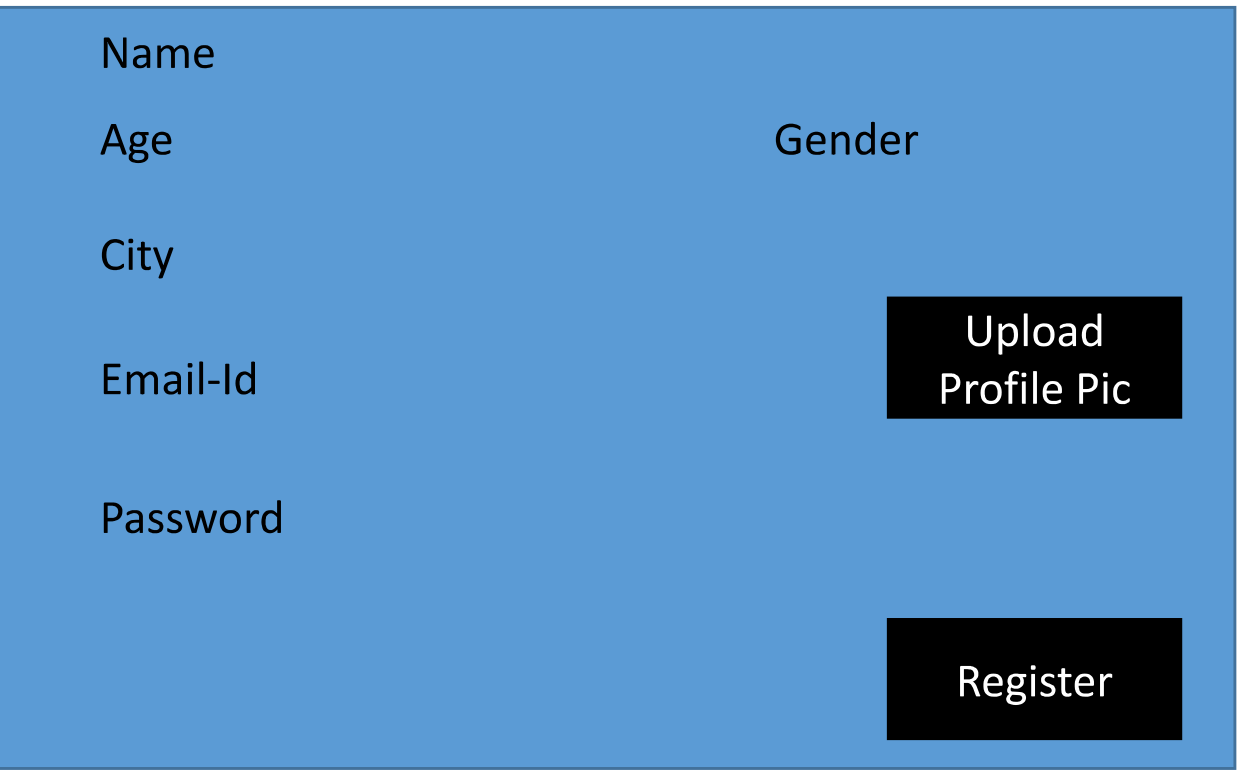
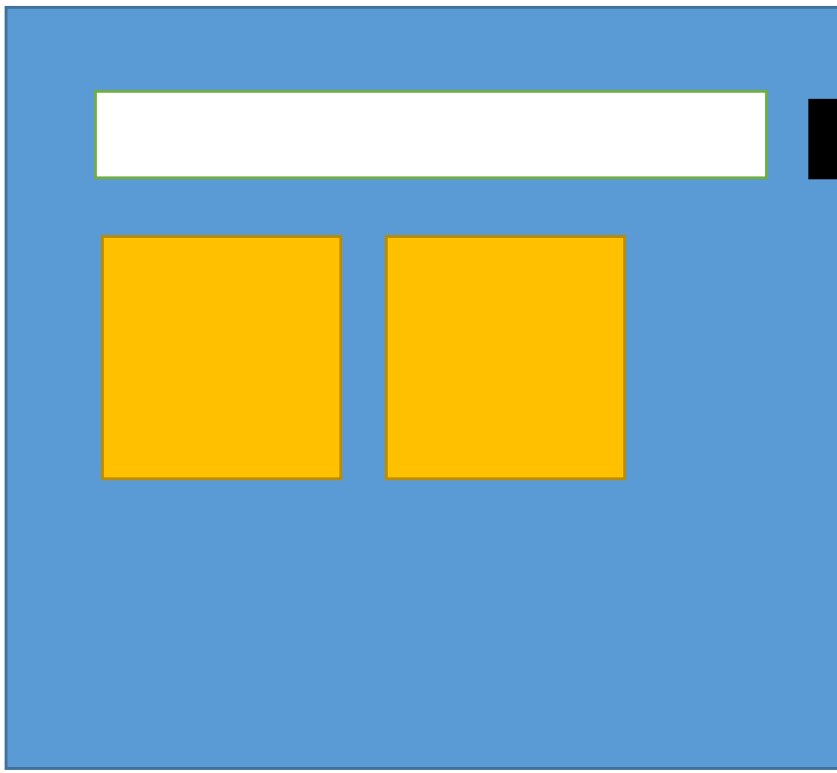
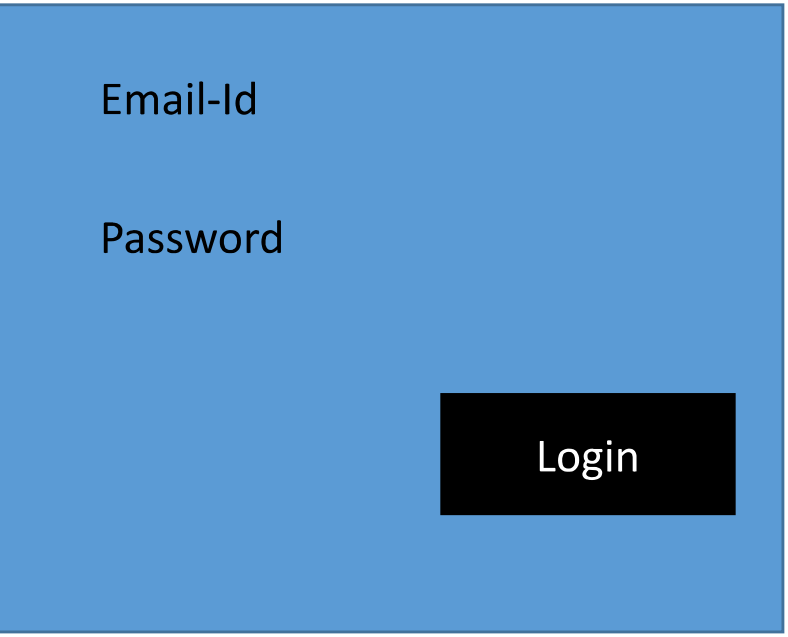
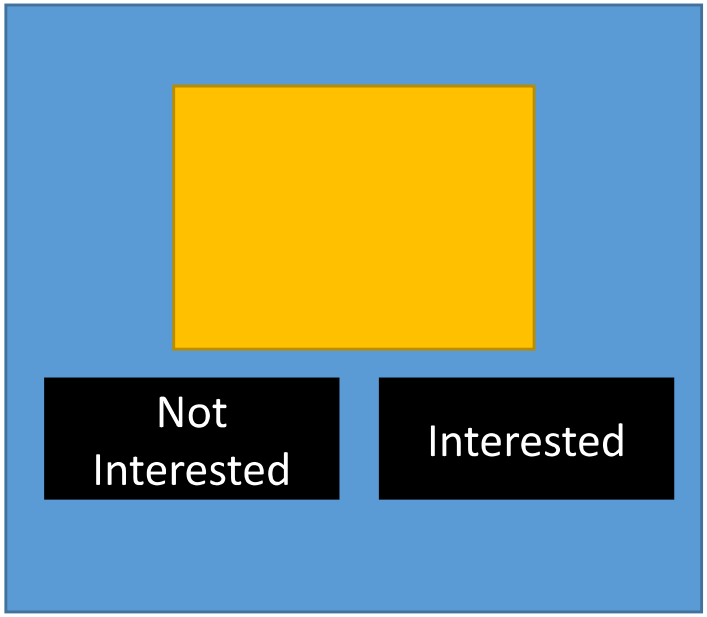
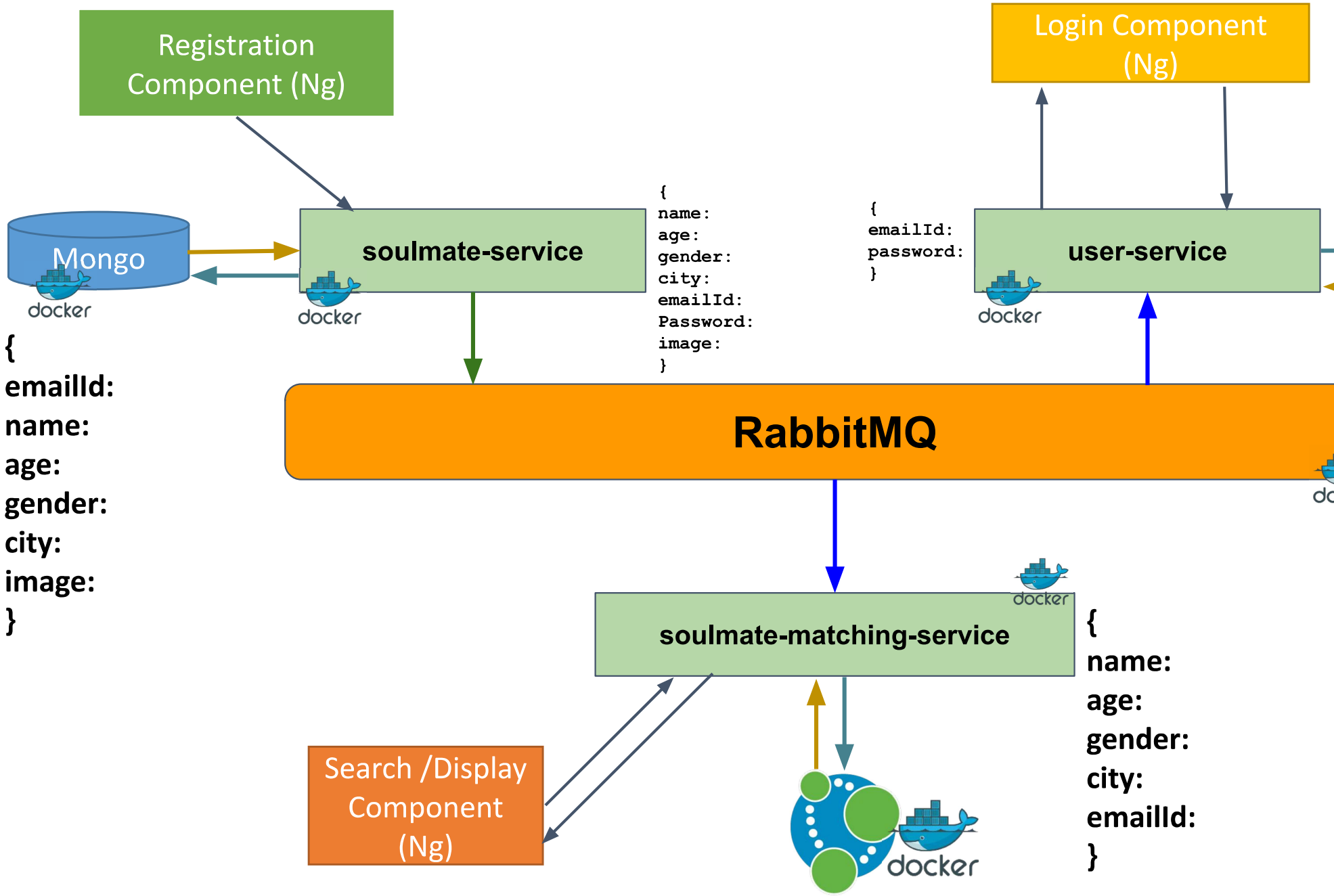

Name
Age
Gender
City
Upload
Profile Pic
Email-Id
Password
Register




Email-Id
Password
Login


Not
Interested
Interested


Registration Component (Ng)
Login Component (Ng)
Mongo
docker
soulmate-service
docker
{
name:
age:
gender:
city:
emailId:
Password:
image:
}
{
emailId:
password:
}
user-service
docker
{
emailId:
name:
age:
gender:
city:
image:
}
RabbitMQ
docker
soulmate-matching-service
{
name:
age:
gender:
city:
emailId:
}
Search /Display Component (Ng)
docker

Requirements:

All layers of the services should have test cases
API Documentation should be generated for the soulmate-service (Swagger/Rest API Doc https://spring.io/projects/spring-restdocs)
Proper exception handling to be performed (Refer video)
Lombok to be used for reducing boilerplate (Refer video)
Logging should be done (Research and use @Sl4J annotation of Lombok to perform logging)
All services, databases, and message bus should be running on Docker. Docker-compose to manage conta
All code should be commented at class level and method level
Ensure continuous Git commits and push with proper commit messages.

Task List

Backend Tasks

Create a Spring multi-module project
Add a user-service module
Add a soulmate-service module
Add a soulmate-mapping-service module
Implement user-service module
Implement soulmate-service module
Implement soulmate-mapping service module
Dockerize the backend services
Use postman to test all the flows

Frontend

Create the Angular app
Create the Registration component and test with the backend
Create the Login component and test with the backend
Create the Soulmate Profile component and test with the backend
Create the Soulmate dashboard component where recommended soulmates are displayed
Dockerize the front end

Use Docker-compose to start the complete application with one command
Demo to Mentor

POC’s

- Elastic Search Database
- Neo4J Database
- Auth Authentication
- RabbitMQ