**Documentation: Group I**

Project Name: Quikr

**Milestones 1:**

1. Login and register to quikr
2. Angular
3. Registration Page: Accepting User name, email id, Phone Number, Password, Confirm password. Followed by client side validation against input requirements, stored in database. Two validations are carried out – one to check if the password and confirm password data match or not and second to check if the input data is according to the requirements set. The user is the redirected to the login page.
4. Creating registration component which contains html, CSS and typescripting.

* Registration.component.html (3.5 hr)
* Registration.component.css
* Registration.component.ts (1 hr)

ii. Client Side validations

User should enter Valid email Id (xyz@gmail.com) and phone number(10 digits number) while registering that will be mentioned in typescript file. Password and confirm password fields should be same. User cannot registered by same emailId twice. (4 hr)

1. Fields

* Username
* Password
* Confirm Password
* Email Id
* Mobile number

1. Login Page: Accepting and validating emailId and Password by checking the user credentials with those stored in the database. If the user exists, he/she is redirected back to the home page as a logged in user and he/she can start buying/selling products.
2. Creating login component which contains html, CSS and typescripting.

* Login.component.html (3 hr)
* Login.component.css
* Login.component.ts( 1 hr)

1. Validations

User should enter Valid username and password which is stored in the database. (2.5 hr)

1. Fields

* emailId
* Password

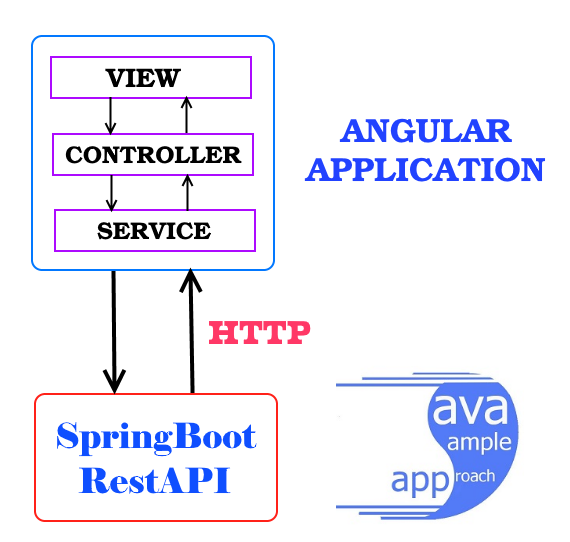
1. Routing:

In Angular, the best practice is to load and configure the router in a separate, top-level module that is dedicated to routing and imported by the root AppModule. The module class name is AppRoutingModule and it belongs in the app-routing.module.ts in the src/app folder.(4 hr)

**Add routes:**

Routes tell the router which view to display when a user clicks a link or pastes a URL into the browser address bar.A typical Angular Route has two properties:

1. path: a string that matches the URL in the browser address bar.
2. component: the component that the router should create when navigating to this route.
3. Business Layer



**Controller Layer**: the controller will handle the navigation between the different views.

1. User Controller
2. CreateUser()

This method will use Save() of mongorepository. The save() method uses either the insert or the update command, which use the default write concern. (1 hr)

1. validateUser ()

This method will call validateUser() of service class. This method will use FindAll() of mongorepository interface and validate user credentials. (2 hr)

1. getAllUser()

This method will use to retrieve all users in database using FindAll () of mongorepository interface. (2 hr)

**Service Layer:**

1. Registration Service
2. CreateUser()

This method will use Save() of mongorepository. The **save**() **method** uses either the insert or the update command, which use the default write concern. (2.5 hr)

1. validateUser ()

This method will call validateUser() of service class. This method will use FindAll() of mongorepository interface and validate user credentials.( 3 hr)

1. getAllUser()

This method will use to retrieve all users in database using FindAll () of mongorepository interface. (3 hr)

1. **Server**

Server side will be spring data along with mongoDB.

1. DAO: Dao will create user collection in database and having methods to create new user and Find one.
2. Save (3 hr)
3. FindOneUser (3 hr)
4. MongoDB: Instead of using [tables](https://whatis.techtarget.com/definition/table) and [rows](https://searchoracle.techtarget.com/definition/row) as in [relational databases](https://searchdatamanagement.techtarget.com/definition/relational-database), the MongoDB architecture is made up of collections and documents. User collection will have user details.(4 hr)
5. Fields in collection:

* Username
* Email Id
* Mobile number
* Password
* Confirm Password

1. **Mockito Testing**

Mockito is a mocking framework, JAVA-based library that is used for effective unit testing of JAVA applications. Mockito is used to mock interfaces so that a dummy functionality can be added to a mock interface that can be used in unit testing.(5 hr)