Emergency Social Network

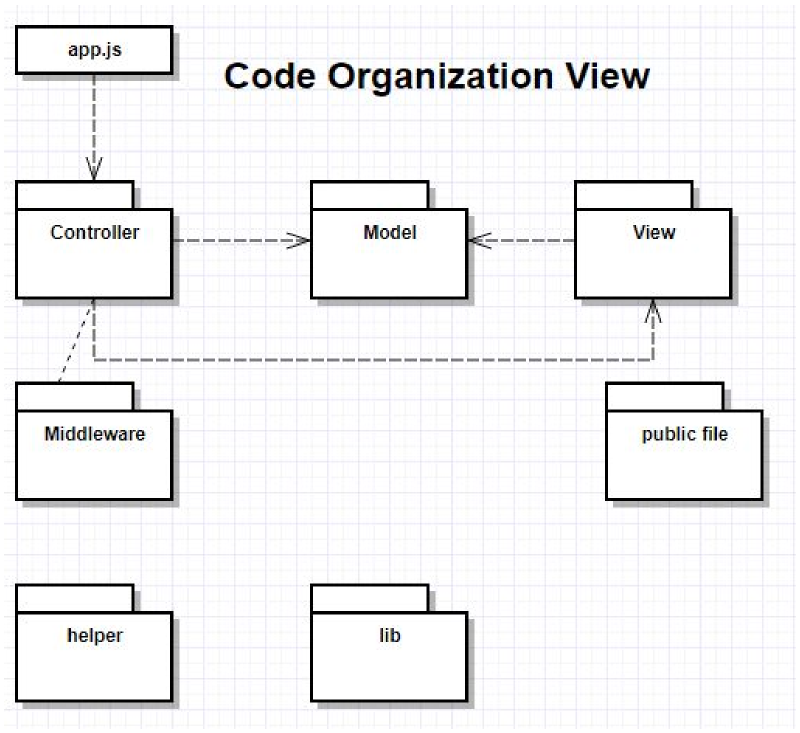
Team SA-04

V1.0

The Emergency Social Network (ESN).The goal is to provide civilians with a social network that they can use during emergency situations. The system is different from other existing social networks because it is specifically designed to effectively support small communities of civilians seriously affected in case of natural disasters like earthquake, tsunami, tornado, wildfire, etc.

Technical Constraints

- App server runs on the cloud server Heroku.
- Clients connect to the app server via their mobile phone browsers. Memory and performance limited by hardware.
- No native app, only web stack (HTML5, CSS, JS) on mobile browser (only Chrome will be supported)
- System has a RESTful API - should function with and without UI
- System supports real-time dynamic updates



High-Level Functional Requirements

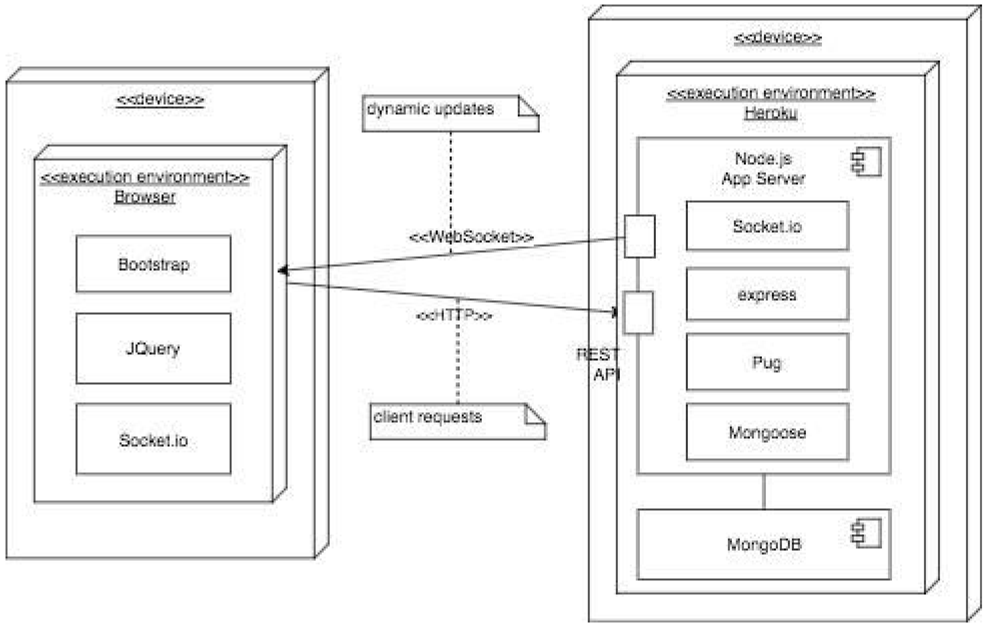
- Users can join the community (reg &login/out)
- Users can chat publicly
- Allow citizen to share status
- Allow citizen to chat privately
- Allow coordinator to post announcement
- Allow the Citizen to search for any information stored in the system
- Allow the Administrator to change the profile information related to a user

Top 3 Non-Functional Requirements

Reusability > Maintainability > Supportability

Architectural Decisions with Rationale

- Client-Server as main architectural style
- Server-side JS (node.js) for small footprint and performance
- Lightweight MVC on the server side using **express** framework
- **Socket.io** allow event-based fast dynamic updates
- RESTful API provides core functionality and reduces coupling between UI and backend
- **MongoDB** as database, and **Mongoose** as ORM library



Design Decisions with Rationale

- Encapsulate data and behavior in models for easy testing and better modularization
- **Adapter** design pattern to substitute a test database for the production database during testing

Responsibilities of Main Components

- **models:** encapsulate data and behavior for entities of the system, main models are Users and Chat Room
- **controllers**: receive user actions, and update the server/DB and client accordingly
- **views**: provide skeleton for rendering
- **middlewares**: control user sessions/cookies
- **app.js** : http interface between server and client
- **public**: static files, like javascript, css, images and so on.