CASESTUDY NAME

# Instructions to Associates:

1. Write the code where instructed as per the requirement. Try not to modify the existing code.
2. .Make sure to use the components already defined rather than creating the same components once again.
3. Work with the code present in the “public” folder of the package. Do not edit the files in the parent folders, unless instructed to do so.
4. Keep the MongoDB server up and running before beginning to code the application.

**Casestudy Description:**

Due to the high availability of applications in the form of web or mobile applications, customers are scattered all over the world. Identifying where the customer is from and providing a better online support turns out to be the core of CRM.

Challenge? How would my application know where the customer is from? How will the relationship managers connect with a specific customer to assist him further?

The product addresses the above challenges in the most advanced way possible. A Google Maps interface will identify all the online customers. A Customer relationship manager will be able to connect with a specific customer directly. This does really help take the CRM to the next level, and thus, gaining customer confidence.



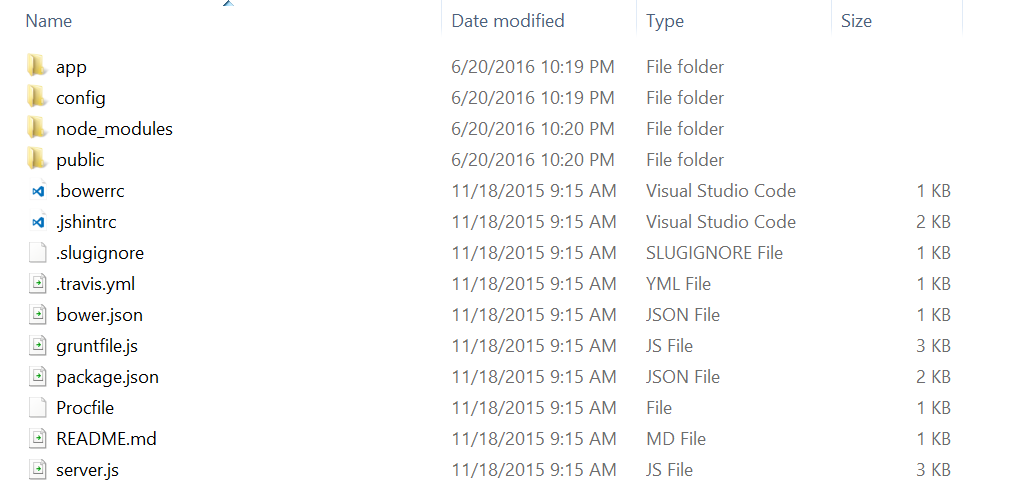
# High level design/Flow Diagram:



***Achieve all the below said requirements using this high level design.***

**Skeleton Code for Development**

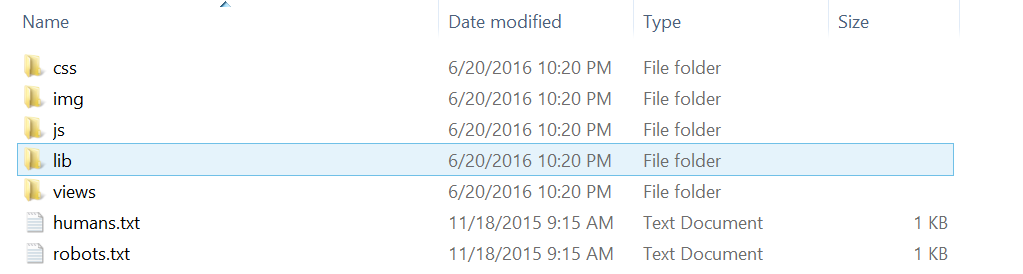
Below is the folder view of the code

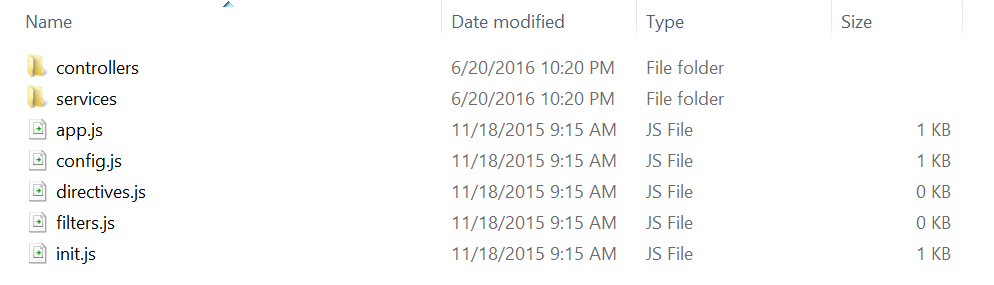


# The following table describes the use of each of the folders shown above:

|  |  |
| --- | --- |
| App | Contains server related code to initialize the application |
| Config | Contains all the configuration files for the application to work |
| Node\_modules | Installed by running “npm install” command on the client machine. This will contain all the dependencies. |
| Public | This folder will contain all the application code like CSS, JS, HTML etc. Controllers, services, directives, filters will also be defined here. |

Public Folder:

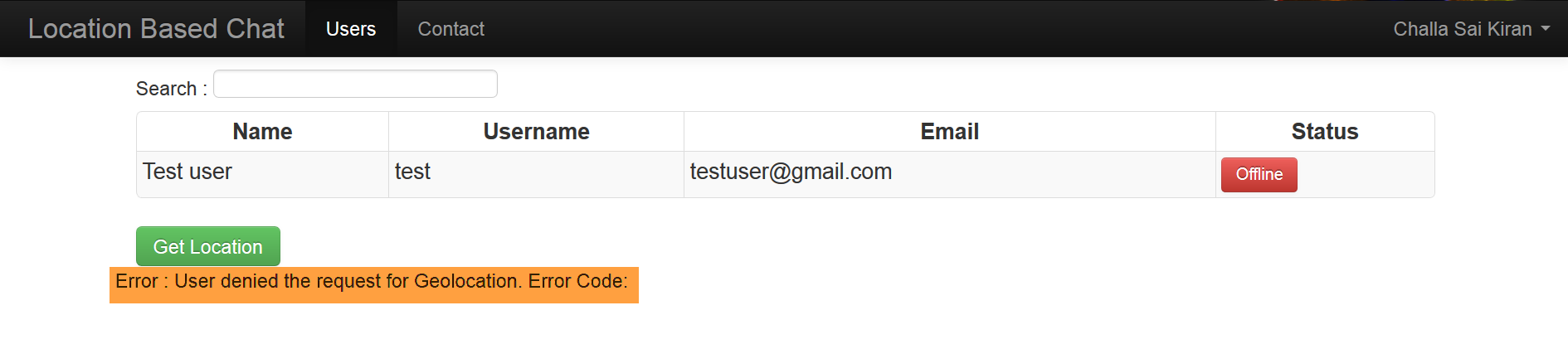




# Requirement 1:

|  |  |
| --- | --- |
| Problem Statement | .When the user sign-in successfully, he/she will be asked to share the location so as to enable other users to identify them. If the user denies sharing the location, an error needs to be show to him by setting error object on scope. |
| Design Consideration | Do not alert messages to the user as it will be a bad design decision. Rather append the error message to a component on the web –page itself. |
| Business Rules & Process | User has to share his location to be able to chat with the other available users. When the user rejects the browser pop-up to never show the location he/she should not be allowed to chat with the other users. |
| Connected code in skeleton | Add the error handling mechanism in the file User.js in the folder “public\js\controllers” on the line number 134. |
|  |  |

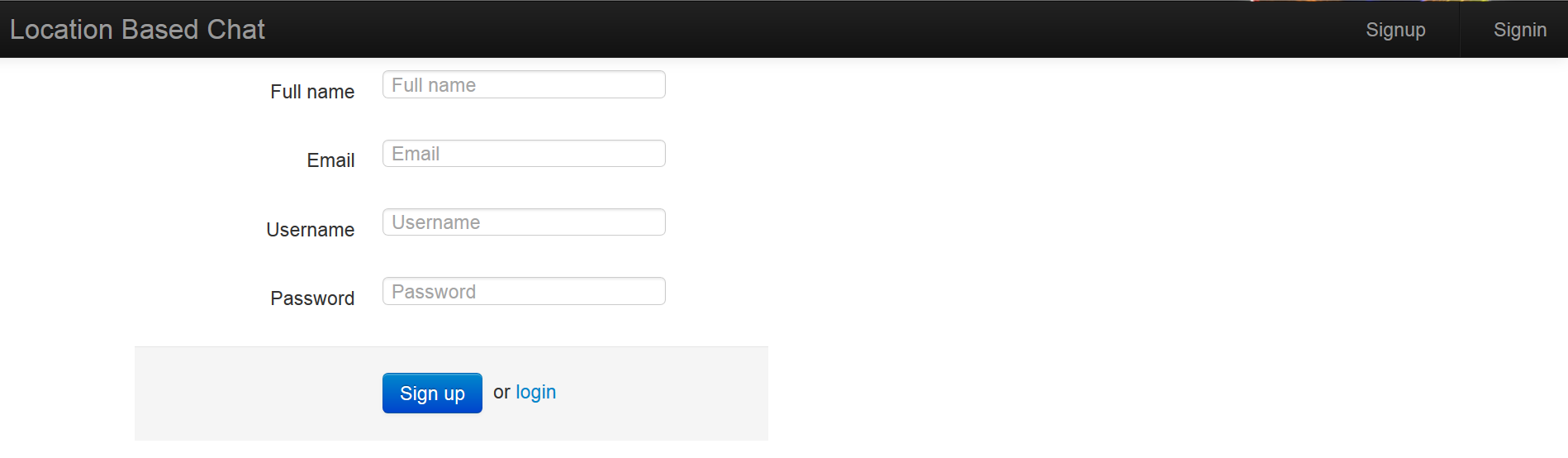
**UI Design:**  Build the following screens.



# Requirement 2:

|  |  |
| --- | --- |
| Problem Statement | .Allow newly entering users to sign-up for using the application. |
| Design Consideration | Design a web-form by using the Bootstrap classes available for vertical forms. Make the UI responsive to changes in device dimension |
| Business Rules & Process | A user will only be allowed to enter the application when he is registered. Anonymous registers should not be allowed to enter the system without registering first. |
| Connected code in skeleton | Create a new HTML page called “signup.html” in the folder “public\views” and add all the required fields. |
|  |  |

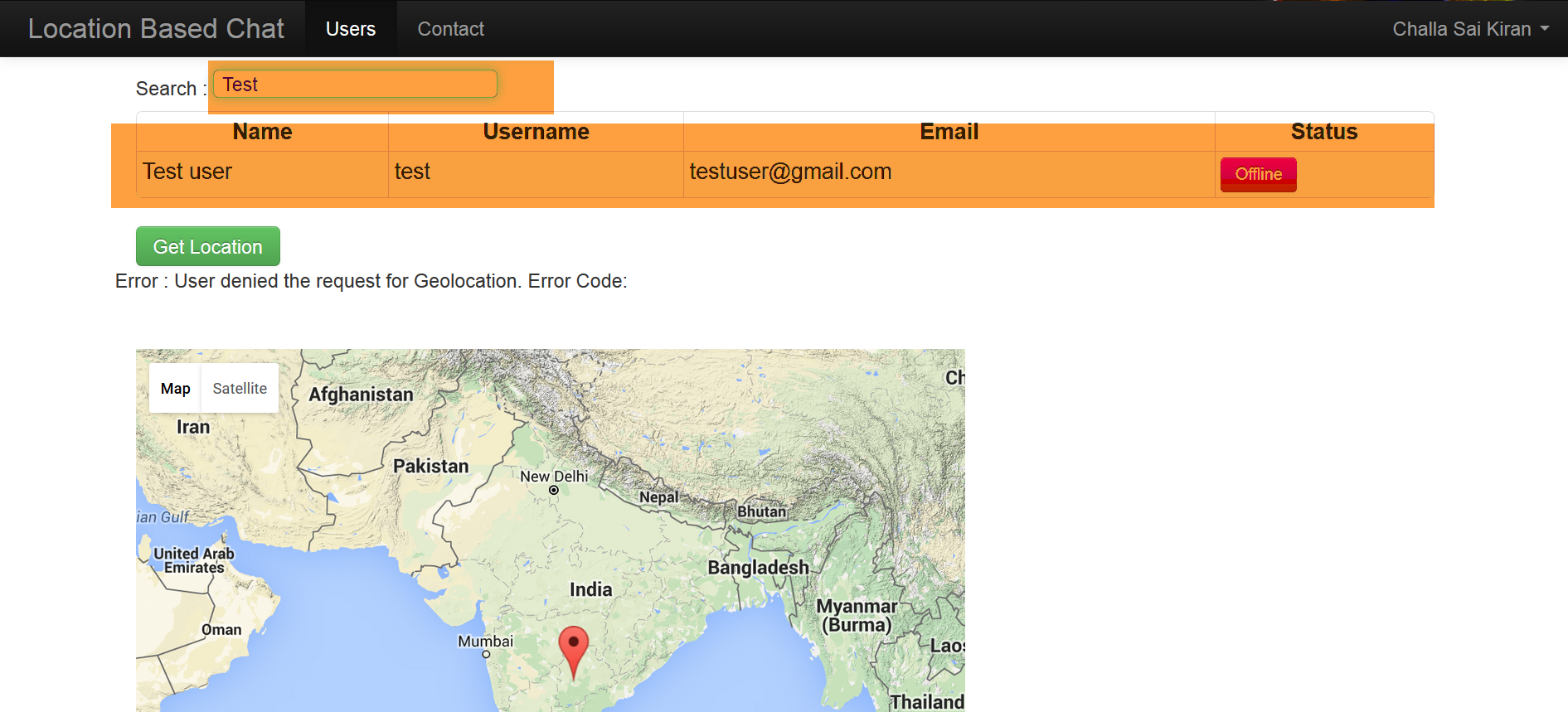
**UI Design:**  Build the following screens.



# Requirement 3:

|  |  |
| --- | --- |
| Problem Statement | .Once the user is successfully logged-in and accepts to share his/her location, the user must be able to view all the users registered with the application. Design a table layout using BootStrap table class to view the available users. Also, to make it easier for the user to search the users, provide him/her a textbox which will be used to narrow down the search results as the user starts typing into the input field. |
| Design Consideration | Make the view look dynamic which will be updated as and when the user starts to fill the input to look for a particular individual in the users table. The search should be performed on the “Name” of the user only. |
| Business Rules & Process | N/A |
| Connected code in skeleton | Create a file called “Users.html” in the folder public\views\User. Add the BootStrap table to display the registered users. Add an input field to the page and use this as an input to the AngularJS Filter. |
|  |  |

**UI Design:**  Build the following screens.



# Requirement 4:

|  |  |
| --- | --- |
| Problem Statement | .Since this is essentially a Chat application which relies on socket communication, the user must be able to send messages to both the online and offline users. If the user is offline the logged-in user must be able to send an offline-message to him/her. To provide this feature, provide the logged-in user a UI which implements BootStrap Modal. This modal will consist of a textbox to enter the message and a send button to send the message. This will also consist of a cancel button to close the Modal. |
| Design Consideration | Make sure to layout the components in the Modal properly. The UI should not be disturbed when viewed on a screen of lesser width. |
| Business Rules & Process | N/A |
| Connected code in skeleton | Add a Modal to the User.html page created in the previous requirement. The modal should be shown when the user clicks on the “Offline” button on the same page. |
|  |  |

**UI Design:**  Build the following screens.

