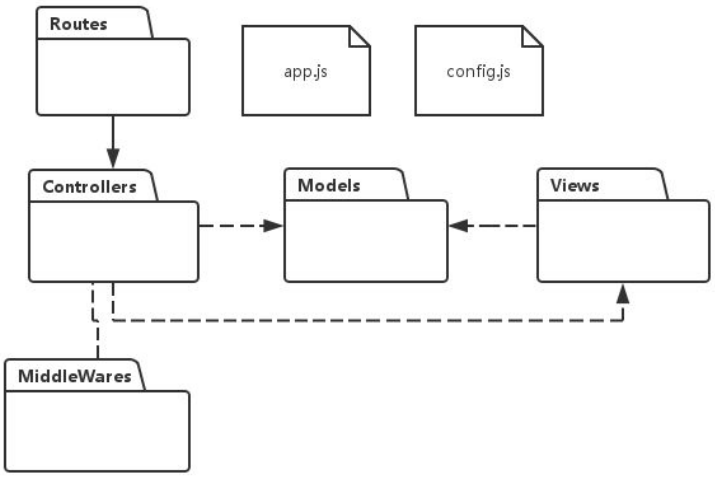
Survivable Social Network on a Chip Team S-16 A2

Explain vision of the system here.

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 Chrome will be supported)



High-Level Functional Requirements

- *Login/Register/Logout for all users (including administrators)*
- *Users can chat in a public chatroom, or chat between two users*
- Users can share and modify their status.
- Coordinators have the privilege to post announcements, administrators have the privilege to check and update users' profiles
- Users could search for information stored in the system according to some key words.

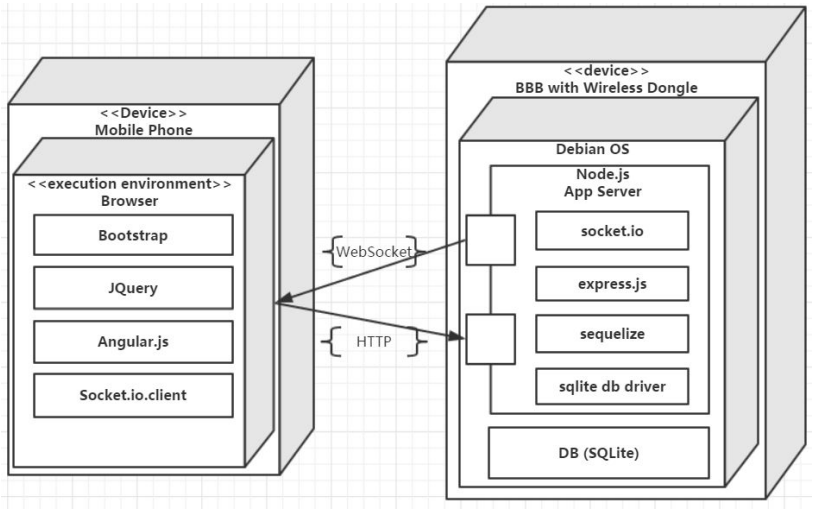
Top 3 Non-Functional Requirements

Works Out-of-the-Box > Usability > *Energy Consumption*

Both BBB and smart phones have the electrical limits. They could only work continuously for a few days or even shorter.

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
- 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 the screen without reloading the whole page.
- MVVM pattern on the front-end side. Angular.js supports two-way data binding.
- Modular development both on server side (nodejs) and client side (sea.js)
- ...



Design Decisions with Rationale

Name the design patterns that will be used and explain why they are used. Omit those implemented internally by the frameworks. Focus on the ones that you will implement yourself. Include other important design decisions.

- Encapsulate data and behavior in models for easy testing and better modularization
- Use **Adapter** design pattern is used to be able to substitute a test database for the production database during testing
- **Singleton**: The DB connection, as well as logger, should be singleton.
- **Observer**: To decrease coherence, related components should subscribe events in the system. For example, when a user posts a new message, DB should record it, and the server should push it to other users.
- **Facade**: All the complicated DB operations should be encapsulated behind a facade interface.
- **Factory** Method: In the front-end, there are many code that are repeatedly used in many places. We need to encapsulate them as different services. Then write some factory methods to create these services.
- **Bridge**: Different users might have different roles. A user should owns an attribute, role. Role should be an interface, which has different concrete implementations, such as common user and administrator.
- ...

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 that deserves explanation.

- **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**: light-weight DB
- **Sequelize**: an ORM tool for node.js, which allows us to implement DB operations in an OO way
- **Sea.js and Grunt:** organize and modularize client-side code ...
- **Angular.js**: MVVM implementation on the client-side, ajax request and response functions.
- **JQuery**: Bootstrap is based on Jquery. Furthermore, when we need to manipulate DOM elements frequently, JQuery is more convenient than Angular.
- **Node.js/Express**: We'll use node.js to develop the http server. Express.js is a basic MVC framework based on node.js.
- **Underscore**: provides many useful functions. We just use those functions instead of re-inventing the wheels.
- **JSON**: Communication protocal between client and server