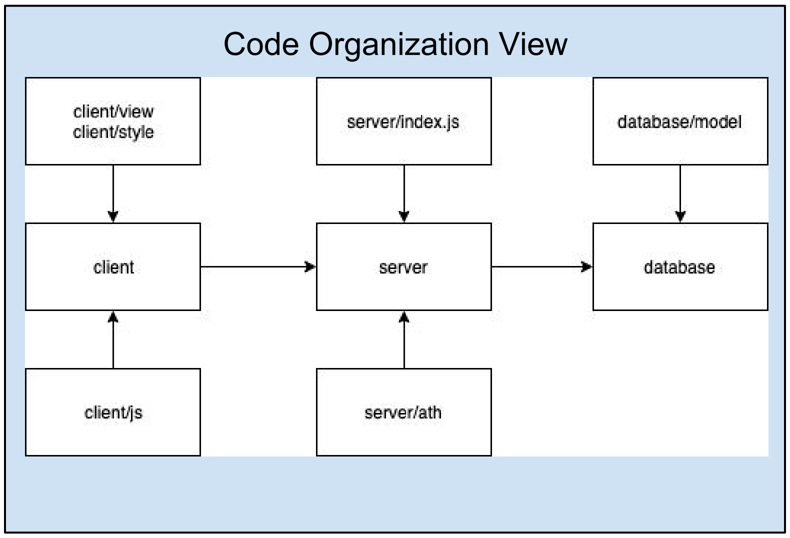
Emergency Social Network — Team SB2

The goal is to provide citizens 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 citizens seriously affected in case of natural disasters like earthquake, tsunami, tornado ,wildfire, etc.

Technical Constraints

- UI and database update dynamically in real time when user states change
- System has a RESTful API - should function with and without UI. Socket.IO is used for data transmission
- Single page website is built to improve user experience
- The solution will be deployed on the Cloud platform Heroku. Website constructed with decoupled tiers
- No native app, only web stack (HTML5, CSS, JS) on mobile browser



High-Level Functional Requirements

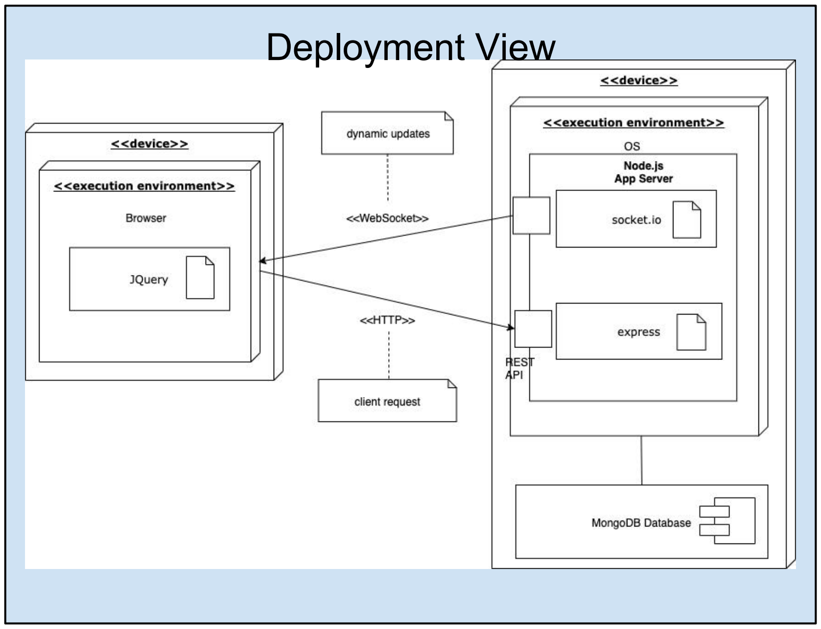
- As a citizen, I can join the community by providing a new username and password. The Citizen is added to the ESN directory. A welcome message is displayed.
- As a citizen, I can login by providing an existing username and password. I can see myself listed in the ESN Directory, along with other citizens. I can also logout.
- As a citizen, I can share my status.
- As a citizen, I can post a message on a public wall (visible to everyone in the community).
- As a citizen, I can chat privately with another citizen.
- As a citizen, I can search for any information stored in the system.
- As a Coordinator, I can post a public announcement.
- As a Administrator, I can change the profile information related to a user.

Top 3 Non-Functional Requirements

Robustness>Security>Performance

Architectural Decisions with Rationale

- Client-Server as main architectural style
- Two decoupled ties for separating front end from back end.
- No framework is allowed.
- HTML/CSS/JS are used for front end and JS with Express.js and MongoDB for back end.
- HTTPS and Socket.IO are used for data transmission.
- Lightweight MVC on the server side using **express** framework
- RESTful API provides core functionality and reduces coupling between UI and back-end
- Web-sockets allow event-based fast dynamic updates
- Lightweight NoSQL DB with small footprint
- Server-side JS (node.js) for small footprint and performance



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
- **Observer** design pattern to update every user when there is an announcement or new message
- **Bridge** design pattern to decouple interfaces of a class from its implementations for future optimization
- **Composite** design pattern to deal with objects uniformly regardless of position in an object hierarchy of front-end design

Responsibilities of Main Components

- **models:** encapsulate data and behavior for entities of the system, main models are user, message, chatroom.
- **controllers**: accept input and convert it to commands for the model or view.
- **views**: display any representation of information such as messages, user profiles, etc.
- **middlewares**: communicate and integrate software between applications and services