A Scalable Web Application Framework for Monitoring Energy Usage on Campus

Group 57: Daniel Schroeder, Aubrey Thenell, Parker Bruni

Introduction

Project Summary and Purpose

Oregon State University is revered for its energy efficiency and sustainability and is committed to reducing its carbon footprint through renovations and sustainable considerations with new projects. Our project aims to provide an web interface application to monitor the energy consumption data of Oregon State Campus buildings so organizations such as the OSU office of Sustainability may have better knowledge about infrastructure decisions regarding energy usage. This project will also replace the current implementation for monitoring energy usage on campus which has proven to be costly and inefficient.

Introduction

Overview of Progress Report Topics

- Proposed Solution
- Goals and Stretch Goals
- Research:
  - Structural and Server Side Frameworks
  - Web/Database Hosting and Database Framework
  - Visualization and Front-end Frameworks
  - Language and Authentication
- Design of Interface Components
- Current Progress
- Issues Impeding Progress
- Solutions to Problem of Progress Restrictions

Introduction

Overview of Application

- Utilize the MEAN stack method of developing a web based application
- Host the application and database on a web based platform (AWS EC2)
- Read energy data from existing energy metering databases
- Be accessible from a web browser with internet access
- Have intuitive navigation tools such as navigation bars
- Has various levels of privileges and control for user accounts (General, Authorized, Administrative)
- Display energy usage data in an intuitive and presentable fashion

Introduction

Overview of Project Goals and Stretch Goals

- Required Goals
  - Allow administrative accounts to control various parts of the website without coding knowledge
  - Display “Blocks” of data, including graphs and charts of energy data
  - Display arrangements of Blocks known as “Dashboards”
  - Allow for groupings of Dashboards known as “Stories”
  - Create pages with a default Dashboard associated with each building
- Stretch Goals
  - Create cost tables and invoices for energy billing
  - View billing trends and information
  - Allow for mobile data entry to the database

Introduction

Documentation of Project

- Problem Statement
- Requirements Document
- Technology Review
- Design Document
- Progress Report

Documents found at: https://github.com/DSchroederOSU/SeniorCapstone

Research and Design Decisions

- Frameworks and utilities used
- Components used
- UI design of components

Research: Structural and Server Side Frameworks

AngularJS

ExpressJS

Image Sources:
https://www.w3schools.com/angular/pic_angular.jpg
https://camo.githubusercontent.com/fc61dcbdb7a6e49d3adecc12194b24ab20dfa25b/68747470733a2f2f692
e636c6f756475702e636f6d2f7a66593366c4c376546612d33303030783330302e706e67

Aubrey

Research: Web/Database Hosting and Database Framework

Amazon Web Services

MongoDB

Image Sources:
https://a0.awsstatic.com/main/images/logos/aws_logo_smile_1200x630.png
https://cms-assets.tutsplus.com/uploads/users/1116/posts/24835/preview_image/mongodb-logo.png

Aubrey

Research: Visualization and Front-end Frameworks

D3.js

Bootstrap

Image Sources:
https://raw.githubusercontent.com/d3/d3-logo/master/d3.png
https://camo.githubusercontent.com/5d417873a11d7175ec40fc3af3c6cb024a531dd7/68747470733a2f2f676574626f6f7473747261702e636f6d2f6173736574732f6272616e642f626f6f7473747261702d736f6c69642e737667

Aubrey

Research: Language and Authentication

node.js

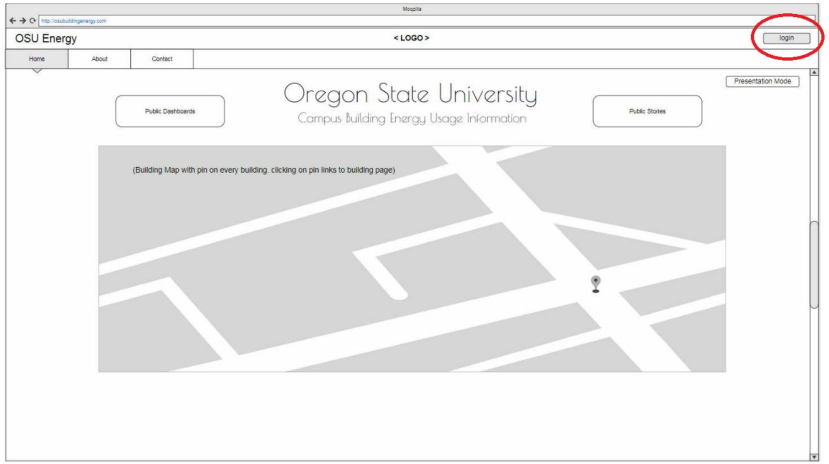
Google oAuth2.0

Image Sources:
https://nodejs.org/static/images/logos/nodejs-new-pantone-black.png
https://cdn.tutsplus.com/net/uploads/2013/07/oauth-retina-preview.jpg

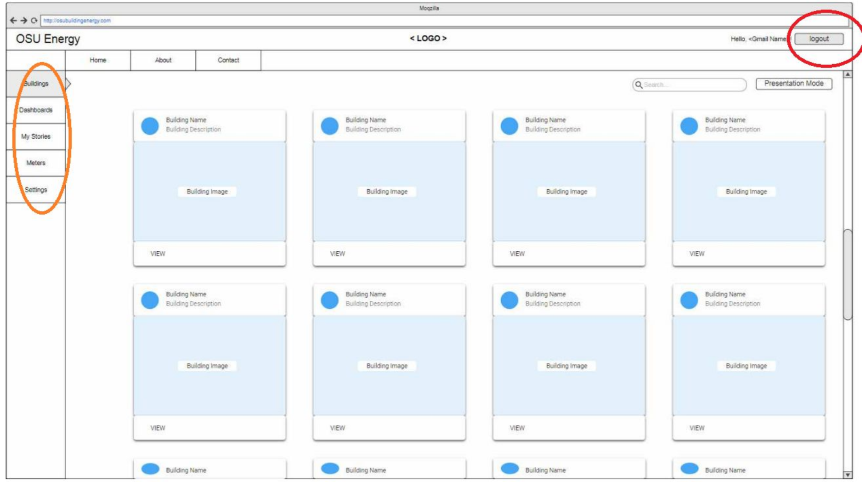
Aubrey

Design: Components

Login


Mozilla
http://osubuildingenergy.com
OSU Energy
< LOGO >
login
Home
About
Contact
Presentation Mode
Public Dashboards
Oregon State University
Campus Building Energy Usage Information
Public Stories
(Building Map with pin on every building. clicking on pin links to building page)

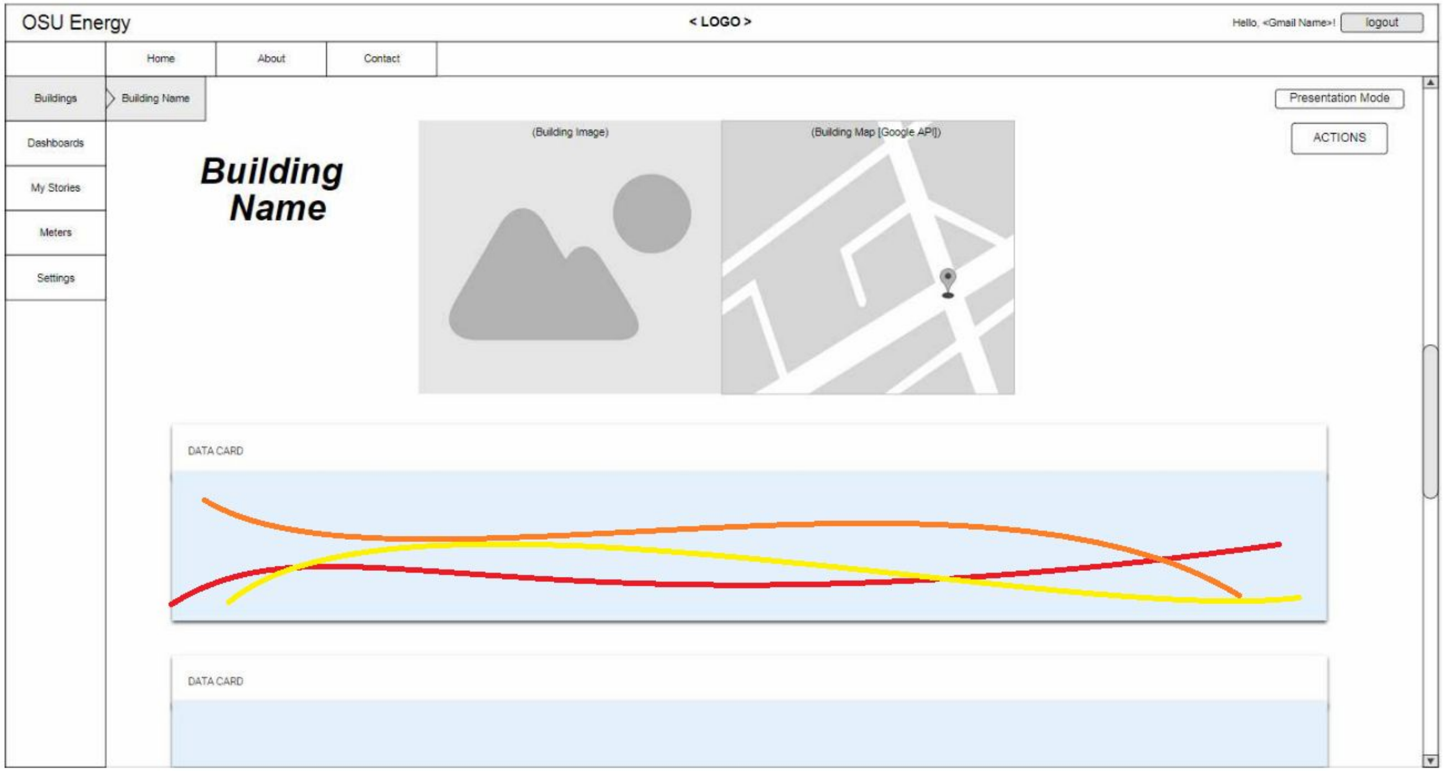
Logged in



Aubrey

Design: Components

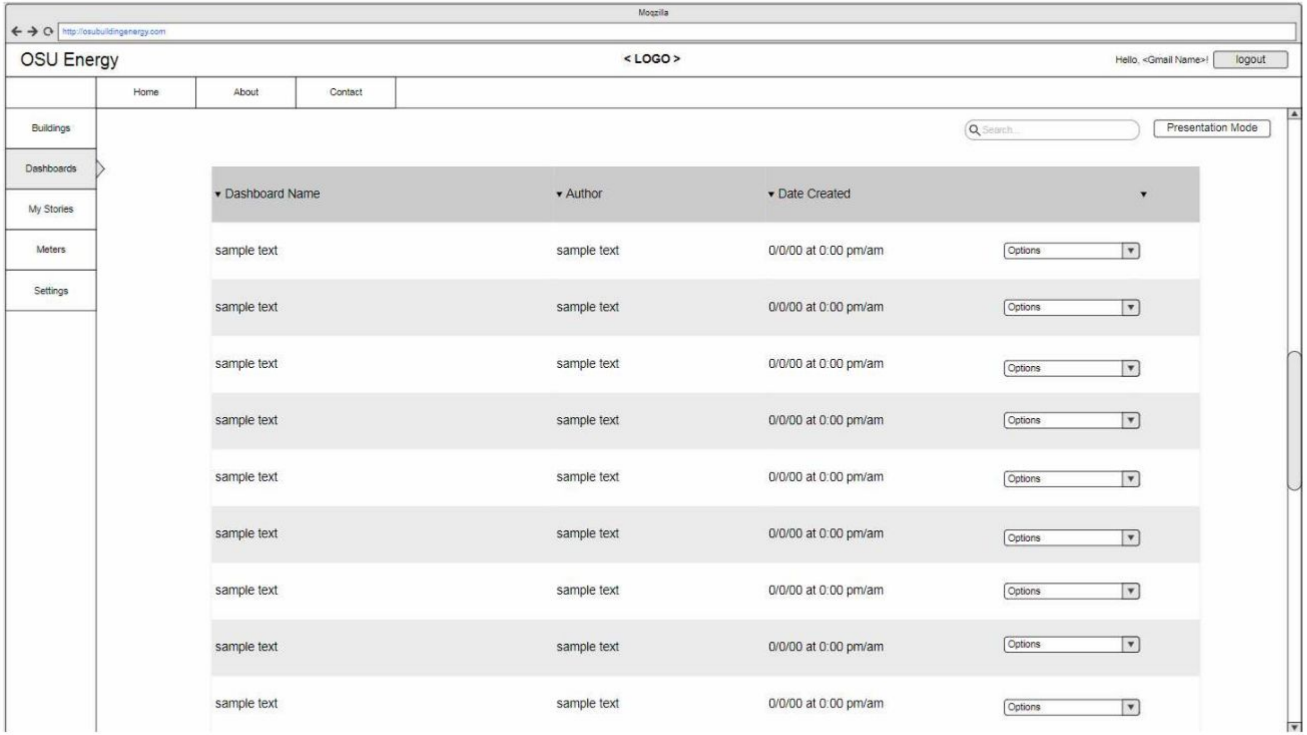
Block/Graph



Aubrey

Design: Components

Dashboard

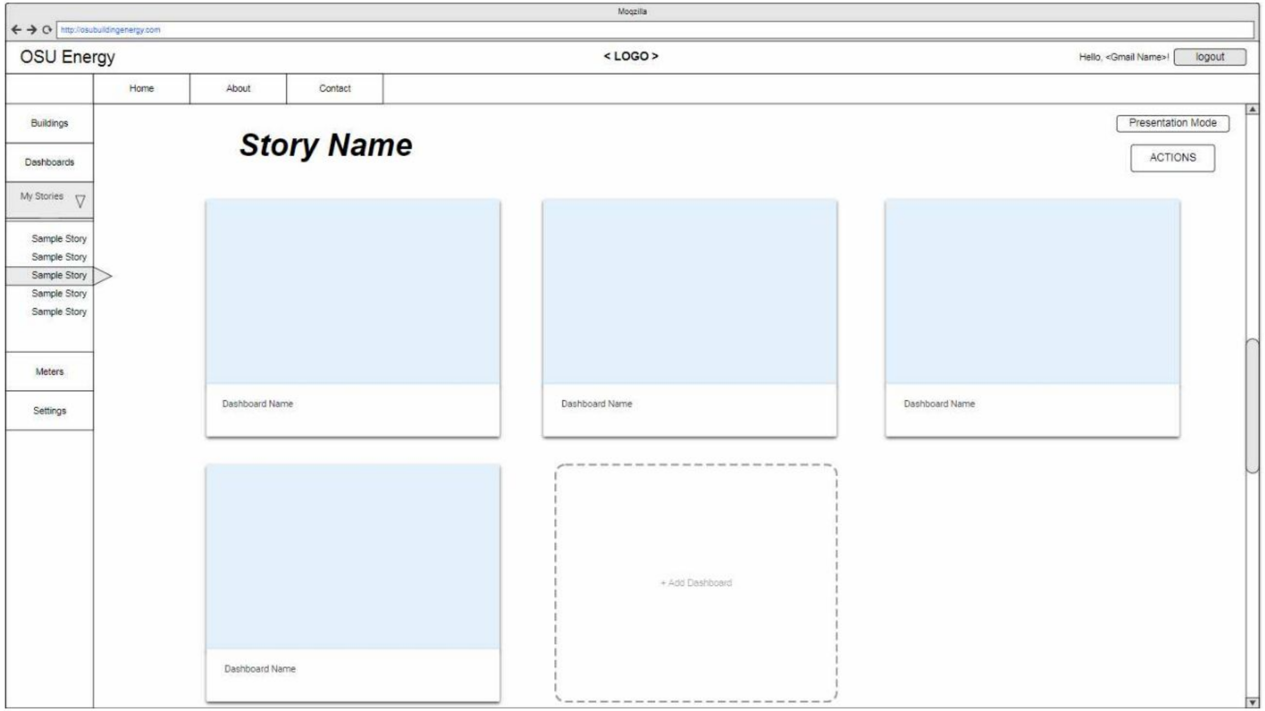


▾ Dashboard Name	▾ Author	▾ Date Created	▾
sample text	sample text	0/0/00 at 0:00 pm/am	Options ▾
sample text	sample text	0/0/00 at 0:00 pm/am	Options ▾
sample text	sample text	0/0/00 at 0:00 pm/am	Options ▾
sample text	sample text	0/0/00 at 0:00 pm/am	Options ▾
sample text	sample text	0/0/00 at 0:00 pm/am	Options ▾
sample text	sample text	0/0/00 at 0:00 pm/am	Options ▾
sample text	sample text	0/0/00 at 0:00 pm/am	Options ▾
sample text	sample text	0/0/00 at 0:00 pm/am	Options ▾
sample text	sample text	0/0/00 at 0:00 pm/am	Options ▾

Aubrey

Design: Components

Stories



Aubrey

Current Status

Usable product: None.

Smaller-scaled solutions: Multiple.

Research: Average.

Plans for execution: Bountiful.

Issues Impeding Progress

Biggest source of pain: AWS.

Source: https://sdtimes.com/amazon-aws-s3-outage-causing-widespread-issues-businesses/

Issues Impeding Progress: AWS

Solutions:

- Local development of small scaled solutions.

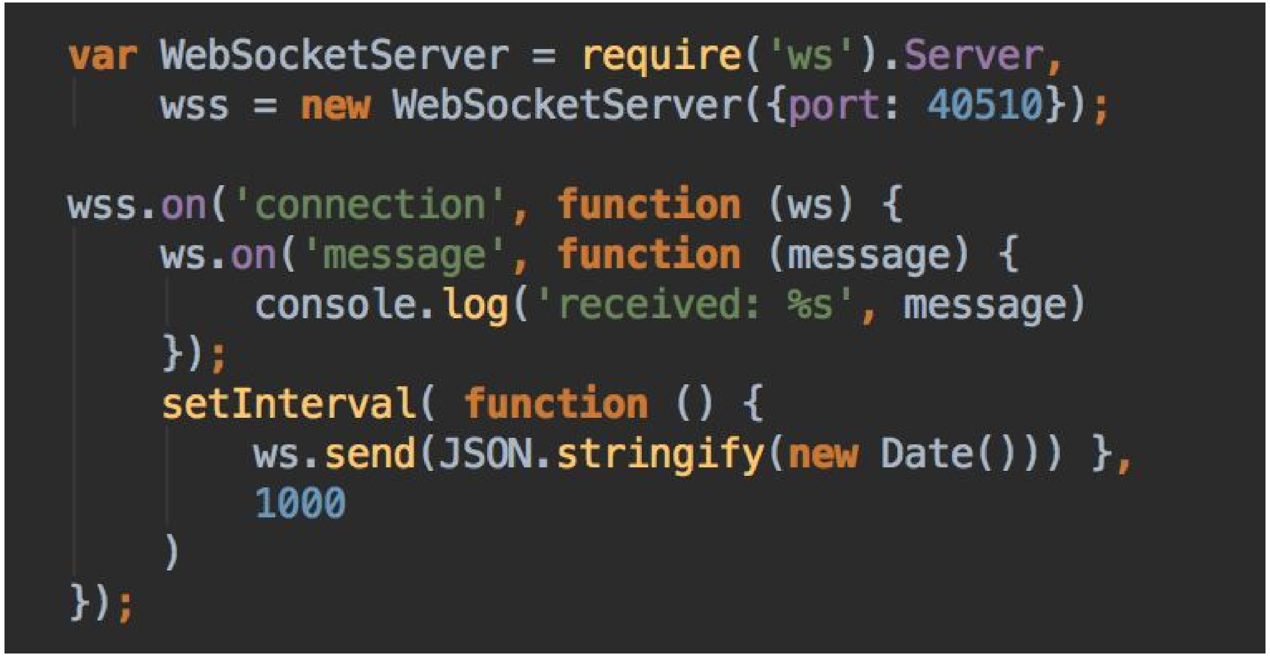
Local Solution: WebSocket

Key resources: npm “ws” package as WebSocket client and server

1. Create a WebSocket server with a specific port.
2. Serve the WebSocket server when serving the application server.
3. Send incremental data from the WebSocket server.
4. Receive data in the page of the application.

Local Solution: WebSocket

The websocket server is extremely simple to set up.

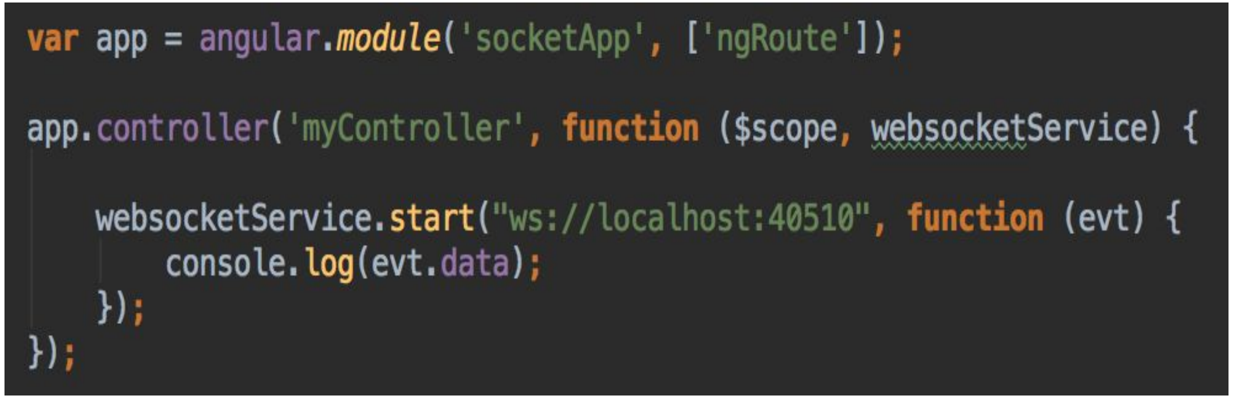


```
var WebSocketServer = require('ws').Server,
    wss = new WebSocketServer({port: 40510});

wss.on('connection', function (ws) {
    ws.on('message', function (message) {
        console.log('received: %s', message)
    });
    setInterval( function () {
        ws.send(JSON.stringify(new Date())) },
        1000
    )
});
```

Local Solution: WebSocket

Within the AngularJS controller, the application starts a connection with the WebSocket server and receives data sent.



```
var app = angular.module('socketApp', ['ngRoute']);

app.controller('myController', function ($scope, websocketService) {

    websocketService.start("ws://localhost:40510", function (evt) {
        console.log(evt.data);
    });
});
```

Our application formats the Date( ) object sent from the WebSocket server and passes it to a scope variable to simulate a real-time clock that updates every second.

Local Solution: WebSocket

Outcomes:

- Higher understanding of Angular frameworks and injecting page data.
- Working implementation of a WebSocket server with real-time data updates.

Local Solution: Authentication

Key resources: passport.js

1. Require passport.js authentication middleware
2. Redirect user to Google oAuth 2.0 API
3. Store Google token in user object
4. Authenticate user through sessions

Photo Source: https://github.com/jaredhanson/passport

Local Solution: Authentication

Passport.js

> "Authenticating requests is as simple as calling `passport.authenticate()` and specifying which strategy to employ." - passport.js documentation

Import Google strategy

```
var GoogleStrategy = require('passport-google-oauth').OAuth2Strategy;
```

Use Google strategy in route

```
app.get('/auth/google',
   passport.authenticate('google', { scope : ['profile', 'email'] }));
```

Local Solution: Authentication

Passport documentation to setup Google strategy.

Route middleware.

```
app.get('/route, isLoggedIn, function(req, res) {}

// route middleware to make sure a user is logged in
function isLoggedIn(req, res, next) {
   // if user is authenticated in the session, carry on
   if (req.isAuthenticated())
       return next();
   // if they aren't redirect them to the home page
   res.redirect('/');
}
```

Local Solution: Authentication

Outcomes:

- Simple Node.js authentication strategies
- Using Google for authentication
- Remove the need for passwords in the database
- Keep track of users throughout session
- Authenticate content being served with middleware

Local Solution: MongoDB

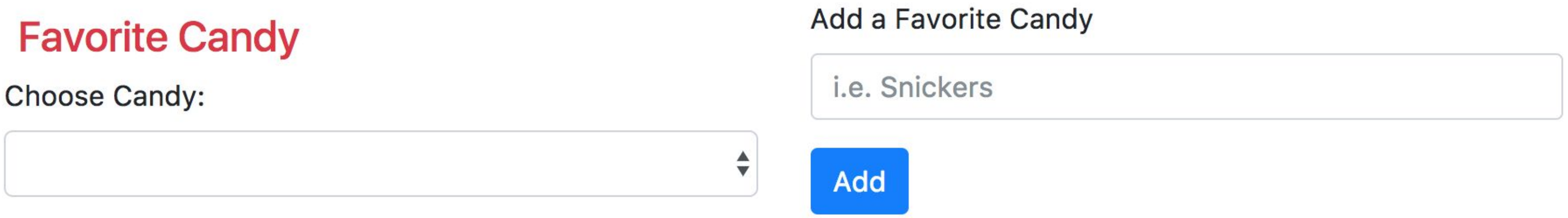
Key resources: MongoDB

Storing data and rendering dynamic content.

1. User submits form
2. AngularJS service posts to a route
3. Route handles data creation and database insertion
4. AngularJS injects new data to the page

Local Solution: MongoDB

Add favorite candy



Local Solution: MongoDB

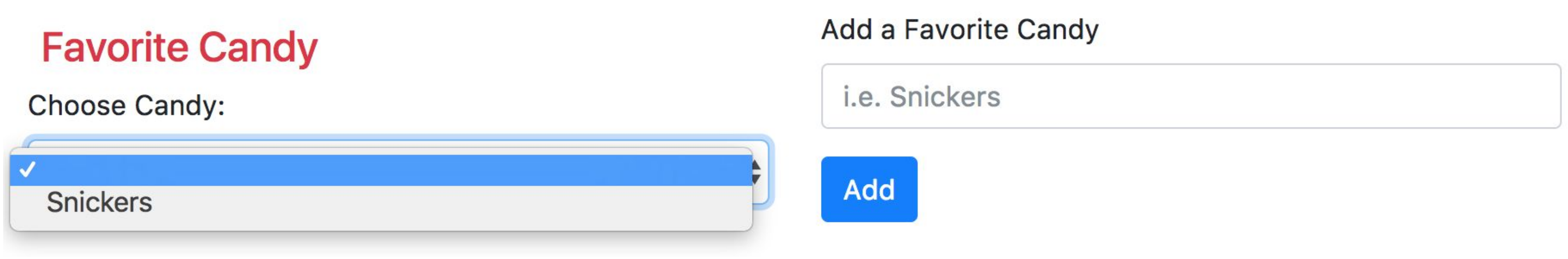
In the Angular service:

```
.factory('AddCandy', function($http) {
  return {
    create : function(candyData) {
      return $http.post('/api/candy', candyData);
    },
  }
});
```

In routes.js

```
app.post('/api/candy', function(req, res) {
  var new_candy = {name: req.body.text};
  req.user.candy.push(new_candy);
  user.save(function(err) {
    if (err)
      throw err;
    res.json(req.user.candy); // return all candy in JSON format
  });
});
```

Local Solution: MongoDB



AngularJS loads the data from the route response into the drop down menu.

Local Solution: MongoDB

Outcomes:

- Storing objects to a MongoDB database
- Pulling data from a MongoDB database
- Dynamically updating data on the page based on changes to the database
- Using AngularJS factory functions as services
- AngularJS attributes like ng-options, ng-click, and ng-include

Local Solution: Dynamic Page Content

Serving new html to the content container from navigation bar

1. User chooses a navigation item
2. Navigation item posts route to parent scope variable
3. AngularJS attribute ng-include calls the selected route
4. New HTML is sent from the route to the page

Local Solution: Dynamic Page Content

```
//navigation.html
<li class="nav-item">
  <a class="nav-link" href="" ng-click="$parent.viewPath='/profile'">
          Home</a>
</li>
<li class="nav-item">
  <a class="nav-link" href="" ng-click="$parent.viewPath='/story'">
          Stories</a>
</li>

//index.html
<div class="container-fluid mt-1">
  <div class="row" >
    <nav id="navigation" class="">
      <div ng-include="'/navigation'"></div>
    </nav>
    <main role="main" class="col-sm-9 ml-sm-auto col-md-10 pt-3">
      <div ng-controller="profileController" ng-include="viewPath"></div>
    </main>
  </div>
</div>
```

Pass route
as variable

Call Route

Send html

```
//routes.js
app.get('/route', isLoggedIn,
function(req, res) {
   res.render('route.html', {
       user : req.user // get the
user out of session and pass to
template
   });
});
```

Local Solution: Dynamic Page Content

Outcomes:

- Learn about AngularJS scopes and nested scopes
- Inject static html to the content container
- Create a practical navigation component

Conclusion

Plans moving forward:

- Begin heavy development over winter break in preparation for winter term
- If AWS issue does not resolve, take matters into our own hands
- Begin pushing code to a collective source repository
- Convert applications to complement our project’s specific needs