Team 4 Mayday

Foundation of Software Engineering Section B

Simrata Gandhi
Keerthana Thangaraju
Jason Wang
Angel Fu

AGENDA

- Vision
- Functionalities
- Software Architecture
- Software Engineering Practices
- Quality Attributes
- Challenges
- Takeaways

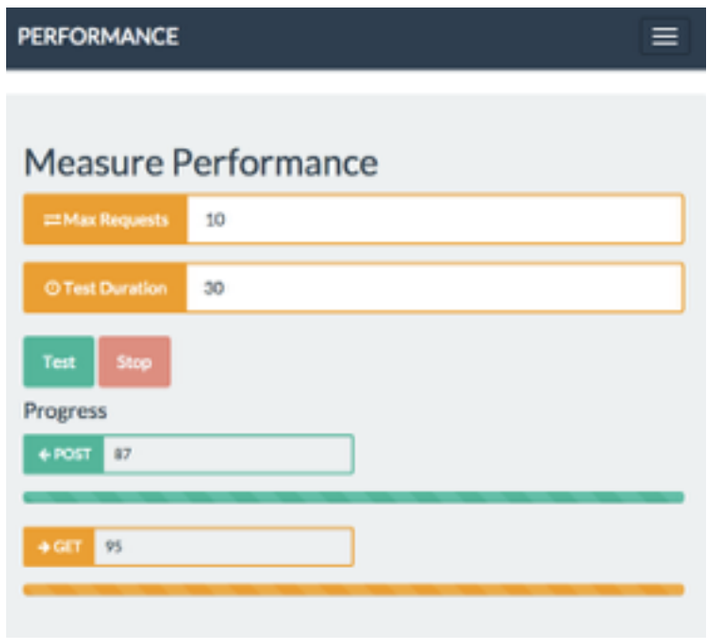
VISION

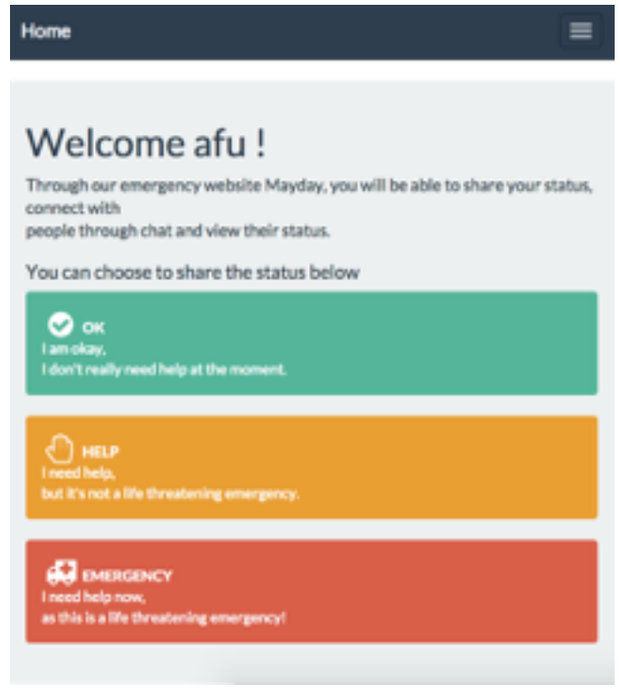
We want to help people to connect with each other when there are no connections to contact others due to natural disasters or other emergencies by building the Mayday emergency chatroom.

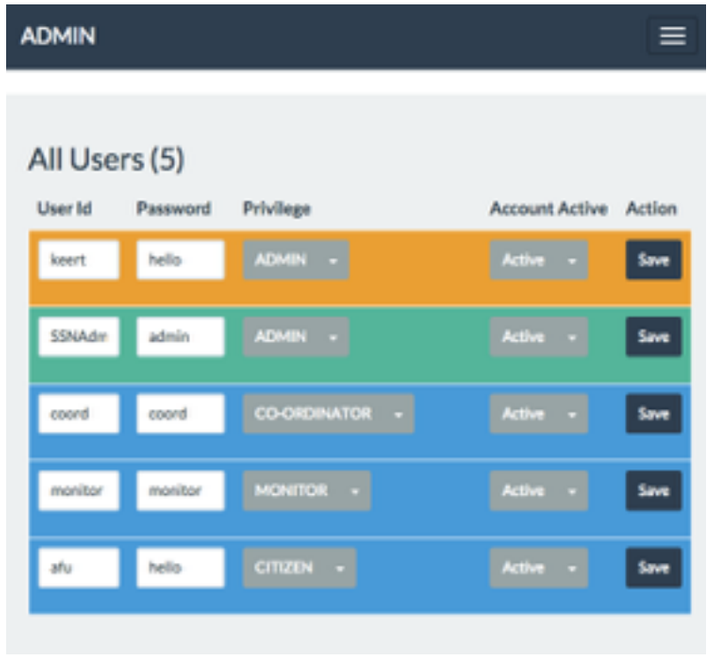
CONNECT THE UNCONNECTED

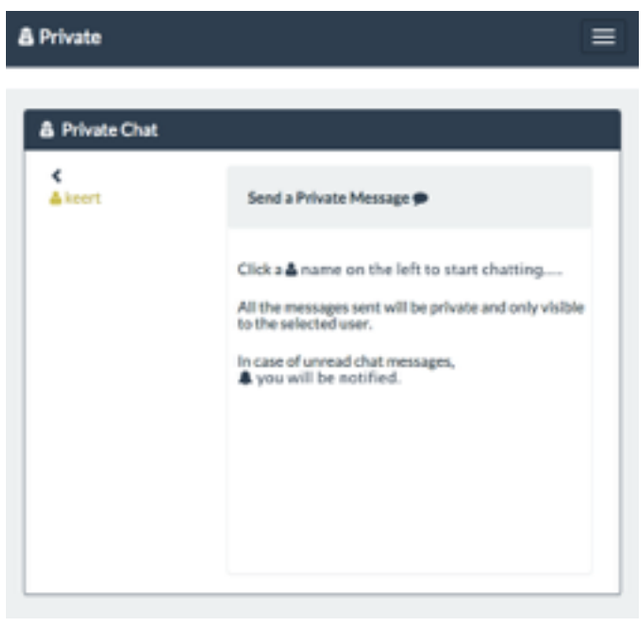
FUNCTIONALITIES

- First-aid Advice
- Join Community
- Chat Publicly
- Chat Privately
- Share Status
- Post Announcement
- Search Information
- Measure Performance
- Administer User Profile



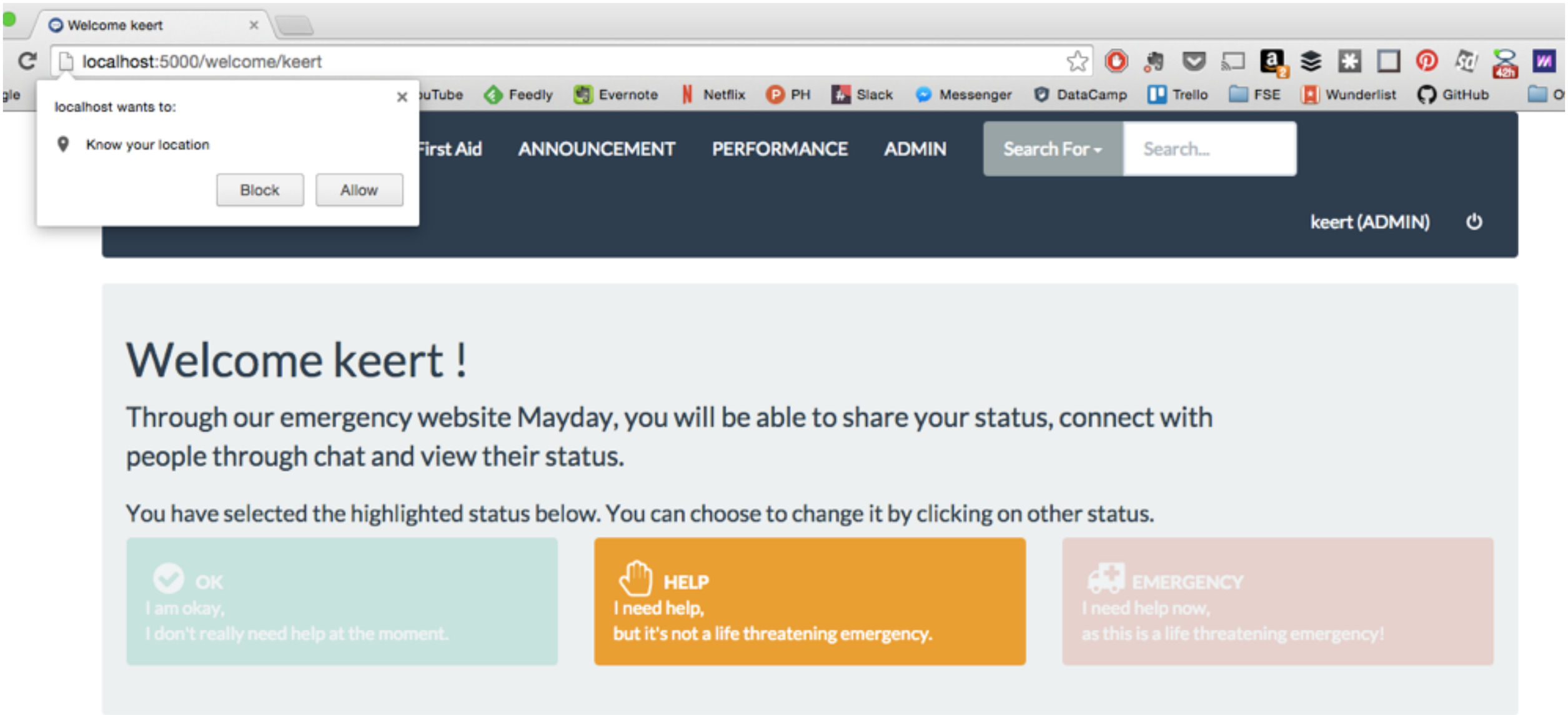




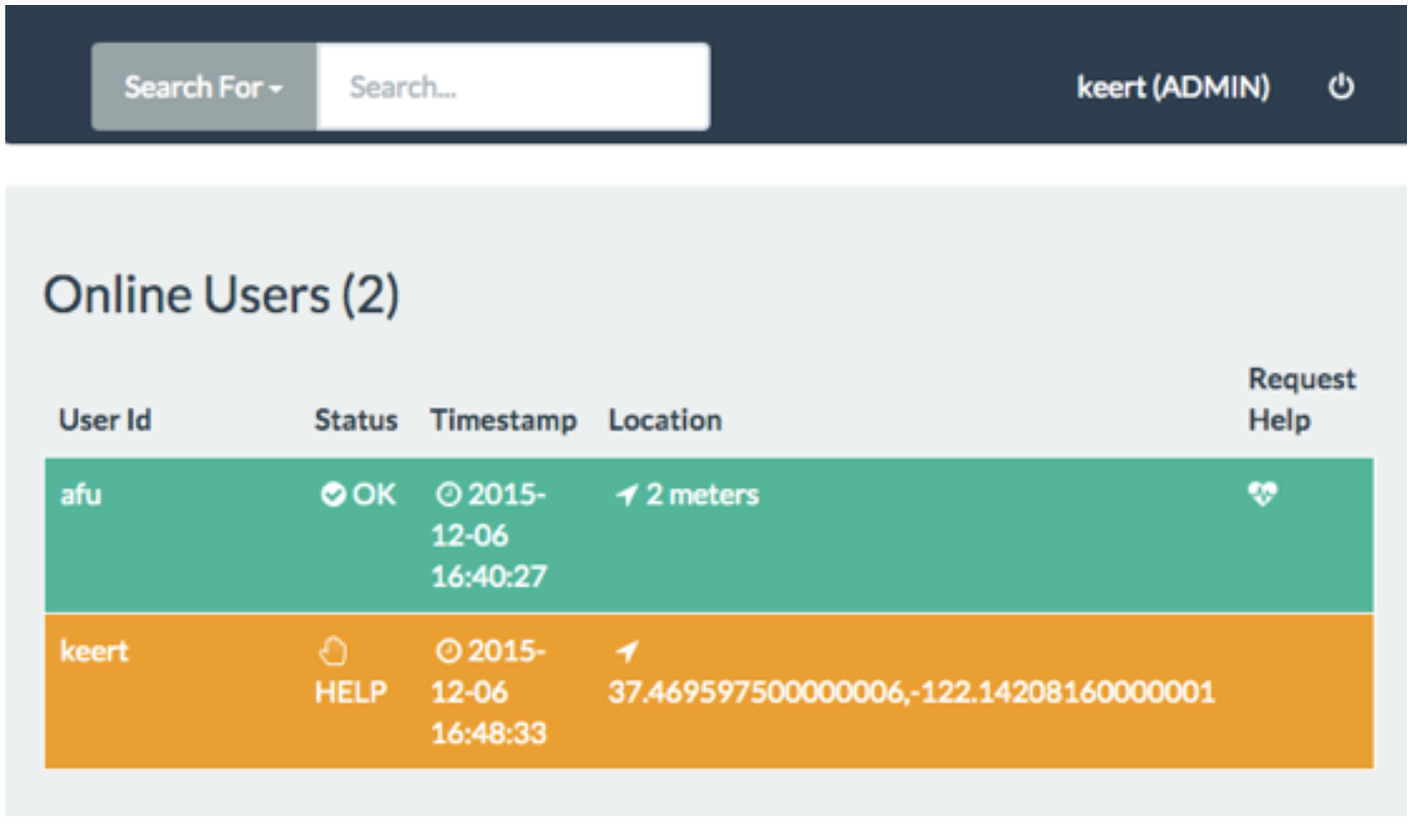


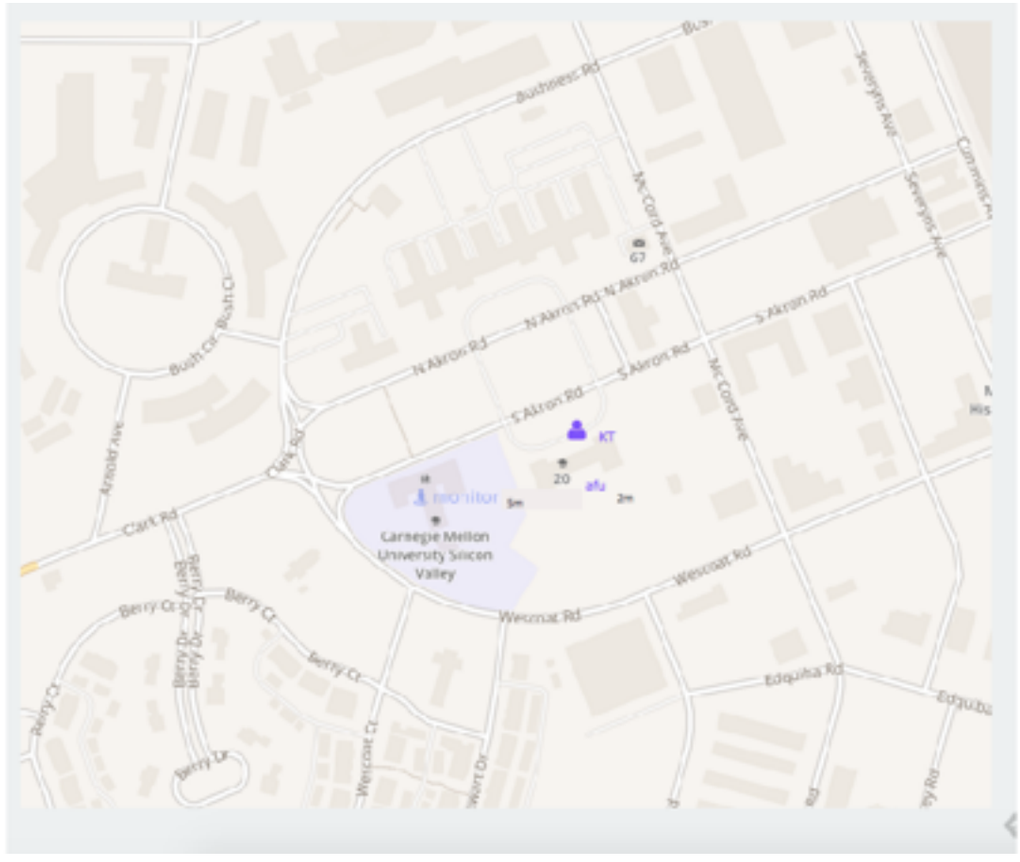
FIRST-AID ADVICE

Goal: Allow users to ask for help from nearby online users and look up basic first-aid advice in the chatroom



FIRST-AID ADVICE CONT'D



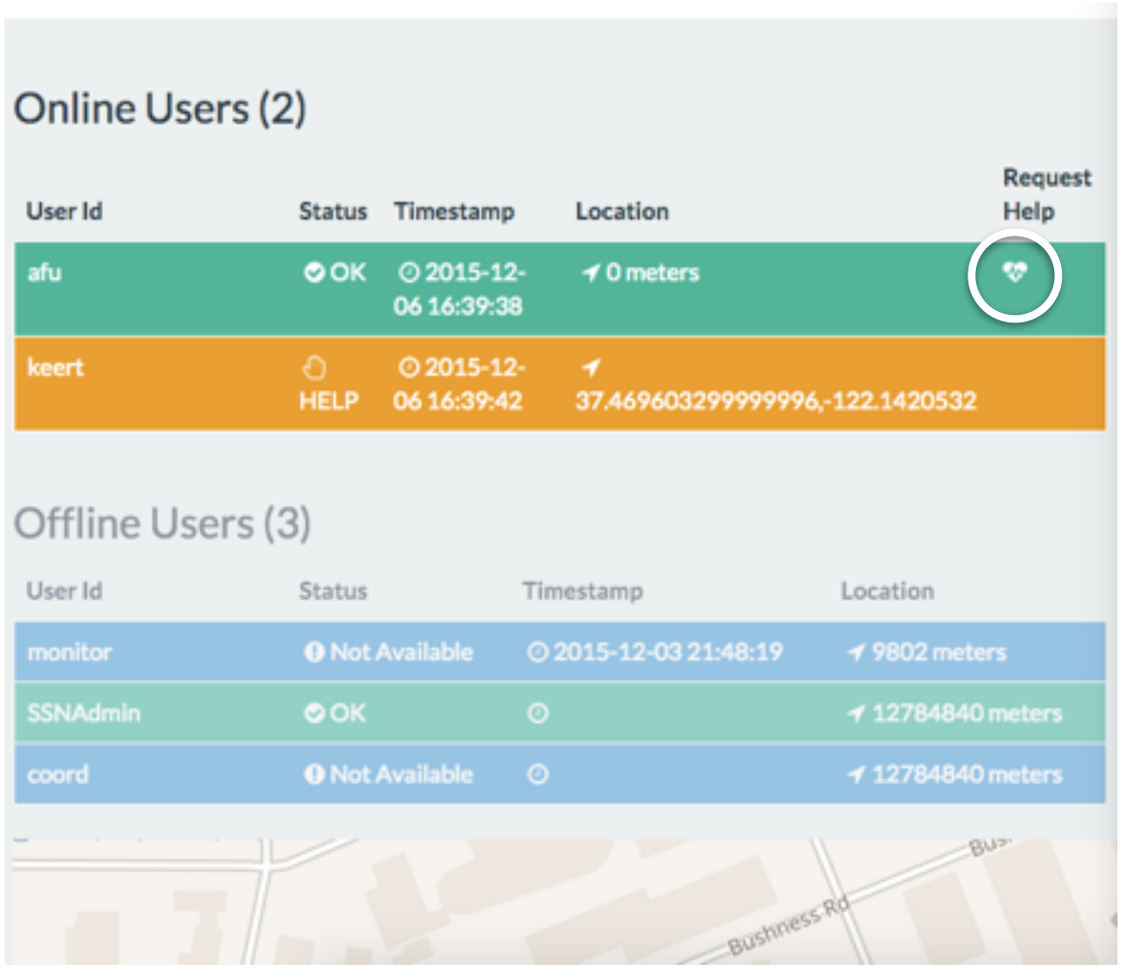


Mapbox

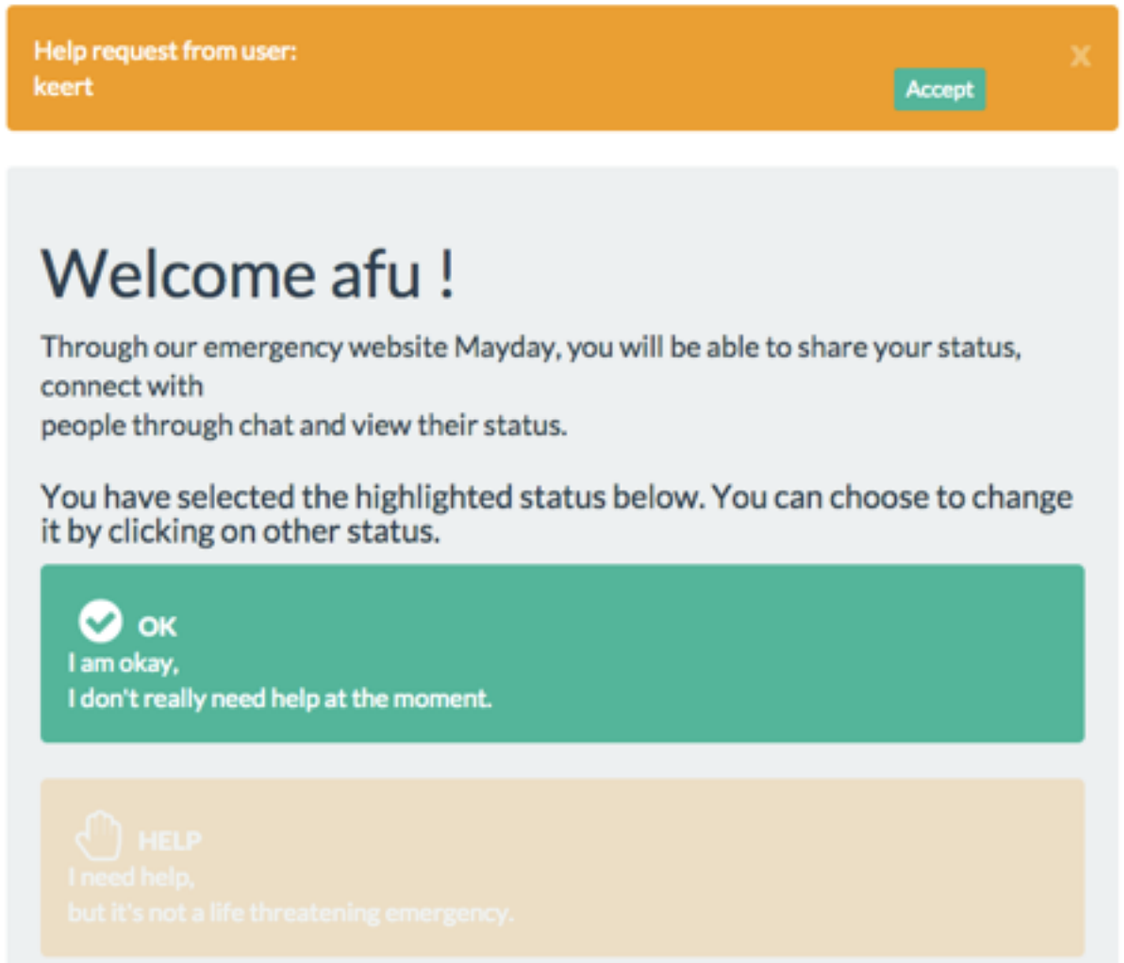
Allows maps to be rendered in real time on web and mobile devices

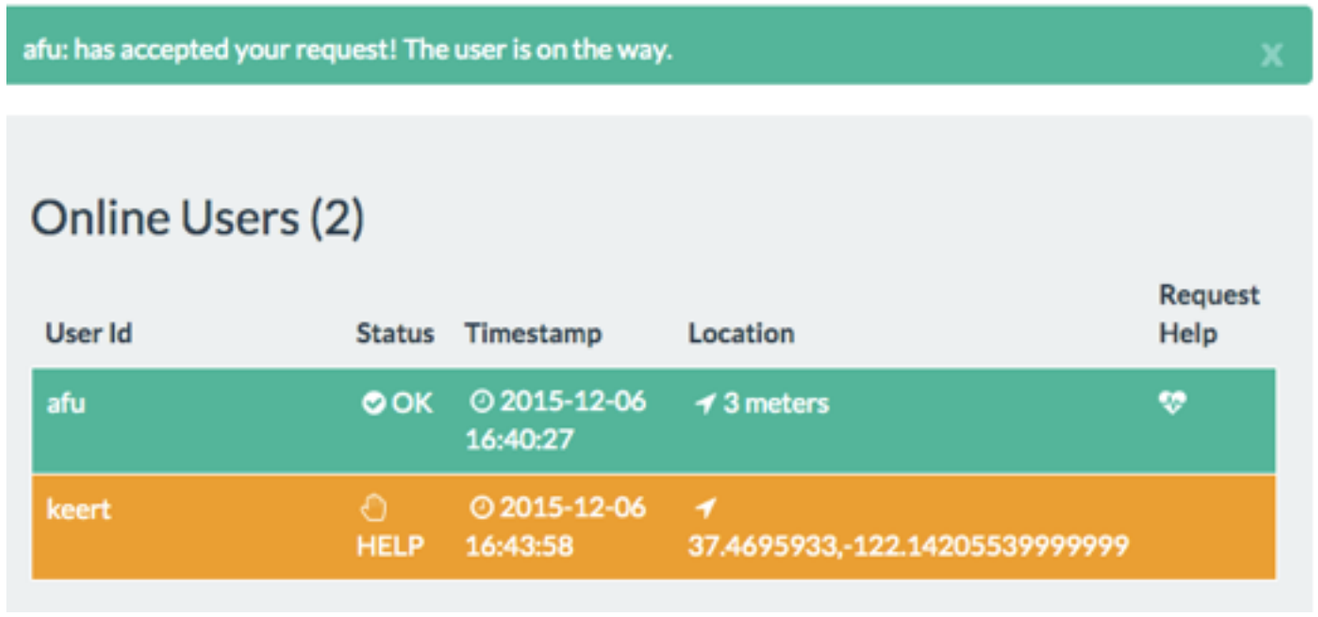
Plots markers on the map

FIRST-AID ADVICE CONT'D



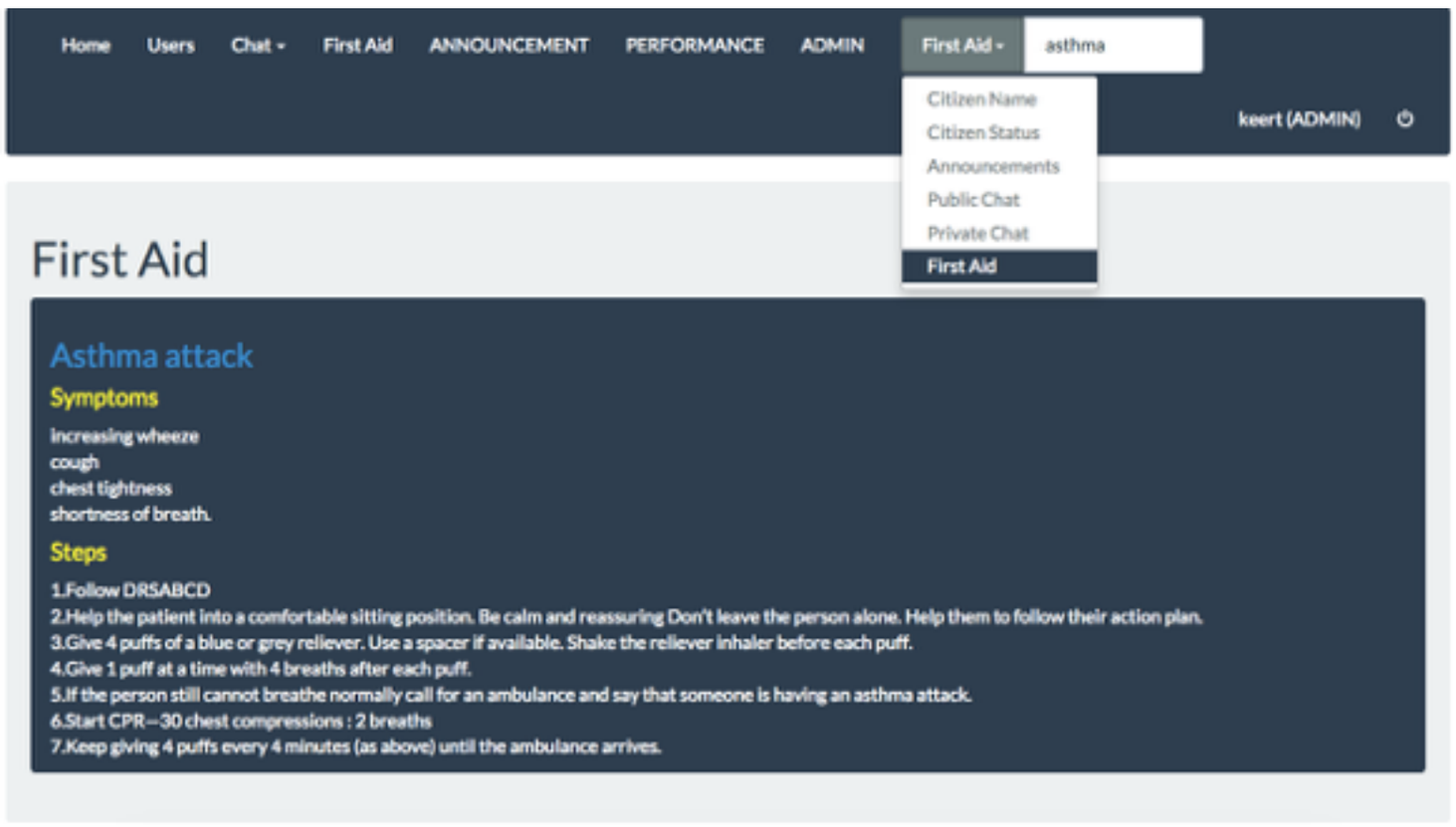
User requests for help from another user





The other user agrees to help

FIRST-AID ADVICE CONT'D

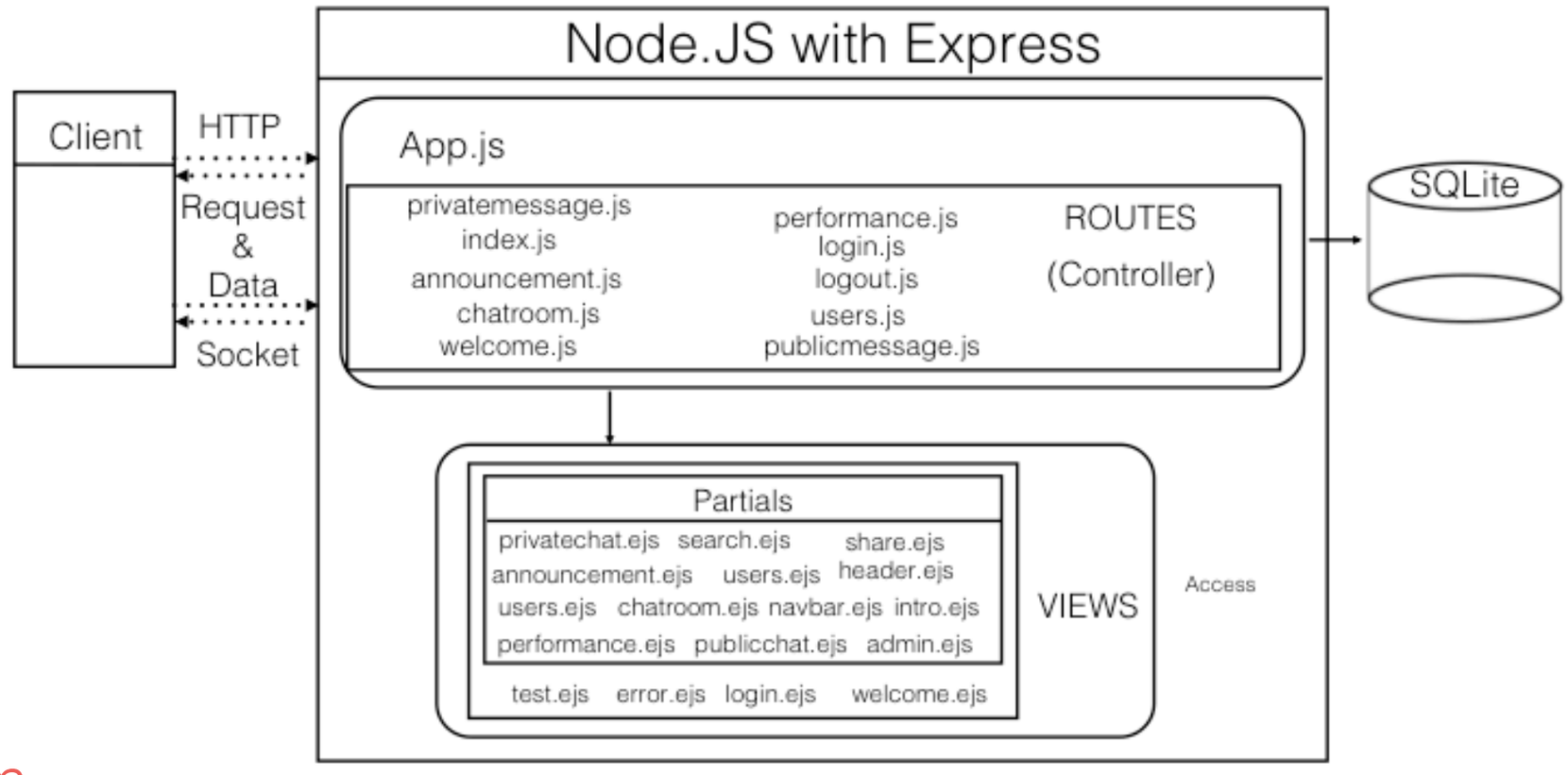


Users can browse and search first-aid advice in the chatroom

SOFTWARE ARCHITECTURE

Before

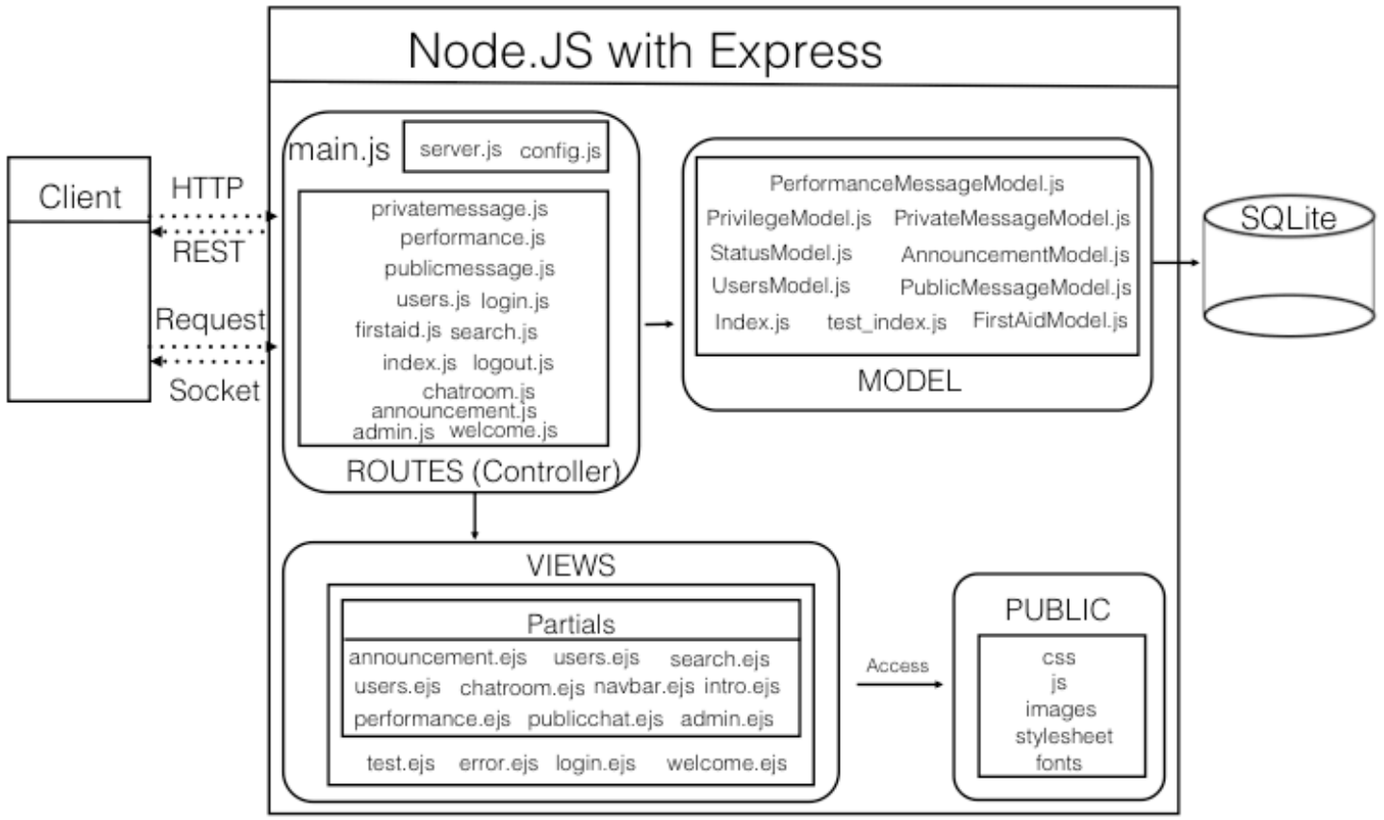
- Client-server
- Event-based
- Repository
- Restful
- Socket used for communication from client and server
- No models
- Unstructured code



SOFTWARE ARCHITECTURE CONT'D

After so much pain

- Client-server
- Event-based
- Repository
- RESTful
- MVC
- Socket communication only from server



MVC WITH EXPRESS, EJS, SEQUELIZE

Express (Routing)

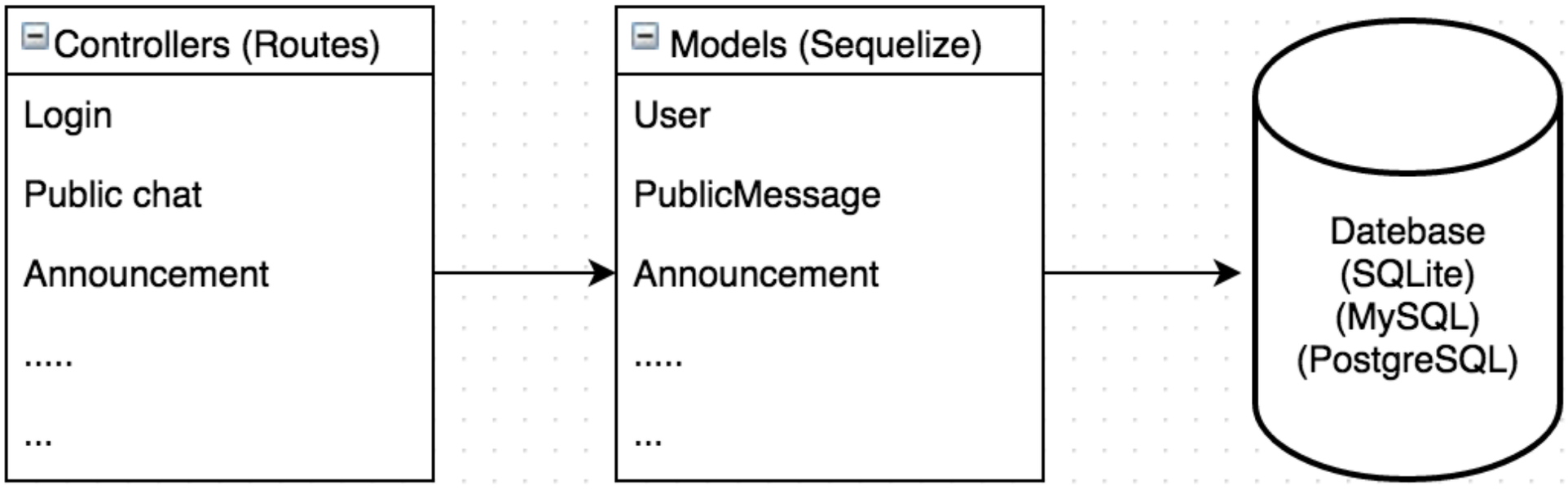
It provides easy routing framework to manage HTTP requests to create RESTful APIs for our application.

EJS (Views)

It allows us to create partial views and we only need to change the content in the layout using EJS.

Sequelize (Models)

It helps model Data Access Objects for SQLite tables.



SOFTWARE ENGINEERING PRACTICES

PAIR PROGRAMMING	• 2 members per team working on each module • Improved efficiency, communication among team members
CONTINUOUS INTEGRATION VERSION CONTROL	• *Branching* and *merging* on Github for every iteration to track changes • Helped tracking issues on time
HYBRID: AGILE + KANBAN	• Emphasis on value and also utilize time effectively to reduce waste
TESTING & COVERAGE (Mocha, Grunt and Shippable)	• Integration tests and automated script for testing http get & post requests • Help to detect bugs early
OO ANALYSIS & DESIGN	• Manage requirements

SOFTWARE ENGINEERING PRACTICES CONT'D

Practices Missed Initially

- In process unit testing: Unable to perform due to absence of Models in code

- Refactoring: Lack of modularization and unorganized code structure

REFACTORING

After we analyzed our problems and re-prioritized….

- Created models
- Conducted in-process unit testing
- Modified socket connections
- Implemented REST API fully
- Re-organized EJS files
- Removed unused code and Added comments for explanation

REFACTORING CONT'D

```
router.post('/', function(req, res){
    var annon = req.body.annon;
    var uid = req.body.uid;
        db.run("INSERT into ANNOUNCEMENTS(annon,user_id) VALUES ('" + annon + "','" + uid + "')", function(err, val) {
        // console.log("insert" + err + ":" + val);
        if (err)
            res.status(500).send({
            error: 'Announcement could not be stored!'
        });
        else {
             res.json({
            success: true,
            message: 'Announcement stored!',
            annon: annon,
            uid: uid

          });
        });
    });

// router.post('/', function(req, res){
//  console.log("post request received");
//  //console.log(req.body.annon);
//  //console.log(req.body.uid);
//  var annon = req.body.annon;
//  var uid = req.body.uid;
//  var result = [];
//      db.run("INSERT into ANNOUNCEMENTS(annon,user_id) VALUES ('" + annon + "','" + uid + "')");


router.post('/getHistory', function(req, res){
    console.log('getHitory request received');
    var lastLogin = req.body.lastLoginTime;
    var token = req.get('x-user-token')
    var result = [];
    console.log(lastLogin);
    console.log(token);
    db.serialize(function() {
        db.each("SELECT * FROM ANNOUNCEMENTS ;", function(err, row) {
            result.push(row);
        }, function() {
            console.log(result);
            res.send(result);
        });
    });
});
```

BEFORE

```
/*jslint node: true */
'use strict';

var express = require('express');
var router = express.Router();
var bodyParser = require('body-parser');
var models = require('../models');

module.exports = function(io) {
/*insert the announcement into database when an administrator announces something*/
    router.post('/publishAnnon', function(req, res) {
        models.ANNOUNCEMENTS.create(req.body)
            .then(function(announcements) {
                io.emit('showAnnouncements', announcements);
                res.status(201).json({
                    success: true,
                    message: 'Announcement stored!',
                });
            }).catch(function() {
                res.status(500).send({
                    error: 'Announcement could not be stored!'
                });
            });
    });

    /*fetch all historic announcemnets once a user enters announcements tag*/
    router.get('/getHistory', function(req, res) {
        models.ANNOUNCEMENTS.findAll()
            .then(function(announcements) {
                res.status(201).json({
                    success: true,
                    message: 'Get historic announcements',
                    announcements: announcements,
                });
            }).catch(function() {
                res.status(500).send({
                    error: 'Unable to fetch historic announcements!'
                });
            });
    });


    return router;
};
```

AFTER

QUALITY ATTRIBUTES

USABILITY	• Implemented the MVC architectural pattern • Allowed us to create synchronized views and provide intuitive usability and responsiveness
SECURITY	• Chatroom allows users to log in with credentials • Validated the token on the backend to ensure security
REUSABILITY	• After refactoring our code and creating models with Sequelize, we are able to reuse some of the code to reduce the redundancy

CHALLENGES

- A gap between team members' level of software engineering skill

- Struggles during the transitions between the previous iteration and the next one

- Not enough time spent on retrospective

TAKEAWAYS

- Helped out each other by answering questions and holding code review sessions
- Slowed down and re-evaluated our code and the team
- Ensured to reduce technical debt during each iteration
- Prioritized functionalities before design due to time constraints