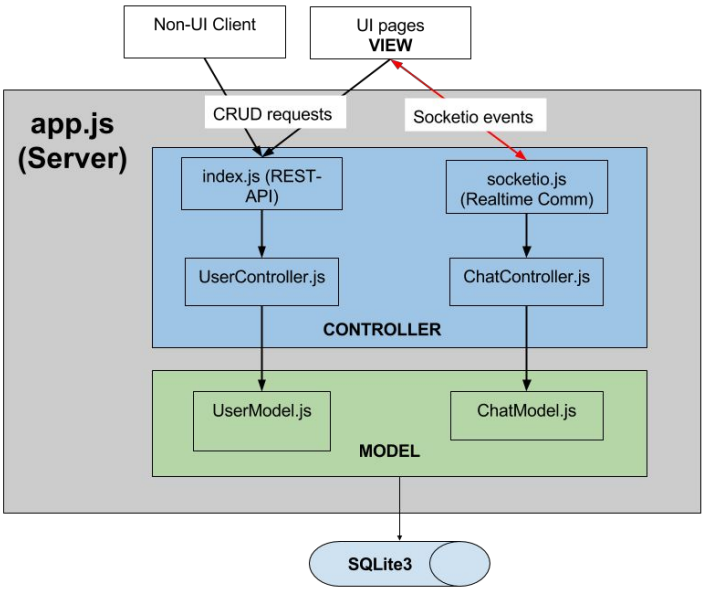
Survivable Social Network on a Chip — Team SA-2 (Social Coders)

A platform for information exchange when serious emergencies happen and for prioritizing the needs for assistance.

Technical Constraints

- The BBB has low computing ability. Battery may be consumed quickly when there is no other available power supplies.
- The application has to run on Debian Linux.
- The site need to work on the BBB system-on-a-chip board with , a Micro SD card and a USB wireless dongle.
- Technologies: Node.js with Express and socket.io



High-Level Functional Requirements

- Join the community and chat with each other (both publicly and privately)
- Share status, and post announcements (not for all users)
- A detailed dashboard for the admin, and an advanced search feature for all the users.

Top 3 Non-Functional Requirements

Performance > Easy access > Usability

- Build a self-monitoring system to measure performance and memory *(Why? health-check for the app)*
- To make the UI render perfectly for most user-agents. *(Why? Even though it's a web-app, most users would use it from hand-held devices)*
- The application should be easy to use without documentation.

Architectural Styles/Patterns with Rationale

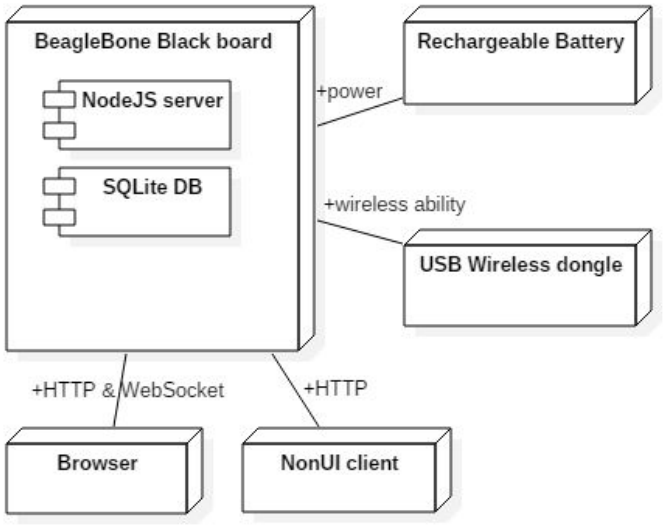
- Repository: server encapsulates all shared data and data services.
- Event-based: Used both in server and client side (duplex communication)
- REST: Part of the server (makes it scalable/reusable/readable)
- MVC: All tiers of the project (low coupling, high coherence)

Design Patterns with Rationale

- Adapter: to provide an abstraction of database so we can easily change the underlying structure without modify other codes.
- Façade: to provide a single portal for the user to easily access the whole system
- Observer: to provide real-time communication

Other Design/Architectural Decisions

- Socket.io: to achieve real-time update of the UI
- Bootstrap : to achieve responsive amongst all user-agents
- jQuery & AJAX: to update partial of the UI dynamically
- Jade: to facilitate rendering the views and make them reusable
- SQLite 3: to provide a lightweight database



Responsibilities of Main Components

List and describe the responsibilities of main components of your system that you are using. These should refer to elements that are included in Deployment and Code Organization views.

- The controller need to handle the main logical function of the system and interact with database.
- The REST API has to be robust enough to work with non-ui clients. A simple machine client should be able to use the basic features of the app.
- The UI pages are an interface for users to access this application. It should be clear and usable without explanation.
- The models provide interfaces to access the database. It should hide implementation details such as the database type the models use, so any components rely on the models won't be affected when we change the database type.