Emergency Social Network Team SV-7

The system aims to provide the users a social network platform that will come in handy during emergencies. The goal is to implement a system that will enable the users to share their status with small groups of civilians and chat with them either privately or publicly during emergency situations.

Technical Constraints

- Clients connect to the app server via their mobile phone browsers
- No native app necessary - only web stack on mobile browser (Chrome, Safari and Firefox will be supported)
- Product will have a RESTful API - will function with or without UI
- The system will support real-time dynamic updates
- Database for persistent data
- Deployment on cloud platform

High-Level Functional Requirements

There will be three actors - Citizen, Coordinator and Administrator. The Coordinator has the permission to post public announcements while the Administrator has the permission to update the profile information of the Citizens.
As a citizen, a user will be able to join the community by specifying a username and password. At the time of emergency, the citizen can share his/her status. The Citizen will be provided with the options of chatting either publicly (visible to everyone in the community) or privately (with another individual Citizen). The Citizens can also search for any information stored in the system.

Non-Functional Requirements

The following non-functional requirements have been ordered based on their priorities in the system:

- Reliability - our system will have a backup DB that periodically makes a copy of the data every 30 minutes
- Usability - our system can be easily used even by a layman
- Extensibility - our design will make it feasible to add more functionalities without disrupting the original system

Architectural Decisions

- Client server as main architectural style
- HTML, CSS, jQuery, Bootstrap JS, and AngularJS for the UI and front-end functionality
- Server-side node.js for small footprint and performance
- Lightweight MVC on the server side using express framework
- Web-sockets using socket.io allow event-based fast dynamic updates
- Lightweight Mongo DB with small footprint
- Shippable for continuous integration
- Heroku will be used to deploy the project in the cloud platform
- Mocha for Javascript unit and integration testing with Grunt
- ESLint for static analysis of JS code
- Selenium IDE for UI tests

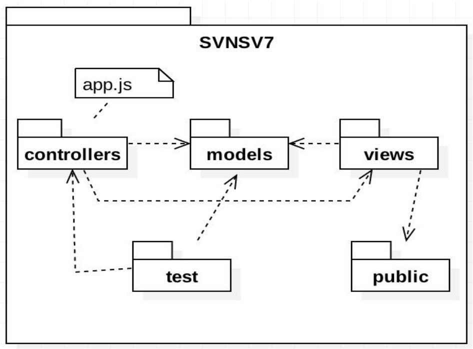
Design Decisions

- Encapsulate data and behavior in models for extensibility, easy testing, and better modularization
- Bridge design pattern to substitute a test database for the production database during testing
- Adapter design pattern to keep interfaces and implementation separate so that it is easier to refactor code while having functionalities intact
- Facade design pattern to show the end users only the high level implementation for them to easily access the functionality while hiding the low level implementation from them

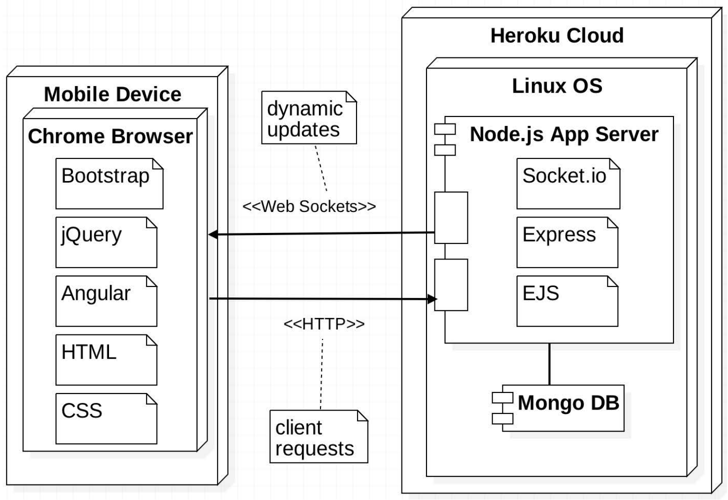
Responsibilities of Main Components

- models: encapsulate data and behavior for entities of the system, serve as the logical core of the system
- controllers: communicate with both the models and the views to update one based on changes in the other
- views: generate the output for the front-end, depends on the models for the information that needs to be represented
- test: corresponds to all the test files such as unit tests for the functionalities
- public: files generating static content as background images, icons, css stylesheets, javascript files
- app.js: the starter file that includes the controllers and starts the application

Code Organization View



Deployment View


Heroku Cloud
Linux OS
Node.js App Server
Socket.io
Express
EJS
Mongo DB
Mobile Device
Chrome Browser
Bootstrap
jQuery
Angular
HTML
CSS
dynamic updates
<<Web Sockets>>
<<HTTP>>
client requests