interests Http Method: POST Parameter 1: InterestsDto Return: InterestsDto	User adds an Interest
/interests Http Method: PUT Parameter 1: InterestsDto Return: InterestsDto	User updates an interest

/interests/{interestId} Http Method: GET Parameter 1: Long (interestId) Return: InterestsDto	Fetches details of Interests based on interestId
/interests/by-user-id/{userId} Http Method: GET Parameter 1: Long (userId) Return: List< InterestsDto>	Fetches details of all interests based on the userId
/interests/{interestId} Http Method: DELETE Parameter 1: Long (interestId) Return: Boolean	Deletes the interest based on interestId

4.3 MatchRestController

URL Exposed	Purpose
/match/{userId} Http Method: GET Parameter 1: Long (userId) Return: List<MatchDto>	Fetches all matches based on userId or matchedUserId
/match/{userId} Http Method: POST Parameter 1: Long (userId) Parameter 2: List<Filter> Return: List<UserDto>	Fetches all the users based on the potential matches like based on Age, Gender, City or Country

4.4 LikesRestController

URL Exposed	Purpose
/likes/{userId} Http Method: GET Parameter 1: Long (userId) Return: List<LikeDto>	Fetches list of likes based on userId
/likes Http Method: POST Parameter 1: LikeDto Return: LikeDto	Adds a like to the user profile

4.5 DislikesRestController

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>/dislikes/{userId}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Long (userId)</td></tr><tr><td>Return</td><td>List<DislikeDto></td></tr></table></td><td>Fetches list of dislikes based on userId</td></tr>
<tr><td>/dislikes<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>DislikeDto</td></tr><tr><td>Return</td><td>DislikeDto</td></tr></table></td><td>Adds a dislike to the user profile</td></tr>
</table>

5 Template Code Structure

5.1 Package: com.iiht.training.datingapp

Resources

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

5.2 Package: com.iiht.training.datingapp.entity

Resources

Class/Interface	Description	Status
User (class)	o Annotate this class with proper annotation to declare it as an entity class with **userId** as primary key. o Map this class with **user** table. o Generate the **userId** using **IDENTITY** strategy	Partially implemented.
Interests(class)	o This class is partially implemented. o Annotate this class with proper annotation to declare it as an entity class with **interestId** as primary key. o Map this class with **interests** table. o Generate the **interestId** using the **IDENTITY** strategy	Partially implemented.
Match(class)	o This class is partially implemented. o Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. o Map this class with **match** table. o Generate the **Id** using the **IDENTITY** strategy	Partially implemented.

Like(class)	o This class is partially implemented. o Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. o Map this class with **likes** table. o Generate the **Id** using the **IDENTITY** strategy	Partially implemented.
Dislike(class)	o This class is partially implemented. o Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. o Map this class with **dislikes** table. o Generate the **Id** using the **IDENTITY** strategy	Partially implemented

5.3 PACKAGE: COM.IIHT.TRAINING.DATINGAPP.DTO

Resources

Class/Interface	Description	Status
UserDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.
InterestsDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.

MatchDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.
LikeDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.
DislikeDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attributes of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.

5.4 Package: com.iiht.training.datingapp.filter

Resources

Class/Interface	Description	Status
Filter (class)	Object of this class is supposed to return the filter based on the Age, Gender, City and Country	Already implemented.
FilterUtils (class)	This class is supposed to return the list of users based on the Age, Gender, City and Country	Already implemented.

5.5 Package: com.iiht.training.datingapp.model.exception

Resources

Class/Interface	Description	Status
ExceptionResponse (class)	Object of this class is supposed to be returned in case of exception through exception handlers	Already implemented.

5.6 Package: com.iiht.training.datingapp.repository

Resources

Class/Interface	Description	Status
UserRepository (interface)	1. Repository interface exposing CRUD functionality for **User** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented
InterestsRepository (interface)	1. Repository interface exposing CRUD functionality for **Interests** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented
MatchRepository (interface)	1. Repository interface exposing the endpoints for getting all the matches, save a match and find potential matches functionality for **Match** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented

LikesRepository (interface)	1. Repository interface exposing the endpoints for getting all likes by a user and save likes functionality for **Like** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented
DislikesRepository (interface)	1. Repository interface exposing the endpoints for getting all dislikes by a user and save dislikes functionality for **Dislike** Entity. 2. You can go ahead and add any custom methods as per requirements	

5.7 PACKAGE: COM.IIHT.TRAINING.DATINGAPP.SERVICE

Resources

Class/Interface	Description	Status
UserService (interface)	Interface to expose method signatures for user related functionality. Do not modify, add or delete any method	Already implemented.
InterestsService (interface)	Interface to expose method signatures for User interests related functionality. Do not modify, add or delete any method	Already implemented.

MatchService (interface)	Interface to expose method signatures for Match related functionality. Do not modify, add or delete any method	Already implemented.
LikesService (interface)	Interface to expose method signatures for Likes related functionality. Do not modify, add or delete any method	Already implemented.
DislikesService (interface)	Interface to expose method signatures for dislikes related functionality. Do not modify, add or delete any method	Already implemented.
LocationServiceApi (interface)	Interface to expose method signature for User filter related functionality. Do not modify, add or delete any method	Already implemented.

5.8 Package: com.iiht.training.datingapp.service.impl

Resources

Class/Interface	Description	Status
UserServiceImpl (class)	• Implements **UserService**. Contains template method implementation.	To be implemented.

	• Need to provide implementation for user related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	
InterestsServiceImpl (class)	• Implements **InterestsService**. Contains template method implementation. • Need to provide implementation for user interests related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	To be implemented.
MatchServiceImpl (class)	• Implements **MatchService**. Contains template method implementation. • Need to provide implementation for user matches related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	To be implemented.

LikesServiceImpl (class)	• Implements **LikesService**. Contains template method implementation. • Need to provide implementation for user likes related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	To be implemented.
DislikesServiceImpl (class)	• Implements **DislikesService**. Contains template method implementation. • Need to provide implementation for user disllikes related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	To be implemented.
LocationServiceApiImpl (class)	• Implements **LocationServiceApi**. Contains template method implementation. • Need to provide implementation for user filter related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	Already implemented.

5.9 PACKAGE: COM.IIHT.TRAINING.DATINGAPP.EXCEPTION

Resources

Class/Interface	Description	Status
GlobalHandler (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	Partially implemented.

Class/Interface	Description	Status
UserNotFoundException (Class)	• Custom Exception to be thrown when trying to fetch or delete the info of a user which does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already created.
InterestsNotFoundException (Class)	• Custom Exception to be thrown when trying to fetch or delete interests of the user if interest does not exist. • Need to create Exception Handler for same wherever needed (local or global)	Already created.

MatchNotFoundException (Class)	• Custom Exception to be thrown when a match for the user is not found for a specific user id. • Need to create Exception Handler for same wherever needed (local or global)	Already created.

5.10 Package: com.iiht.training.datingapp.controller

Resources

Class/Interface	Description	Status
UserRestController (Class)	• Controller class to expose all rest-endpoints for user related activities. • May also contain local exception handler methods	To be implemented
InterestsRestController (Class)	• Controller class to expose all rest-endpoints for user interests related activities. • May also contain local exception handler methods	To be implemented
MatchRestController (Class)	• Controller class to expose all rest-endpoints for profile match related activities. • May also contain local exception handler methods	To be implemented
LikesRestController (Class)	• Controller class to expose all rest-endpoints for user-like related activities. • May also contain local exception handler methods	To be implemented

DislikesRestController (Class)	• Controller class to expose all rest-endpoints for user dislike related activities. • May also contain local exception handler methods	To be implemented

6 CONSIDERATIONS

A. There is only one Role of possible value
   1. User
B. The user can perform the following 4 possible actions

Interests
Profile Match
Profile Like
Profile Dislike

7 EXECUTION STEPS TO FOLLOW

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 Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.**
3. **To build your project use command:**
   mvn clean package -Dmaven.test.skip
4. **To launch your application, move into the target folder (cd target). Run the following command to run the application:**
   java -jar DatingApplication-0.0.1-SNAPSHOT.jar
5. **This editor Auto Saves the code**
6. **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.**

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. **This is a web-based application, to run the application on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**

   Note: The application will not run in the local browser
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

13. **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.**