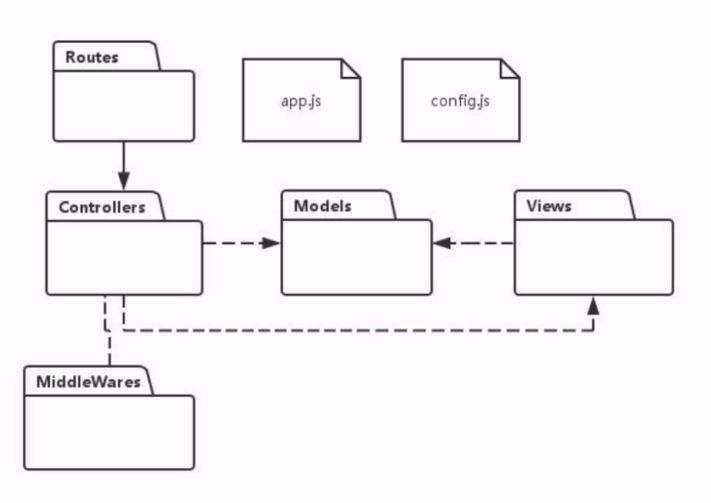
Survivable Social Network on a Chip Team S-16 A2

A social network that easily can be setup in a local area to allow for people to talk to each other without larger internet access



Hardware: Beaglebone Black with wireless dongle and portable rechargeable battery. Clinet connection devices

Software: The main server is written in NodeJS with a sqlite database. The front end uses a combination of framework7. The communication between the backend frontend is implemented with http restful get/put requests and the socketIO framework.

Software Abstraction: To simplify the system a model view controller abstraction is used. Each functional page has its own controller on both the front and back end. The functional pages are:
Login handles new user creation and returning user login

- *Search* handles the look up of information that is on the system such as past posts or user active users
- *Test* handles load testing and performance monitoring of the system
- *Public Message* handles user send public messages to everyone
- *Private Message* Handles users send a private message to a specific user
- *Announcement* handles sending announcements that immediately show up on all users screens
- *Map* Handles uses viewing and recording position information
- *History* Handles the user history and history viewing
- *Profile* Handles the users profile and profile viewing

The SNOC is designed to be very responsive and handle a moderate scale, however it has not been designed to support a massive user base. This allows for us to optimize performance for a smaller number of people so direct socket connections are used for server client message passing of dynamic data such as new message.

Architectural Decisions with Rationale

- Server-side JS (node.js) for low footprint and reasonable performance (event-based, non-blocking asynchronous I/O, easily configurable pipe-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
- The CRUD operations on DB are done in an OO way through a ORM library.
- Single Page Application: most requests are sent through Ajax. Update the content on screen without reloading the whole page.
- MVVM pattern on the front end side. Angular.js supports two way data binding

Design Decisions with Rationale

- ORM(Proxy): Design and implement the models in an OO way and map models to db tables
- Use Adapter design pattern to substitute a test database
- The Singleton design pattern is used for DB connections and for logger functionality