FSE ESN Project

Team Justice League (SB5)

An application that allows citizens to be able to join an online community and in a disaster situation be able to communicate to others in a public chat. Users are able to post announcements, share their status, send private messages to other users, and report and find missing people. Comes with prepackaged administer account that can update a user's profile.

Technical Constraints

- **Heroku**: The app needs to run on the Heroku Cloud based Application platform.
- **Client Side Software**: No native app, only web application on a mobile browser.

High-Level Functional Requirements

- **Join Community**: citizens can register and login as a user and look at a directory of all other users
- **Chat Publicly**: users can chat with all other users, where messages are dynamically updated
- **Chat Privately**: users can initiate 1-on-1 chats with other users, where messages are dynamically updated
- **Share Status**: users can share their status
- **Post Announcements**: administrators can post announcements that are broadcasted to all users
- **Find Missing People**: users can report missing people that other users can find, notifying the reporter about it.
- **Administer User Profile**: administers can change profile information of other users

Top 3 Non-Functional Requirements

- **Security**: ensure that password is never sent unencrypted between client and server. Password is also decrypted and then reencrypted using another algorithm.
- **Maintainability**: code is modular with low coupling and high cohesion.
- **Reusability**: code is reusable for future updates.

Architectural Decisions with Rationale

- Client-Server as main architectual style
- Node.js Server: event-based, nonblocking asynchronous I/O
- Lightweight MVC on server side via **express** framework
- RESTful API for core functionality to reduce coupling between UI and backend
- Event-based fast dynamic updates via web-sockets for chat

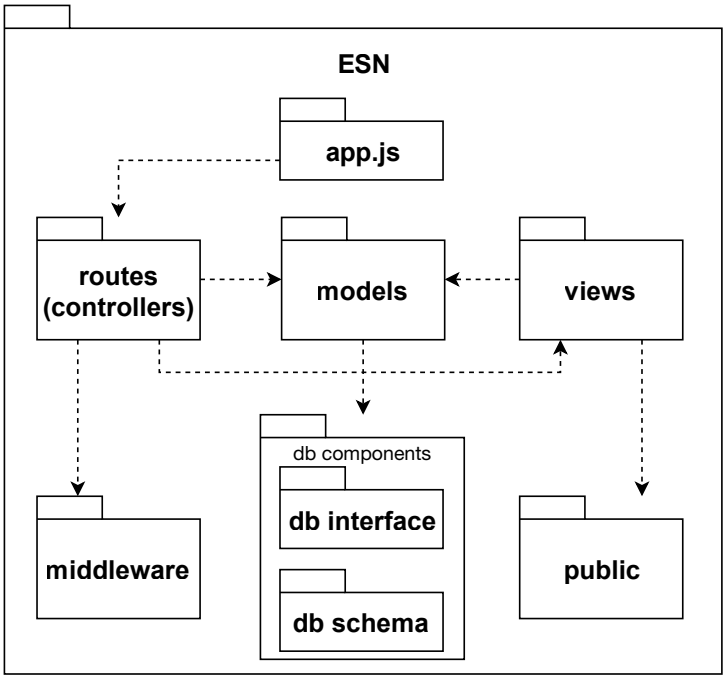
Design Decisions with Rationale

- Encapsulation of data and behavior in classes for easy testing and modularization
- Uses **Bridge** design pattern to decouple DB interface from its implementation
- Uses **Observer** design pattern for dynamic message and notification updates to all users

Responsibilities of Main Components

- Custom **Bootstrap**: responsive design, clean, scalable UI layout with a custom theme
- **pug**: template engine for rendering UI views
- **Mongoose**: provide relational modeling for DB
- **socket.io**: dynamic updates from server to client
- **express-session**: sessions through cookies with express
- **crypto-js**: front and back end cryptography library

Code Organization View



- **app.js**: main file of app, like a master controller
- **routes**: controller files that call models and views
- **models**: models such as users, messages, announcements, etc.
- **views**: front end pug files that are rendered to the user
- **middleware**: mongoose, crypto-js, express, pug, socket.io
- **db interface**: interface for models with MongoDb
- **db schema**: mongoose models to sqlize MongoDb
- **public**: frontend resources: css, js, and images

Deployment View