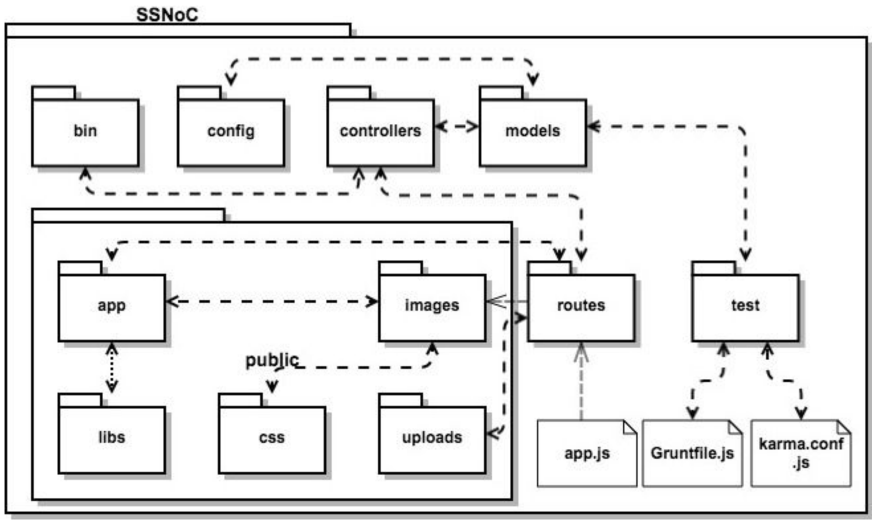
# Survivable Social Network on a Chip — Team SV-7 Warriors

**Vision**: The system allows users to communicate with other users in case of an emergency.

### 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**: Web package: HTML5, CSS, JS;  Mobile browser: Google Chrome. BBB can not support MongoDB.



### High-Level Functional Requirements

- Users can join the community with name and password
- Users can chat publicly
- Users can chat privately with other users
- Users can share their status
- Users can search information stored in the system
- Monitor can do the measure performance
- Coordinator can post a public announcement
- Administer can change a user profile

### Top 3 Non-Functional Requirements

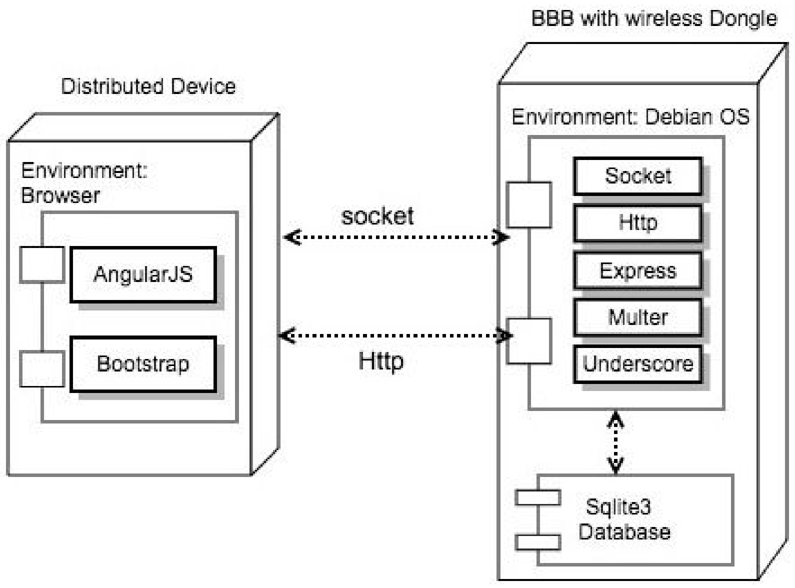
*1. Energy Consumption: Adopt a thick client thin server style, control logic mainly locates in the client side to save the energy of the server.*

*2. Maintainability:codes should be maintainable so that we can easily extend their usage by inheritance or delegation*

*3. Reliability: The system should be able to work without bugs to support the communication in case of emergency*

### Architectural Decisions with Rationale

- Client-Server 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** framework
- RESTful API for core functionality to reduce coupling between UI and back-end
- Event-based fast dynamic updates via web-sockets



### Design Decisions with Rationale

- Encapsulate data and behavior in models for easy testing and better modularization
- **Singleton** design pattern is used to define the global variable and models in the project
- **Observer** design pattern is used to notify all the other users about the incoming message

### 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
- **SQLite**: lightweight DB
- **AngularJS**: organize and modularize client-side code