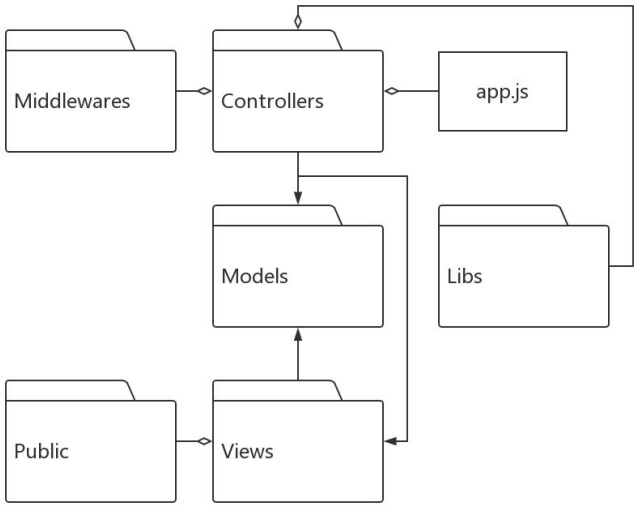
Emergency Social Network-Team SV2 Eleos

Vision

The goal of ESN 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.



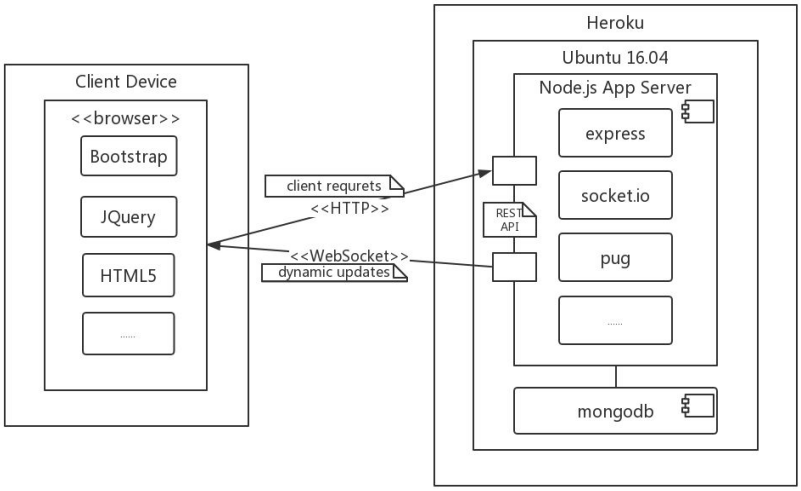
Code Organization View

Technical Constraints

- **HTML**, **CSS**, and **JavaScript** for the UI and front-end functionality;
- **Node.js** with **express.js** and any other supporting JS-based frameworks for the back-end;
- Deploy the solution on a Cloud platform. We will be using **Heroku** for this purpose.
- Clients connect to the app server via their mobile phone browsers.
- System has a RESTful API - should function with and without UI
- System supports real-time dynamic updates

High-Level Functional Requirements

- Join Community: Allows the Citizen to join the community by providing a username and password.
- Chat Publicly: Allows the Citizen to post a message on a public wall (visible to everyone in the community).
- Share Status: Allows the Citizen to share his/her status.
- Chat Privately: Allows Citizen C1 to chat privately with Citizen C2
- Post Announcement: Allows the Coordinator to post a public announcement.



Deployment View

Top 3 Non-Functional Requirements

Extensibility > Usability > Maintainability

Architectural Decisions with Rationale

- Client-Server is the main architecture style.
- Node.js on the server-side.
- 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.
- NoSQL(MongoDB) on the server-side, for fast iteration

Design Decisions with Rationale

- **Well-defined REST API documents**, reviewed before implementation. For better communication and ensure identical common understandings among workmates.
- **Test-driven development**, write test cases before implementation. For revealing potential caveats and reviewing designs.
- Use **Adapter** pattern to wrap up external dependencies, such as Databases and third-party OAuth logins. For reduction of coupling and easier testing.
- Use **Observer** pattern to develop in an event-driven programming paradigm as Node.js recommended.
- Use **Filter** pattern to make cleaner queries for messages and users.
- Use **dependency injection** to avoid heavy coupling and make fine-grained testing possible.

Responsibilities of Main Components

- Models: Encapsulate data and behavior for entities of the system, main models are Account and Message.
- Controllers: Collect data from the client-side using the RESTful API and push updates to all clients simultaneously.
- Views: Present UI on the client side. Current use cases mainly requires a ESN user directory and a public chatting wall.
- Middlewares
    - Passport.js for authentication and encryption
    - Pug.js for HTML render engine
    - Mongoose for object modelling
    - Routing, Error handling
    - JSON/URL encoding parsing

- Zipping/Unzipping and more