Survivable Social Network on a Chip — Team Alpha-Axes

The system is a lightweight application aimed at providing citizens the opportunity to communicate with one another - from friends to relatives independently of communication infrastructure. It leverages on mobile phone browser and a Beaglebone server ensuring connectivity in the event of a natural disaster (where it possible for communication to fail) to enable prompt evacuation of people

Technical Constraints

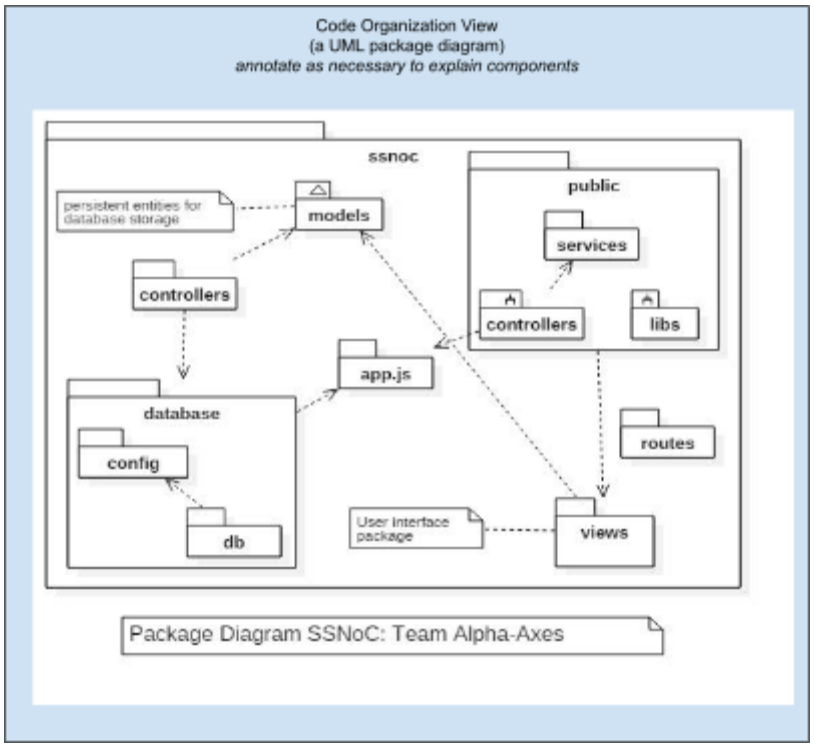
- **Hardware**: App server runs on a Beaglebone Black with wireless dongle and powered by a rechargeable battery. Clients connect to the app server via their mobile phone browsers. Memory and performance limited by hardware.
- **Client Side Software**: no native app, only web stack (HTML5, CSS, JS) on mobile browser (initially only Google Chrome will be supported)

High-Level Functional Requirements

- Citizen is able to login, logout and sign-up
- Citizen is able to share his/her status
- Citizen is able to post a public message
- Citizen is able to search for information
- Citizen is able to share his/her location

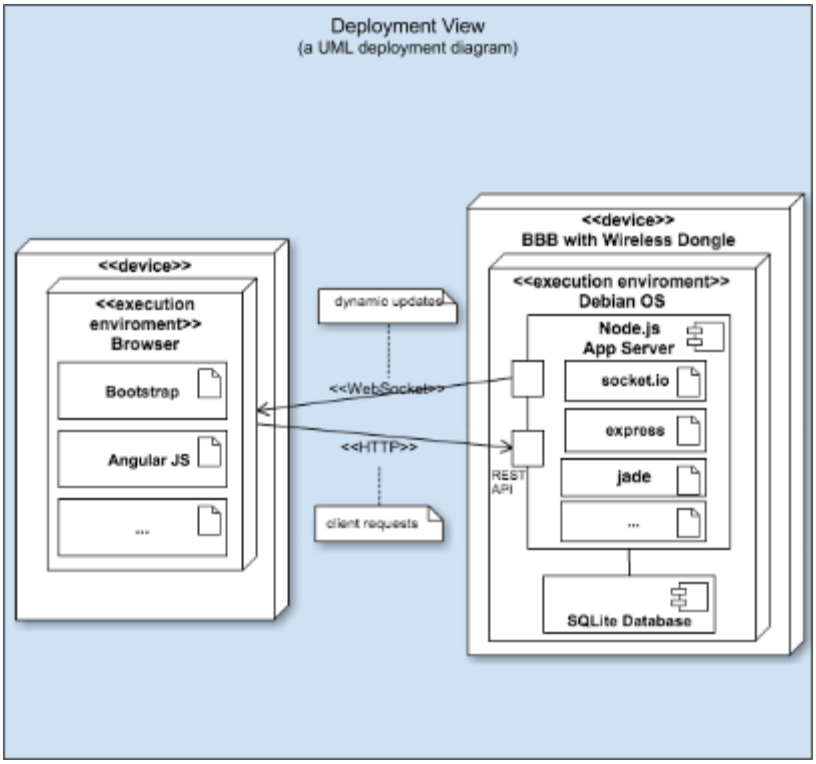
Top 3 Non-Functional Requirements

- Usability: All kinds of citizens; tech-savvy or not must be able to use the application
- Cross-platform portability: Application must be able to perform with same consistency across different devices
- Energy awareness: Application is aware of the limited energy in the server. So most computing lies in client side.



Architectural Decisions with Rationale

- Model-View-Controller as main architectural style
- Server-side JS (node.js) for low footprint and reasonable performance (event-based, non-blocking asynchronous I/O, easily configurable pipe-and-filter for processing incoming requests via middleware)
- Lightweight MVC on the server side via the **express.js** framework
- RESTful API for core functionality to reduce coupling between UI and back-end
- Event-based fast dynamic updates via web-sockets
- Bootstrap CSS framework for responsiveness across different devices



Design Decisions with Rationale

- Use **Facade** to hide implementation details from client classes in the application
- Use **Adapter** design pattern is used to be able to substitute a test database for the production database during testing
- Use **Singleton** for Socket IO and Angular services used throughout the application

Responsibilities of Main Components

- **Socket.io:** dynamic updates from server to client, clients' views are automatically updated when new messages are post or when new new users login
- **Bootstrap**: responsive design, clean, scalable UI layout
- **Jade**: template engine for rendering UI views
- **SQLite**: light-weight DB (alternative: a No-SQL db that works on BBB, e.g., check **tingodb**, which works with **mongoose**)
- **Angular JS**: organize and modularize client-side code and provide dynamic rendering of client-size code.