FSE Iteration 1

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 and send a private message to other users.

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 (OK, Help, Emergency)
- **Post Announcements**: users can post announcements that are broadcast to all users

Top 3 Non-Functional Requirements

- **Security**: ensure that password is never sent unencrypted between client and server
- **Maintainability**: code is modular with low coupling and high cohesion
- **Reusablility**: 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

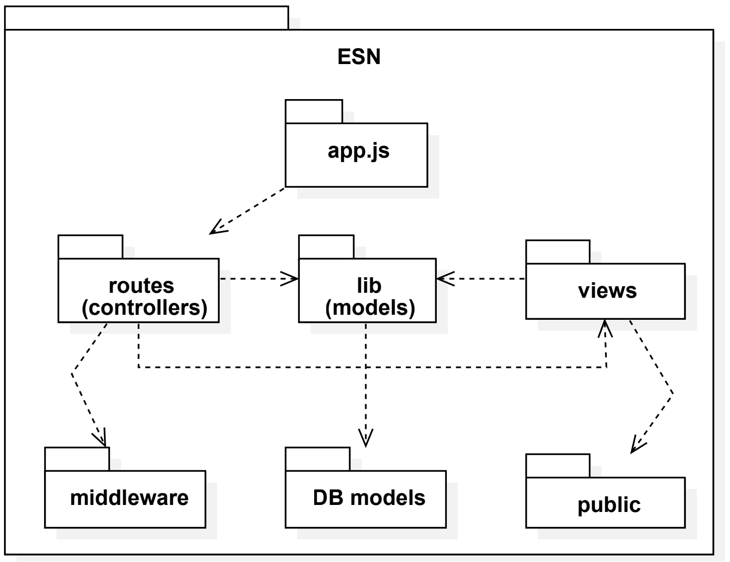
Design Decisions with Rationale

- Encaspsulate data and behvaior in classes for easy testing and better modularization
- Use **Bridge** design pattern for decoupling DB interface from its implementation
- Use **Observer** design pattern for dynamic message and notification updates to all users

Responsibilities of Main Components

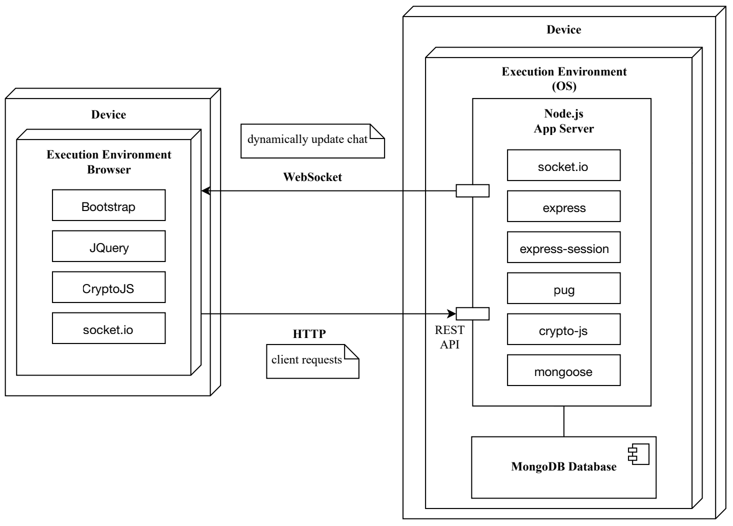
- Custom **Bootstrap**: responsive design, clean, scalable UI layout with a custom theme
- **pug**: template engine for rendering UI views
- **Mongoose**: provide relational modeling for DB
- **socket.io**: dynamic updates from server to client, clients' views are automatically updated when new messages are posted

Code Organization View



- **app.js**: main file of app, like a master controller
- **routes**: controller files that call models (in lib) and views
- **lib**: models (classes) of users, messages, and database adapter
- **views**: front end pug files that are rendered to user
- **middleware**: mongoose, crypto-js, expree-session, express, pug, socket.io
- **DB models**: mongoose models for MongoDb
- **public**: frontend resources: css, js, and images

Deployment View



Device
Execution Environment Browser
Bootstrap
JQuery
CryptoJS
socket.io
dynamically update chat
WebSocket
HTTP
client requests
REST API
Device
Execution Environment (OS)
Node.js App Server
socket.io
express
express-session
pug
crypto-js
mongoose
MongoDB Database