Emergency Social Network

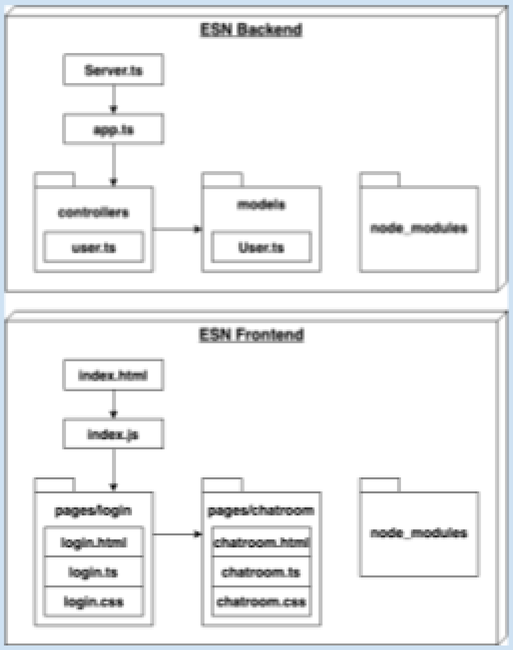
Team SA-03

We are providing an easy-to-use, highly accessible social network for people in emergency situations like natural disasters, terrorist attacks or earthquakes.

Technical Constraints

- Frontend must be written in pure Javascript without any frameworks
- Clients usually use the service on 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

Code Organization View
(a UML package diagram)



High-Level Functional Requirements

- The user will be able to join the community with his/her choice of username, password, whether he/she is a new user or a returning user
- The user will be able to see all other users who join the community and post messages on a public wall where everyone can see

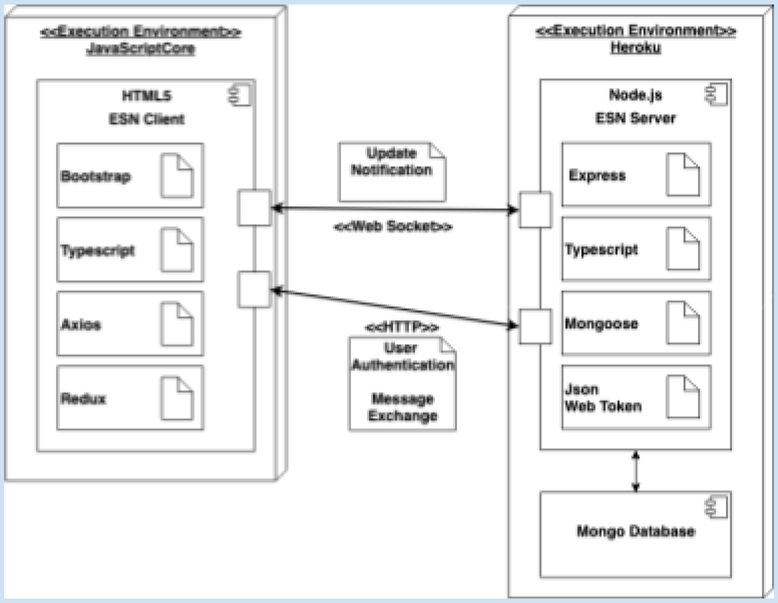
Top 3 Non-Functional Requirements

- Security is important since users care about their privacy.
  1. Do not cache user tokens on the server side.
  2. Enforce HTTPS in the authentication process and message transmission.
- Performance determines whether users like the app and will keep using it.
  3. Offer fluid and fancy login, logout, and chat experience.

Architectural Decisions with Rationale

- Client-Server as main architectural style
- Typescript on both sides to ensure safe RTTI.
- **Bootstrap** on client side for fluid and responsive UI.
- **Redux** on client side to cache client side data.
- Lightweight MVC on the server side using **Express**.
- **Json Web Token** enforces secure credential transmission.
- RESTful APIs for standardized web authentication.
- **WebSockets** allows event-based fast message exchange.
- **MongoDB** with high scalability and small footprint.

Deployment View
(a UML deployment diagram)



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 notify members of a restricted group when a new group's announcement is posted.
- **Bridge** design pattern to make user's different implementations in terms of its roles (Citizen, Coordinator, Administrator).

Responsibilities of Main Components

- **models:** encapsulate data and behavior for entities of the system, main models are User, Message, Announcement
- **controllers**: control the application's workflow and send proper response in terms of request, main models are authController, msgController, announceController.
- **views**: provide user interface, visualized data in the model, main views are Join Community page, Welcome Message page, Administer page.