Survivable Social Network on a Chip Team Pittsburgh TeamA1

An application based on an on-board chip has been developed for people to communicate with other user when normal communication mode is invalid through their PCs or mobile phones.

Technical Constraints

- **Hardware**: Sever and database run on a Beaglebone Black. Wireless coverage of WiFi network is limited. Security is not guaranteed. Memory and performance limited by hardware.
- **Client Side Software**: no native app, only web stack (HTML5, CSS, JS) on mobile browser
- **Database Side Software**: should work well with multi-threads.

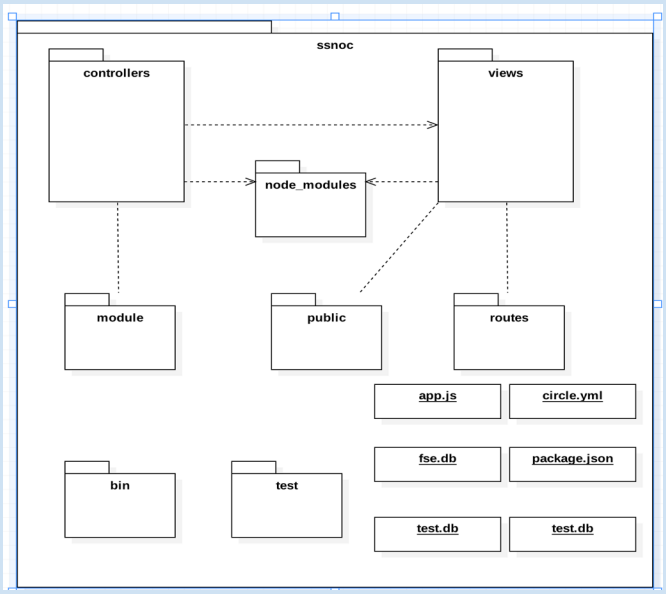
High-Level Functional Requirements

- A common user can register for SSNoC network to join the community, share their status, search information they need, check other's location and communicate with others.
- Administer can post announcement, measure the performance of system and mark the map.

Top 3 Non-Functional Requirements

1. **Reliability**: The system should first work stably
2. **Robustness**: The system should have ability to deal with noises, because the application is based on a on-board chip and environment changes from time to time.
3. **Maintainability**: The system should easy to manger and maintain for testing and further improvement.

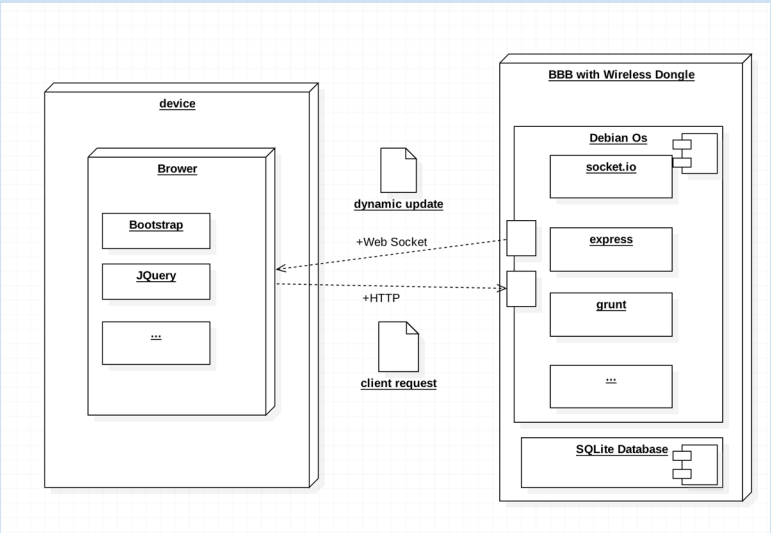
Code Organization View
(a UML package diagram)



Architectural Decisions with Rationale

- Client-Server as main architectural style
- Node.js on the server side for its characteristics (event-based, non-blocking asynchronous I/O, easily configurable pipe-and-filter for processing incoming requests via middleware)
- SQLite database for embedded system.
- Express framework to deploy a lightweight MVC structure.
- RESTful API to reduce coupling between subsystems and increase cohesion within subsystems.
- Event-based web update via jQuery.
- Socket.io for dynamic event-based update.

Deployment View
(a UML deployment diagram)



Design Decisions with Rationale

- Development **Strategy** pattern to encapsulate data and behavior in models
- **Encapsulate** the subsystem with applying a simply User Interface.
- **Singleton** pattern has been applied in the system design as only one instances for per model in the system.
- Remove the coupling between different subsystems by apply **Bridge** patterns
- **Composite** pattern has been used when designed User Interface.

Responsibilities of Main Components

- **socket.io:** dynamic updates from server to client, clients communicates with server with methods of socket.io.
- **Bootstrap**: responsive design, clean, scalable UI layout
- **jQuery**: light-weight JavaScript library, free and open source, easy to deal with HTML event.
- **SQLite**: light-weight DB, ACID database manage system, designed for embedded system, saves the information and message of users.
- **Express**: simple and flexible framework based on Node.js, provides powerful API for web application.