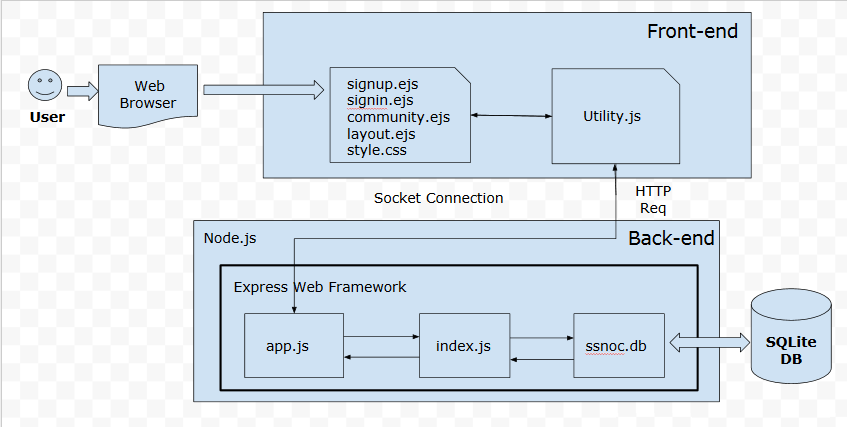
# Survivable Social Network on a Chip Team SB-1 SOS

Create a social network/platform where users can communicate in emergency situation mostly in case of power outage.

## **Technical Constraints**

* Lightweight database, SQLite
* Beaglebone Hardware (Limited CPU & Memory)
* Beaglebone board connected to external battery
* Debian operating system on BBB
* Rely on Express.js, Node.js, Socket.io
* Wi-Fi Speed and coverage

## High-Level Functional Requirements

* User should be able to create an account, sign in, set his/her status, and sign out.
* User should be able to chat publicly and chat privately
* User should be able to edit profile
* Co-ordinator should be able to post announcements
* Admin should be able to change user roles and delete user accounts

## Top 3 Non-Functional Requirements

* Availability of system (server)
* Performance & Reliability : The application must maintain its performance during emergency situations in terms of power outage, quick response time, strong signal strength
* Real-time Messaging: Application should allow real-time communication between various users.
* Usability :The application must be easily accessible and have user friendly UI, facilitating an intuitive user interaction in the case of an emergency situation.
* Extensibility : Application is built on iteration basis, therefore design and architecture should be compliant with extensible principles in order to accommodate weekly amendments.

## Architectural Styles with Rationale

* MVC : Express imposes MVC framework. We have structurized code in have dedicated directories for Controller, Model (js) and View (ejs) files.
* Client-Server : Client will provide interface to users and Server will process requests from a client. Application will be hosted on server side.
* RESTful : Allows reduced client/server coupling and thus extreme scalability.
* Repository : The data layer of our SSNoC application is isolated and accessed through sequelize.
* Event-based : Socket.io imposed to facilitate real-time chatting application. We have used socket extensively for chat, status sharing and post announcement operations.
* Pipe & Filter - Imposed as part of express web application framework. Each layer of pipe and filter is setup through the app.use() express function. This pattern is used in SSNoC to check user session before routing users to various pages.



## Design Patterns with RationaleUntitled (1).png

* Observer Pattern: to implement distributed event handling systems. Through server-client interaction broadcasting can be achieved with one-to-many relationship.
* Adapter Pattern: to implement an interface to access the database. This allows the underlying DB to be switched without having to change all the functions that access the DB.
* Facade: to manage several independent tasks through a single function. This has been used to delegate various join community needs (welcome modal, directory, public chat) to several independent functions.

## Other Design/Architectural Decisions

## EJS : easy and cleaner code

SQLite : Lightweight database

Bcryptjs : For password hashing to maintain security of user acc

Bootstrap : To create responsive UI for Mobile, Tablet, Desktop

Express-session : For maintaining user session

Ejs-locals : For code reusability in layouts

Sequelize : As ORM level to abstract DB (for MVC)

Shippable : Continuous integration

Istanbul : Code Coverage

## Responsibilities of Main Components

Node.js : In server side scripts to achieve event driven web application

SQLite : User accounts, chat messages, user roles will be saved in different database tables