IIHT

Time To Complete: 10 to 12 hr

E-Dating Application

Contents

[1 Problem Statement 2](#_Toc76387085)

[2 Proposed E-Dating Application Wireframe 2](#_Toc76387086)

[3 Tool Chain 3](#_Toc76387087)

[4 Business Requirements: 4](#_Toc76387088)

[5 Proposed Rest Endpoints to be exposed 7](#_Toc76387089)

[5.1 Rest APIs: 7](#_Toc76387090)

[6 Rubrics/Expected Deliverables 7](#_Toc76387091)

[7 Implementation/Functional Requirements 8](#_Toc76387092)

[7.1 Product and Frameworks: 8](#_Toc76387093)

[7.2 Governance and Tooling: 9](#_Toc76387094)

[7.3 Code Quality/Optimizations 9](#_Toc76387095)

[8 Platform 9](#_Toc76387096)

[8.1 Cloud Specific Design 9](#_Toc76387097)

[8.2 Design Specification - 1 9](#_Toc76387098)

[8.3 Design specification - 2 9](#_Toc76387099)

[8.4 Design specification – 3 10](#_Toc76387100)

[9 Methodology 10](#_Toc76387101)

[9.1 Agile 10](#_Toc76387102)

# Problem Statement

**E-Dating Application** is Restful Application. The Main purpose of **E-Dating** is to provide the ability to users to find match, likes and dislikes the profiles based on the interests of the other users.

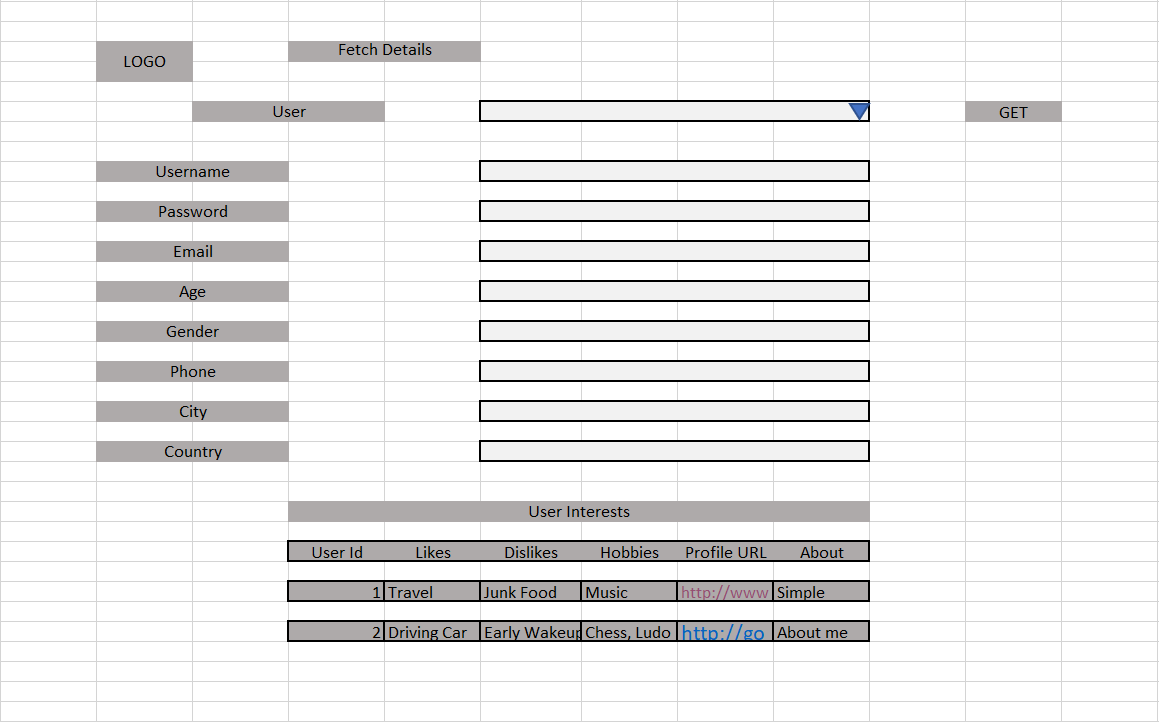
. The core modules of E-Dating app are:

* User can register itself with the application.
* User can add the Interests
* A user can see the interests of other users.
* User can get Potential matches based on Age, Gender, City and Country

The scope includes developing the application using tool chain mentioned below.

# Proposed E-Dating Application Wireframe

Below is the wireframe for reference.



# Tool Chain

|  |  |  |
| --- | --- | --- |
| Competency | Skill | Skill Detail |
| Engineering Mindset | Code Quality |  |
|  |  |  |
| Programming Languages | Application Language | Java |
| Products & Frameworks | Presentation | Angular/React |
|  |  | Javascript/Typescript |
|  |  | Bootstrap |
|  | Compute & Integration | Spring Boot |
|  |  |  |
|  | Governance & Tooling | JUnit |
|  |  | Mockito |
|  |  | Jasmine |
|  |  | JaCoCo |
|  |  | SonarQube |

# Business Requirements:

Below are the user stories for the given problem statement

|  |  |  |  |
| --- | --- | --- | --- |
| **User**  **Story #** | **User Story Name** | **User Story** | **Development** |
| US\_01 | User Functionalities | As a User I can register myself to the application  *While registering, following information is required.*   * Username * Password * email * Age * Gender * Phone number * City * Country   It also requires user interests’ information as follows:  Interest Information   * Likes * Dislikes * Hobbies * Profile URL * About   Constraints:   1. The Username should be unique 2. Hobbies can be one or more than one 3. Profile URL should be a proper URL starting with http or https.   Validations:   1. Username is not null, min 5 and max 30 characters. 2. Password is not null, min 5 and max 30 characters. 3. The age is not null, min age is 18 and max age is 45. 4. email is not null, and it should be valid email pattern, containing a single @. 5. Phone number is not null, min 10 and max 10 character and all must be numeric. | Only API to be developed |
| US\_02 | Interests Functionalities | As a User I can Update my interests  Constraints:   1. If user is trying to be update the interest which does not exists, it must throw a custom exception. 2. A user can update his own interests, if user tries to update other user’s interests, throw a custom exception. | Only API to be developed |
| US\_03 | Match Functionalities | As a User I am able to find the matches based on the Gender, Age, City or Country  *While finding the matches’, following information is required.*   * List of matched profiles   Precondition:   1. User id must be of an existing user available (Can get from Database)   Constraints:   1. If the user is not found with the given user id, it must throw a custom exception. 2. If there is no match found, then it should return that No matches found | Only API to be developed |
| US\_04 | User functionality | As a User I can list all the users with their interests  Considerations:   1. While fetching the above details the API must return the Interests details (Likes, dislikes, hobbies etc) along with the user details. | API and Frontend to be developed |
| US\_05 | Interests functionality | As a user I am able to delete the interest belong to me.  Constraints:   1. If the interest id does not exists, then throw a custom exception | Only API to be developed |

# Proposed Rest Endpoints to be exposed

## Rest APIs:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **URL Exposed** | **Purpose** | | /e-dating/api/v1/users/register-user | Register a new User | | / e-dating/api/v1/interests/update | Fetches details of all bids on product with product details | | / e-dating/api/v1/users/matches/{userId} | Potential matches for a user | |
| |  |  | | --- | --- | | **URL Exposed** | **Purpose** | | /e-edating/api/v1/users/get/all | Get all the users with the interest | | / e-dating/api/v1/users/delete/{interestId} | Delete the Interest | |

# Implementation/Functional Requirements

## Product and Frameworks:

1. **Presentation**

Develop the frontend for User Story 4. (Implementation as follows)

1. Implement using either Angular or React.
2. Implement any one of the Gang of four Patterns to compose data using typescript before presenting the same on UI.
3. Implement at least one approach for UI performance consideration.
4. Identify and Implement client-side Optimization Techniques for Bootstrap.
5. **Compute and Integration**

Develop the backend application as a RESTful Application. (Implementation as follows)

* + 1. Use any one of the Creational Design patterns for composing the model object to be sent back as response on following endpoint:

/ e-dating/api/v1/users/matches/{userId}

(Fetches all the matches for a given user)

* + 1. Optimize you REST endpoints to allow filtering, sorting, and pagination.

## Governance and Tooling:

* + - 1. Follow the practise of Creating Testable Component
      2. Configure your frontend application to implement Unit Testing using Jasmin & Karma.
      3. Configure your backend application to implement UnitTesting using Junit and Mockito.
      4. Test suites must contain exception situation testing.
      5. Generate the Code Coverage report of the same.
      6. Generate the Code Quality Report of the backend application.

## Code Quality/Optimizations

1. Associates should have written clean code that is readable.
2. Associates need to follow SOLID programming principles.
3. Generate the code quality report of the same.