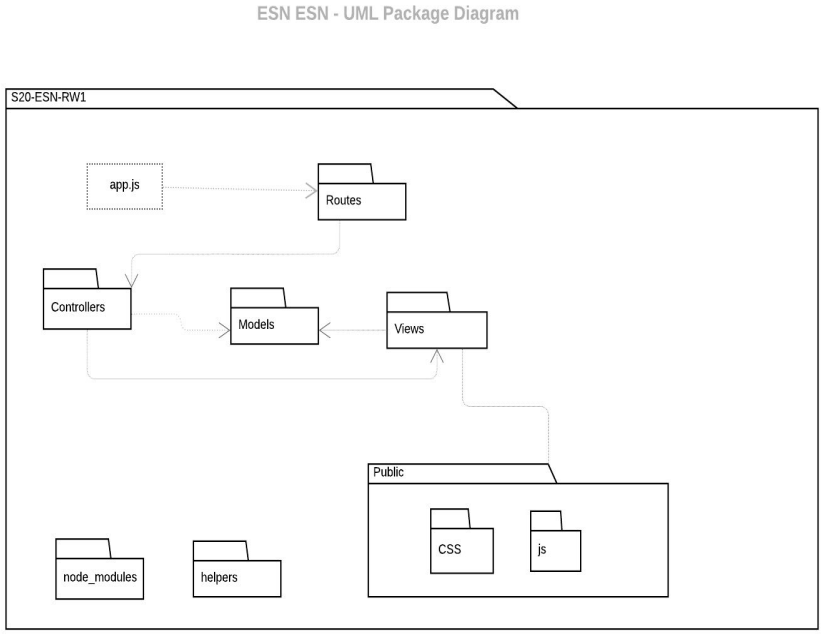
Emergency Social Network

Team Zeroth

To save lives by issuing early warnings about impending calamities and have emergency incidents attended to quickly



Technical Constraints

- App server runs locally
- Clients connect to the app server via their browsers.
- For UI development, HTML5, CSS, JS must be used and the UI should be responsive(PC and Mobile views)
- For back-end, Node.js shall be used with Express.js to handle http requests routing
- System has a RESTful API - should function with and without UI
- System supports real-time dynamic updates

High-Level Functional Requirements

- Citizens shall be able to register into the system
- A citizen should be able to login
- A citizen can sign out of the system
- A citizen will be able join a community using a username and password with additional optional information
- A citizen shall see themselves listed in the directory alongside other citizens
- A citizen will be able to post a message on a public wall (can be seen by everyone in the community)

Top 3 Non-Functional Requirements

Extensibility/Expandability > Usability> Testability> Performance

Architectural Decisions with Rationale

- Client-Server as main architectural style as the server will act as a central point that coordinates and controls clients' communication
- Server-side JS (node.js) for small footprint and performance
- 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 real-time communications or updates
- Lightweight SQL DB with small footprint

Design Decisions with Rationale

Include other important design decisions.

- Encapsulate data and behavior in models for easy testing and better modularization
- **Adapter** design pattern to switch between a test database during testing, development database and production database
- **Observer** design pattern to keep the connected citizens in the community in sync with new updates (realtime behavior)
- **Factory** design pattern facilitate creating a user object based on their username and password
- **Singleton** design pattern to provide a single connection to the database throughout the application
- **Strategy** design pattern shall be used to switch between chat types (Public or Private)

Responsibilities of Main Components

*List and describe the responsibilities of main components of your system that you are using. These should refer to elements that are included in Code Organization and Deployment views **and** deserve some brief explanation.*

- **models:** encapsulate data and behavior for entities of the system. The main models are:
    - Message
    - User
- **controllers**: ChatController and RegistrationController
- **Views**: JoinCommunityForm and ChatForm