

# **Web Controlled Smart Drainage and Sub Irrigation Control System**

DEC1602: Michael Parker, Anne Ore, Adam Wolter, Rodney Barto, Griffen Clark

dec1602.com

docs.dec1602.com

dec1602@iastate.edu

SE 492

Adviser: Nicola Bowler

Client: Agri Drain

December 6, 2016

# Table of Contents

<b>Revised Project Design</b>	<b>2</b>
Introduction	2
Design	2
Initial Designs	2
Screen Flow Diagram (Front End)	3
System Block Diagram (System)	4
Project Requirements/Specification	4
<b>Implementation Details</b>	<b>5</b>
JavaScript and Node.JS	5
React and React-Router	5
Semantic UI	6
Express	6
Bookshelf and Knex	6
<b>Testing Process and Results</b>	<b>7</b>
Mocha/Chai/Sinon	7
Istanbul.js	7
CircleCI	8
<b>Appendix I: Operational Manual</b>	<b>9</b>
Setup	9
Demo	10
Log in screen	10
Device map	11
Device Information	12
Alarms	13
Graphs	14
Users	15
Create New User	16
Edit User	17
Account Settings	18
Test	19
<b>Appendix II: Alternative versions of design</b>	<b>20</b>
Front End	20
Backend	21
<b>Appendix III: Other Considerations</b>	<b>23</b>
<b>References</b>	<b>24</b>

# Revised Project Design

## Introduction

Our project's aim is to create a web application to allow farmers to remotely control the amount of water that is being used to irrigate crops and to observe flow rates, nutrient levels, water levels, and rainfall totals from either a computer, tablet, or smartphone.

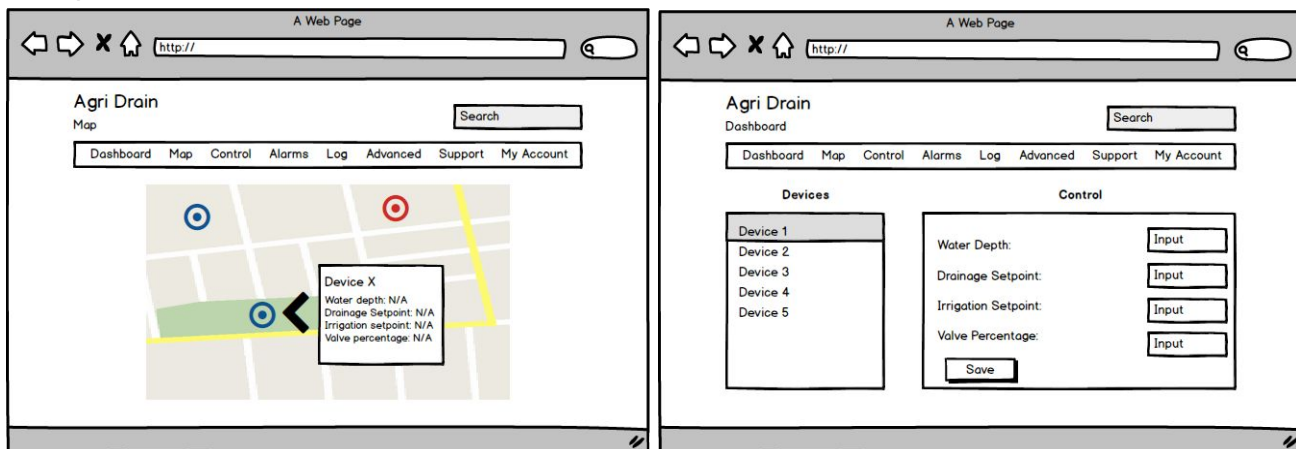
The purpose of this project is to allow remote communication between the farmers and their irrigation systems that are mechanically implemented. It will allow more freedom and ease for the farmers to control and monitor water levels in their crops by using a user-intuitive graphical user interface. The web application that will stem from the success of this project will have a low learning curve in order to make it convenient and easy to use. The purpose is also to bridge communication between data sent from the user interface to a server and how that data can be interpreted by the server, and vice-versa.

We aimed to create a functional prototype that represents the web application that would be used to allow the user to retrieve readings from water logger devices, and send control settings to irrigation control devices in the field.

## Design

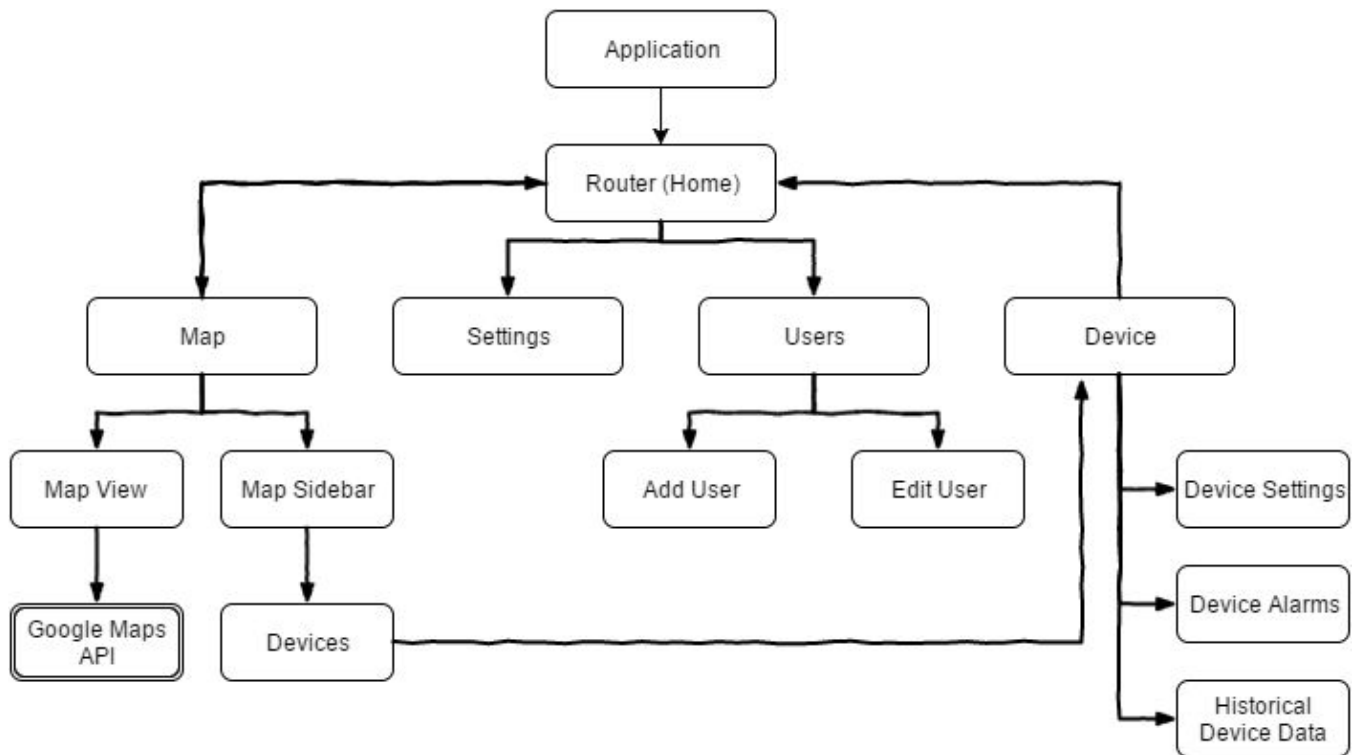
### Initial Designs

For our front-end design, we approached the website with the idea that it should be user intuitive, which was one of our major non-functional requirements. The user must be able to know how to navigate everywhere on the website without getting lost. This led to our initial set of mockups. Below are a couple examples of those.



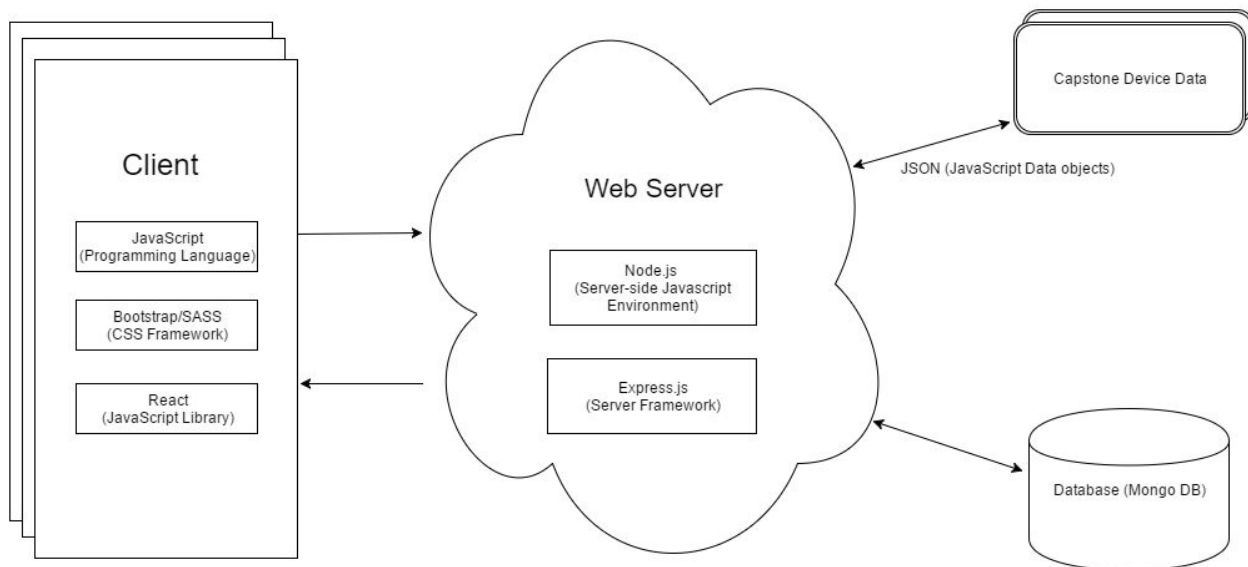
We wanted to use what's commonly referred to as a "Navbar" which is visible to the user at all times while using the application. This also helped us better understand our project's modules. These mockups then led to our front-end block diagram, which is essentially a screen-flow diagram, that outlines the navigation of our web application.

## Screen Flow Diagram (Front End)



This block diagram allowed us to further modularize our website. For example, instead of having a “Devices” tab on the Navbar, we decided to allow the user to navigate to the list of devices from the Map page. Also, instead of an “Alarms” tab, we decided to create a tab view on the individual device page that allows the user to edit the Alarm settings for that specific device.

## System Block Diagram (System)



For our back-end, we use Express.js (server-side framework) to read and write to a MySQL Database. Our entire project architecture looks roughly like the above figure. However, since a separate group was tasked with creating the hardware, we have not been able to have our web application communicate with the water level control structures.

## Project Requirements/Specification

The following are functional requirements for the application:

- This project needs to allow users to remotely change the settings on an Agri Drain device.
- The users need to be able to see current and historical sensor data.

The following are non-functional requirements for the application:

- This project will be a web application, but it needs to be easily usable on mobile devices.
- We will need to have an intuitive user interface.
- The historical data should be in a format that is easy to back up.
- The code should be documented well enough that future developers can add to it.

# Implementation Details

## JavaScript and Node.JS

Our project was completed using the JavaScript programming language, running on systems with the Node.JS runtime installed. There are several reasons for why we selected this programming language:

1. Our team is more familiar with JavaScript than any other programming language, making it easy for the team to contribute to the project without too much overhead that comes with learning a new language.
2. JavaScript is the most used language on GitHub [1], and the Node Package Manager (npm) has more packages than any other package manager for any other language [2], despite its youth. This means that we have a lot of resources available to us to learn from, and tons of tools and libraries to choose from.
3. JavaScript is currently the only language that allows you to write full stack web applications without having to use a different language for the backend. This means that our team didn't have to switch contexts when working with server-side or client-side code, which reduced the complexity of the project.

## React and React-Router

For our client-sided code, we used React, a library written and maintained by Facebook, for creating user interfaces.

React emphasizes a design pattern in which you create small, reusable components which serve as different pieces of your user interface. Each of these components consists of both markup and behavior, allowing them describe what they look like and how they can be interacted with.

React components use a special syntactic sugar called JSX [3], which allows you to insert XHTML-like syntax in your JavaScript code. JSX is the main reason why React components can define both markup (XHTML) and behavior (JavaScript).

Because React components are meant to be small, re-usable, and data-driven, they are really easy to write test cases for, and easy to consume in other components. This has allowed us to simplify our testing approach and achieve a high percentage of test coverage.

React-Router is a library that is used to expand the capabilities of React by adding client-sided routing to our application. This means that when a user navigates to a new page, the page transition is done entirely on the client side, without needed to fetch new HTML or JavaScript from the server, and the URL is updated using the HTML5 History API. As a result, it is extremely fast to switch between each of our pages, which are defined as their own React components.

## Semantic UI

The look-and-feel of our client-sided code was provided by a library called Semantic UI. Semantic UI provides us with a consistent set of styles and theming that we can use to make the application extremely user-friendly. Semantic also gives us the ability to customize the set of styles to use different sets of primary colors, which allowed us to utilize Agri Drain's brand colors and enforce consistency between our application and the main Agri Drain website.

Unlike other front-end style frameworks, such as Bootstrap and Material Design Lite, Semantic's usage of Progressive Truthfulness [4] allows for easily understandable class names to be added to DOM elements in order to enhance the markup. This, combined with our small React components, makes for easily readable, easily testable code.

## Express

Our backend code used Express.js, which is a framework for Node.js web applications. We used Express.js to provide a simple backend for our client-sided code, which is used to serve all of the assets needed to the browser in order to view the app (Semantic UI CSS, bundled JavaScript code, fonts, icons, etc) and respond to different requests made by the client.

Our server-side code adheres to a typical Model-View-Controller architecture. We have configured Express.js to serve static client-sided view files, created controllers that configure routes that the client can interact with to read and create resources on the backend, and set up models that define the different resources that clients will be interacting with.

## Bookshelf and Knex

Bookshelf is an Object Relational Mapping (ORM) framework, powered by the Knex query builder library. We were able to utilize Bookshelf to define each of our models and how they should be represented in our database. Bookshelf also made the process of interacting with the database much easier, as it took care of generating SQL queries for us, as we were able to represent our INSERT, UPDATE, and DELETE statements through JavaScript code.

Using these tools, we were also able to define what the database schema looks like and how each table is related to each other. This allowed us to create a few database helper scripts - one that performs an entire database migration for an easy setup process for the client, and another that adds fake data to the database for testing and demoing purposes.

# Testing Process and Results

## Mocha/Chai/Sinon

Our very thorough testing process was driven by the combination of Mocha, a testing framework, with Chai and Sinon, which are assertion and stub libraries, respectively. With the use of these technologies, our project has a test case for almost every line of code used in production. With this implementation, if a single line of code is unintentionally changed, it should cause tests to fail until further inspection and correction is done. The Mocha framework allows for tests to be written in a very easy to understand structure; below is an example of a Mocha/Chai test case.

```
describe('Login Page', () => {
  it('should contain an image header', () => {
    const imgHeader = renderedElement.childAt(0);

    expect(imgHeader).to.have.tagName('nav');
    expect(imgHeader).to.have.className('header ui fixed menu');
  });
});
```

Initially, our team wanted to follow a development process called test-driven development (TDD). During TDD, the developer looks at what is required, creates a failing test case that satisfies the requirement, and then proceeds to write code to make the test case pass. This approach is supposed to “encourage simple designs test suites that inspire confidence.”<sup>[5]</sup> However, since the majority of the team was unfamiliar with this approach, most of the team members wrote code and tested later, while still ensuring the project maintained high code coverage with our unit tests.





# Istanbul.js

Our team used a tool called Istanbul.js for ensuring we had a high code coverage metric. Istanbul.js provides an in depth report on various code coverage criteria. For example, at a first glance of the given report, it is easy to see an overall metric for statement coverage, branch coverage, function coverage, as well as individual line coverage. To get more detailed, it is possible to dig into separate components until eventually the code is presented with exactly what is tested and what isn't tested. Below is an Istanbul.js generated report of our project's code coverage.

## All files

93.63% Statements 147/157   85.19% Branches 23/27   92.31% Functions 84/91   93.57% Lines 131/140

File ▲		Statements ⇅		Branches ⇅		Functions ⇅		Lines ⇅	
client		90.91%	40/44	100%	7/7	87.1%	27/31	91.67%	33/36
client/components		100%	25/25	50%	2/4	100%	17/17	100%	21/21
client/pages		100%	17/17	100%	2/2	100%	10/10	100%	16/16
client/pages/device		89.66%	26/29	100%	6/6	87.5%	14/16	88.89%	24/27
client/pages/map		92.86%	39/42	75%	6/8	94.12%	16/17	92.5%	37/40

# CircleCI

Our project uses CircleCI, a continuous integration and delivery platform, to ensure that only high quality code gets pushed to our master branch in our project repository. CircleCI is able to integrate with various version control systems, such as GitHub or Bitbucket. By integrating CircleCI with our GitHub repository, team members are forced to have tested and well written code otherwise it notifies the whole team. This is enforced by making CircleCI run all of the project's test cases as well as ESLint. ESLint is a linting tool that ensures all of our code follows the same coding conventions/styles. The makes the project code easier to read and maintain as well as improving efficiency by not allowing unused functions or variables.

# Appendix I: Operational Manual

## Setup

Below are the instructions for setting up this web interface for demoing on your computer.

All data is randomly generated in the setup stage. An administrator is generated with the username "admin@agridrain.com" and the password "admin". A fake user with fake devices will also be created, for demo purposes. All of this information can be modified (and new users/devices can be added) by the administrator.

For this tutorial, you will need to use Terminal on Mac OSX or an equivalent on Windows.

1. **Install npm (version 3 or higher) and node (^6.0.0).**
2. **Get access to the repository.** If you don't already have access, contact Michael Parker or anyone else in DEC1602 (see title page).
3. **Clone the repository:**

Install git.

In your terminal, navigate to the directory you wish to download this project into.

```
$ git clone https://github.com/Mrparkers/agridrain-app.git
```

4. **Set up the database:**

Install MySQL.

Set root password to nothing.

```
$ mysql -u root
```

```
mysql> CREATE DATABASE agridrain;
```

(Note: Database settings/credentials can be changed in `src/server/database/settings.js`.)

Navigate back to the repository.

```
$ npm install
```

```
$ npm run create-db
```

```
$ npm run create-fake-user (Note: You can run this more times for multiple users.)
```

Save the credentials of the created user and the administrator.

5. **Start the server:**

```
$ npm run build
```

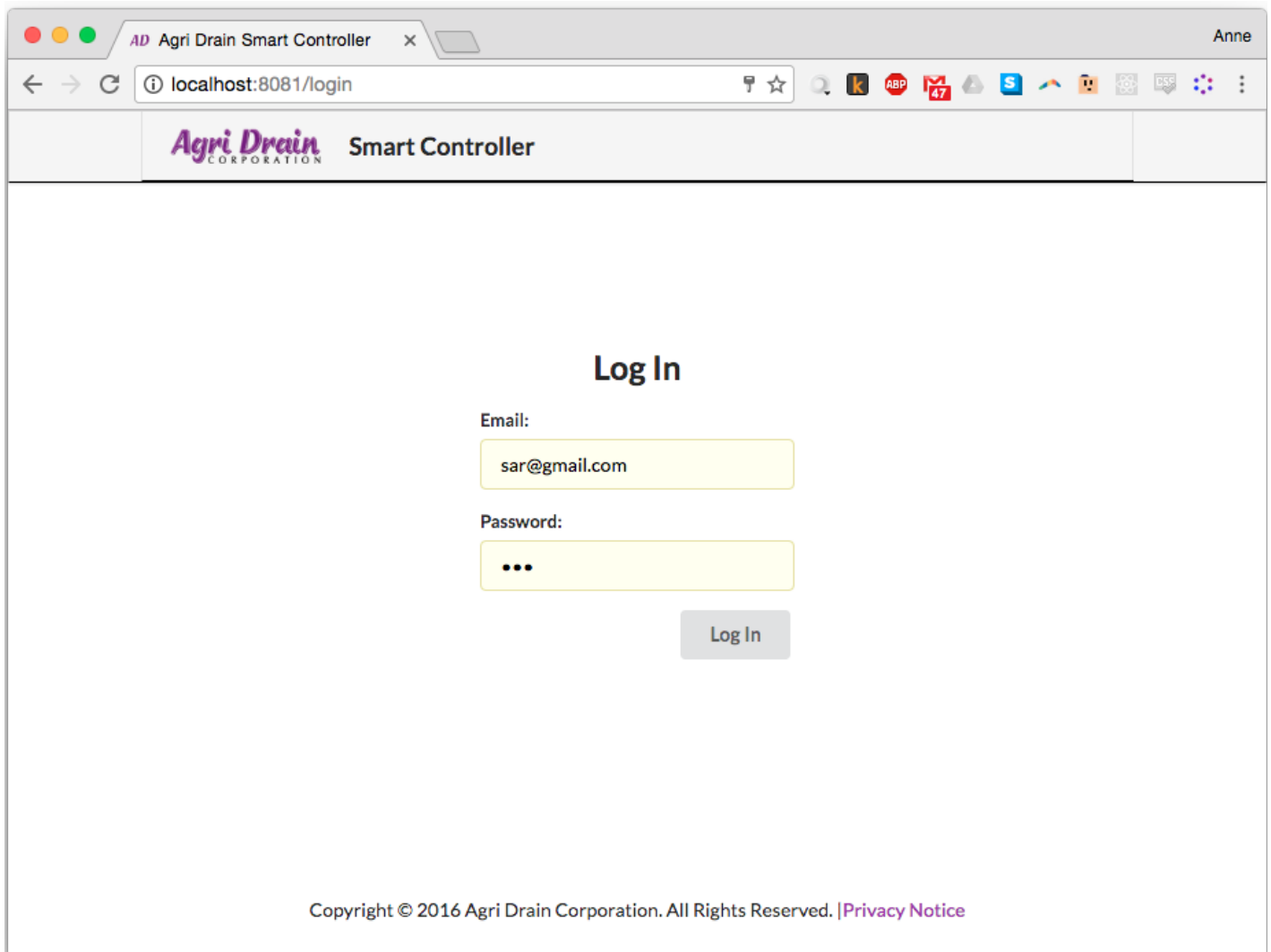
```
$ npm start
```

Navigate to localhost:8081 in your web browser.

Log in as the administrator or fake user created in the database setup stage.

# Demo

## Log in screen



Log in with admin@agridrain.com (password: admin), or log in with credentials generated for a random fake user. (In the example above, a random fake user is used.)

## Device map

The screenshot shows a web browser window with the URL `localhost:8081`. The page title is "AD Agri Drain Smart Controller" and the user is logged in as "Anne". The navigation menu includes "Smart Controller", "Users", "Settings", and "Logout".

### Device Map

- Ophion (Ames, Iowa)
- Artemis (Ames, Iowa)
- Apollo (Ames, Iowa)
- Pallas (Ames, Iowa)

**Ophion**

**Device Type:** Water Logger Device  
**Location:** Ames, Iowa  
**Set Point:** 18 in.  
[See More...](#)

Copyright © 2016 Agri Drain Corporation. All Rights Reserved. | [Privacy Notice](#)

`localhost:8081/device?id=1`

After logging in, click on any device to see more information. Click "See More..."

# Device Information

The screenshot shows a web browser window with the URL `localhost:8081/device?id=1`. The page title is "Agri Drain Smart Controller" and the user is logged in as "Anne". The navigation menu includes "Smart Controller", "Users", "Settings", and "Logout".

## Water Logger Device

Option - ID: 1

placeholder

**Technical Information**   Alarms   Graphs

**Structure Type:** Water Logger Device  
**Structure Height:** 58 inches  
**Structure Width:** 64 inches  
**Top Stop Log Distance From Top:** 2 inches  
**Top Stop Log Type:** Standard  
**Pipe Size:** 2 inches  
**Valterra Size:** 2 inches  
**Actuator Control:** 1 relay  
**Ground Level:** 5 feet above sea level  
**Minimal Drainage Time (Water Below Set Point):** 9 minutes  
**Calculate Flow:** No  
**Latitude:**  
**Longitude:**  
**Location Description:**

**Water Level Configuration**

**Set Point:** 18 inches

Copyright © 2016 Agri Drain Corporation. All Rights Reserved. [Privacy Notice](#)

There are three tabs -- in the first one, you can see technical information for the device.

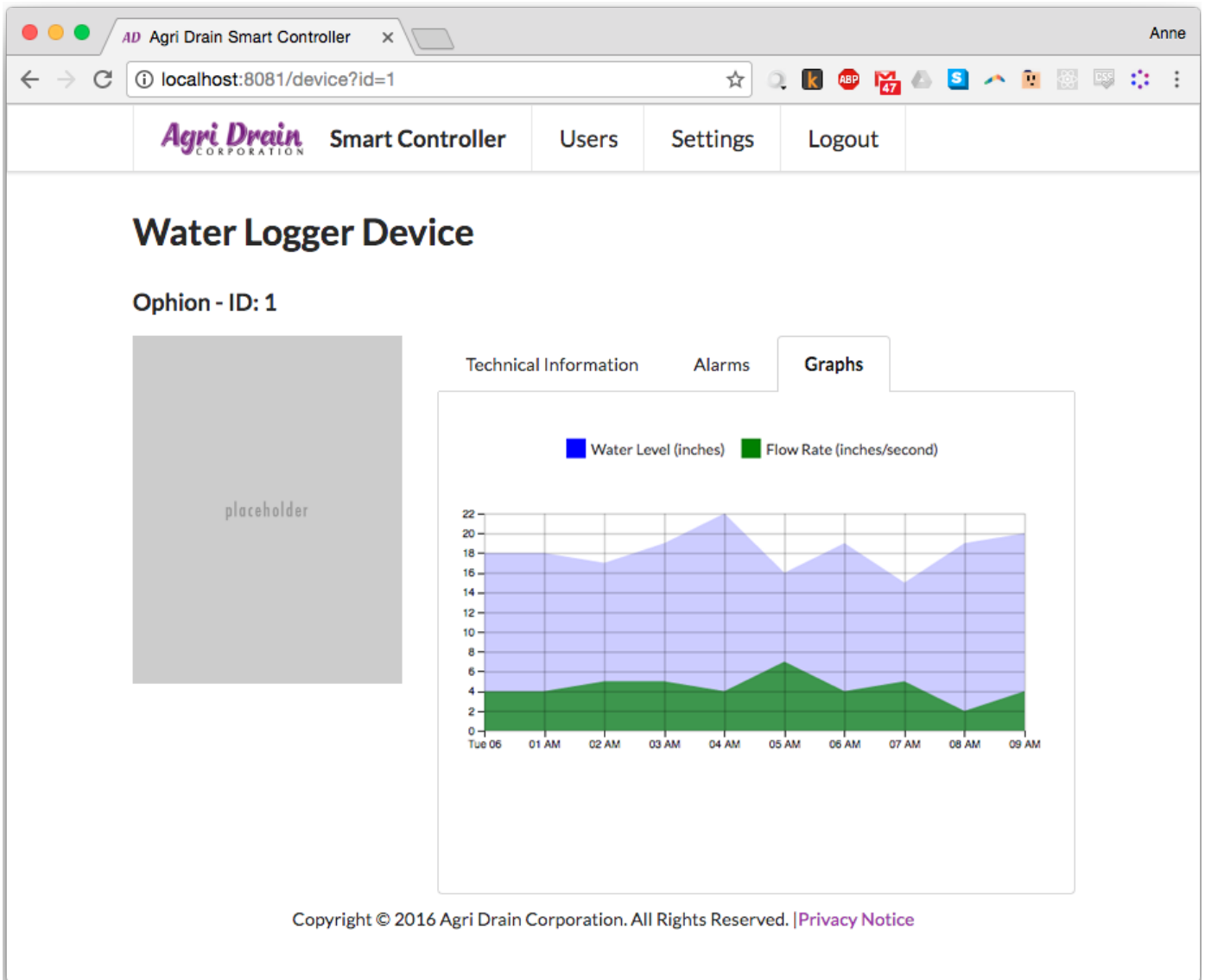
# Alarms

The screenshot shows a web browser window with the URL `localhost:8081/device?id=1`. The page title is "Agri Drain Smart Controller" and the user is logged in as "Anne". The navigation menu includes "Smart Controller", "Users", "Settings", and "Logout". The main content area is titled "Water Logger Device" and shows "Option - ID: 1". A large gray placeholder box is on the left. On the right, the "Alarms" tab is active, displaying a "Communication Failure Alarm" with the description "Sends a notification when the device has failed to communicate for over 12 hours". Under "Notification Settings", there are two radio buttons: "Text Notifications" (selected) and "Email Notifications".

Copyright © 2016 Agri Drain Corporation. All Rights Reserved. [Privacy Notice](#)

In the second tab, you can see alarms for the device. On this device, there is an alarm for when the device fails to communicate for over 12 hours. The user will receive a text message if/when this occurs, but there is an option to enable email notifications as well.

# Graphs



In the third tab, there is a graph with sensor data for the last day. This graph can be adapted to display additional sensor data our client may choose to add in the future.

# Users

The screenshot shows a web browser window with the URL `localhost:8081/users`. The page title is "Users" and the user is logged in as "Anne". The navigation bar includes "Agri Drain CORPORATION Smart Controller", "Users", "Settings", and "Logout".

## Users

[Add new user](#)

Leo Reeves	sar@gmail.com	(213) 694-1522	<a href="#">Edit</a>
Leo has <b>4 device(s)</b> . Text notifications are on. Email notifications are off.			
Chris Dean	tuiji@gmail.com	(418) 773-6164	<a href="#">Edit</a>
Chris has <b>2 device(s)</b> . Text notifications are on. Email notifications are off.			
Admin user	admin@agridrain.com	(515) 867-5309	<a href="#">Edit</a>
Admin has <b>0 device(s)</b> . Text notifications are on. Email notifications are off.			
Ethel Parker	nup@gmail.com	(526) 676-5019	<a href="#">Edit</a>
Ethel has <b>1 device(s)</b> . Text notifications are on. Email notifications are off.			

Copyright © 2016 Agri Drain Corporation. All Rights Reserved. [Privacy Notice](#)

If you're logged in as a user with administrative privileges, you will see a "Users" button in the main navigation bar. This is a list of all users in the system. You can add a new user, edit a user, or see their devices on the map view by clicking on "x device(s)".



## Create New User

The screenshot shows a web browser window with the URL `localhost:8081/users`. The page title is "Agri Drain Smart Controller" and the user is logged in as "Anne". The navigation menu includes "Smart Controller", "Users", "Settings", and "Logout". The main content area is titled "Users" and contains a "Create New User" modal form. The form has the following fields:

- First Name:
- Last Name:
- Email (required):
- Phone Number:

Below the form, there is a note: "The new user will be sent an email to set their password." At the bottom right of the modal, there is a purple "Create User" button with a checkmark icon.

Copyright © 2016 Agri Drain Corporation. All Rights Reserved. | [Privacy Notice](#)

Here's what it looks like to add a new user. They will be sent an email to set their password; admins cannot set users' passwords directly, for security reasons.

# Edit User

AD Agri Drain Smart Controller x Anne

localhost:8081/users

Agri Drain CORPORATION Smart Controller Users Settings Logout

## Edit User

First Name: Chris

Last Name: Dean

Email (required): tuiji@gmail.com

Phone Number: (418) 773-6164

### Change Password

If you want to have this user change their password, press this button to send them an email with instructions on changing their password.

Send Email

Add Devices Save ✓

Copyright © 2016 Agri Drain Corporation. All Rights Reserved. | Privacy Notice

This is what it looks like to edit a user. Note that you can add a device associated with the selected user from this screen.

# Account Settings

AD Agri Drain Smart Controller x Anne

localhost:8081/settings

Agri Drain CORPORATION Smart Controller Users Settings Logout

## Settings

First Name: Leo

Last Name: Reeves

Email: sar@gmail.com

Phone Number: (213) 694-1522

## Change Your Password

If you'd like to change your password, enter your current Agri Drain password and a new password.

Current Password: Current Password

New Password: New Password

Save Changes

Copyright © 2016 Agri Drain Corporation. All Rights Reserved. [Privacy Notice](#)

Lastly, click on the “Settings” button to edit the logged-in user’s account settings. Any user can edit their own account settings.

## Test

To verify the codebase is up to standards, run `npm run verify` in the command line. This runs ESLint, which confirms the code all conforms to the same style, and it runs all test cases. `npm run verify` only passes if 100% of the tests pass.

```
MacBook-Pro-9:agridrain-app anneore$ npm run verify
> agridrain-app@1.0.0 verify /Users/anneore/git-projects/agridrain-app
> npm run eslint && npm test && echo Verify Passed!

> agridrain-app@1.0.0 eslint /Users/anneore/git-projects/agridrain-app
> eslint src/** test/unit/**

> agridrain-app@1.0.0 test /Users/anneore/git-projects/agridrain-app
> mocha

[-----]

199 passing (2s)

Verify Passed!
```

Expect something like this when you run the tests.

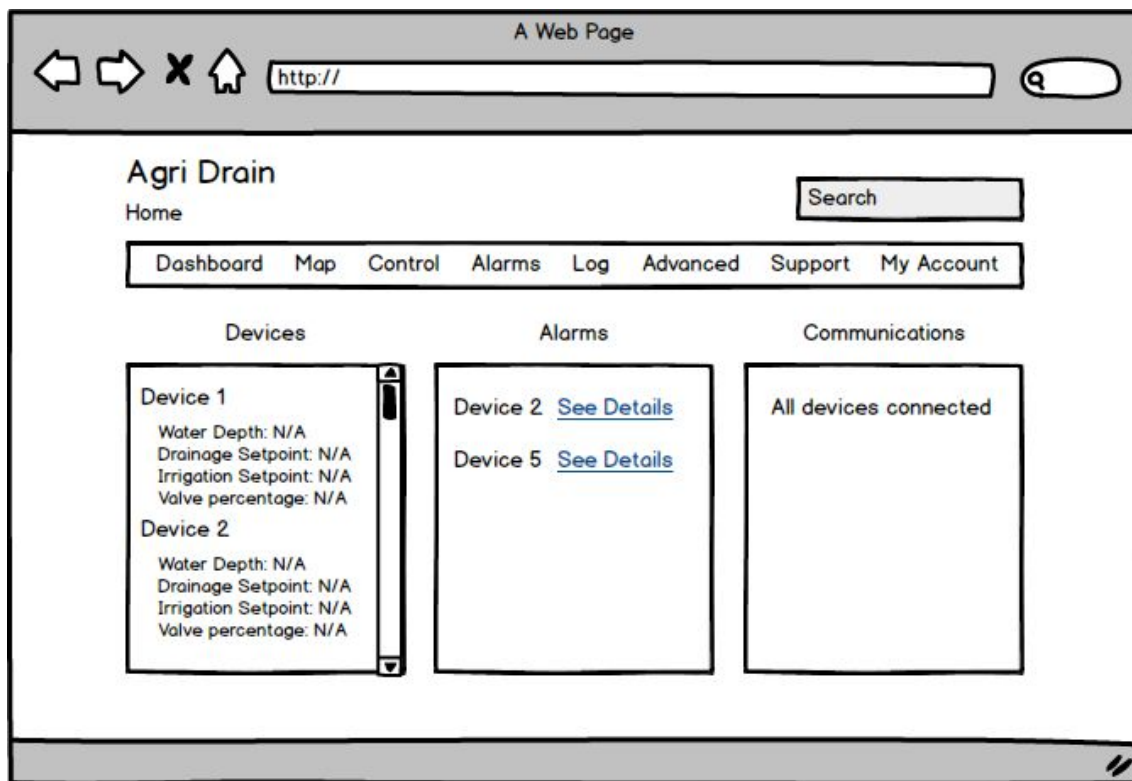
This project uses CircleCI on GitHub to ensure builds pass `npm run verify` before they are allowed to merge into the master branch. Future extensions of this project should use CircleCI or an equivalent for automatic regression testing, so that additional features cannot break old code.

# Appendix II: Alternative versions of design

## Front End

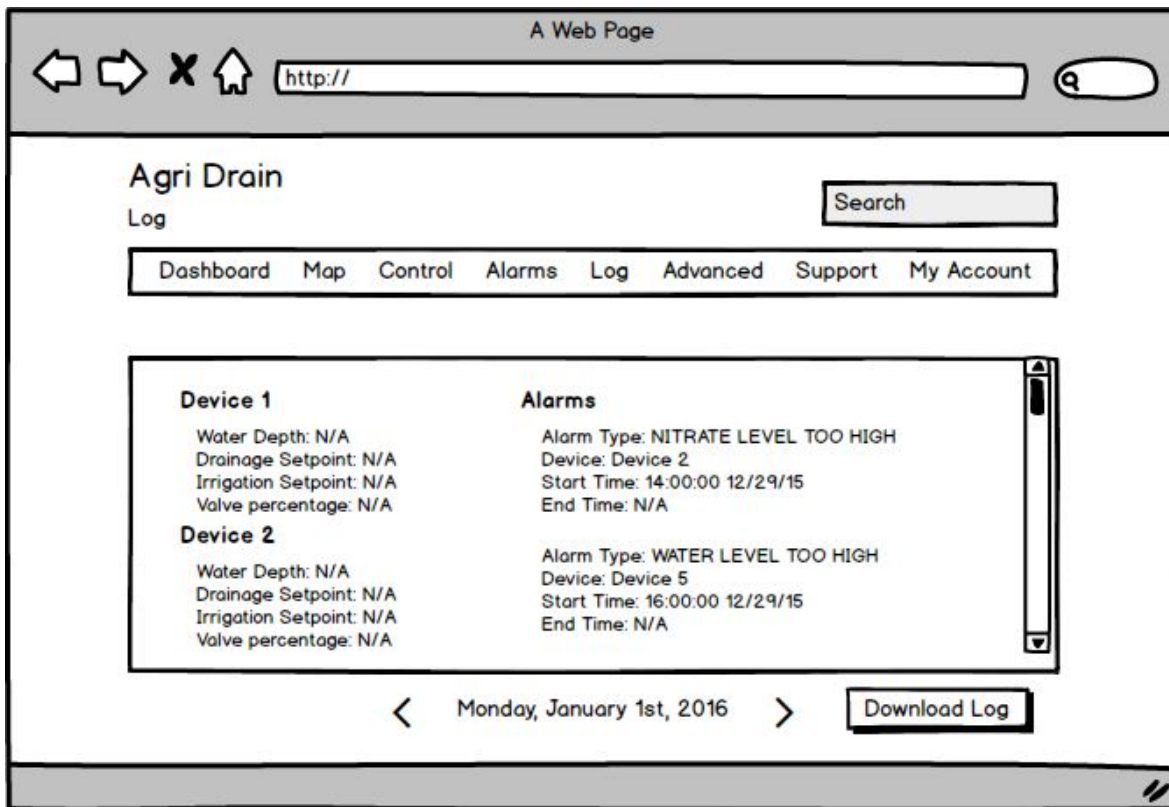
We went through a few different design ideas before reaching our final design of the web application. Here are some examples of screen sketches that we initially drafted, then scrapped for what our current user interface looks like.

### Homepage



Here is a mockup of our initial idea for the homepage. We thought that the home page would be a great place to view multiple partitions of data that the user might want to see immediately, but we later thought that would be too much information. We decided that presenting the map on the homepage would meet our users' needs best.

## Log Page



After initial meetings with our client, we thought a log page would be great for users to view historical device data and export the logs to a CSV file. After more in-depth discussions, we found that our client would prefer to view a graphical representation of the device data., This is how we arrived at the graph tab on the device page.

## Backend

As a team, we discussed how to architect our backend to meet the needs of the product. We all came from different backgrounds of web development, with different preferences for backend technologies. Here are a couple examples of ideas we had for the backend.

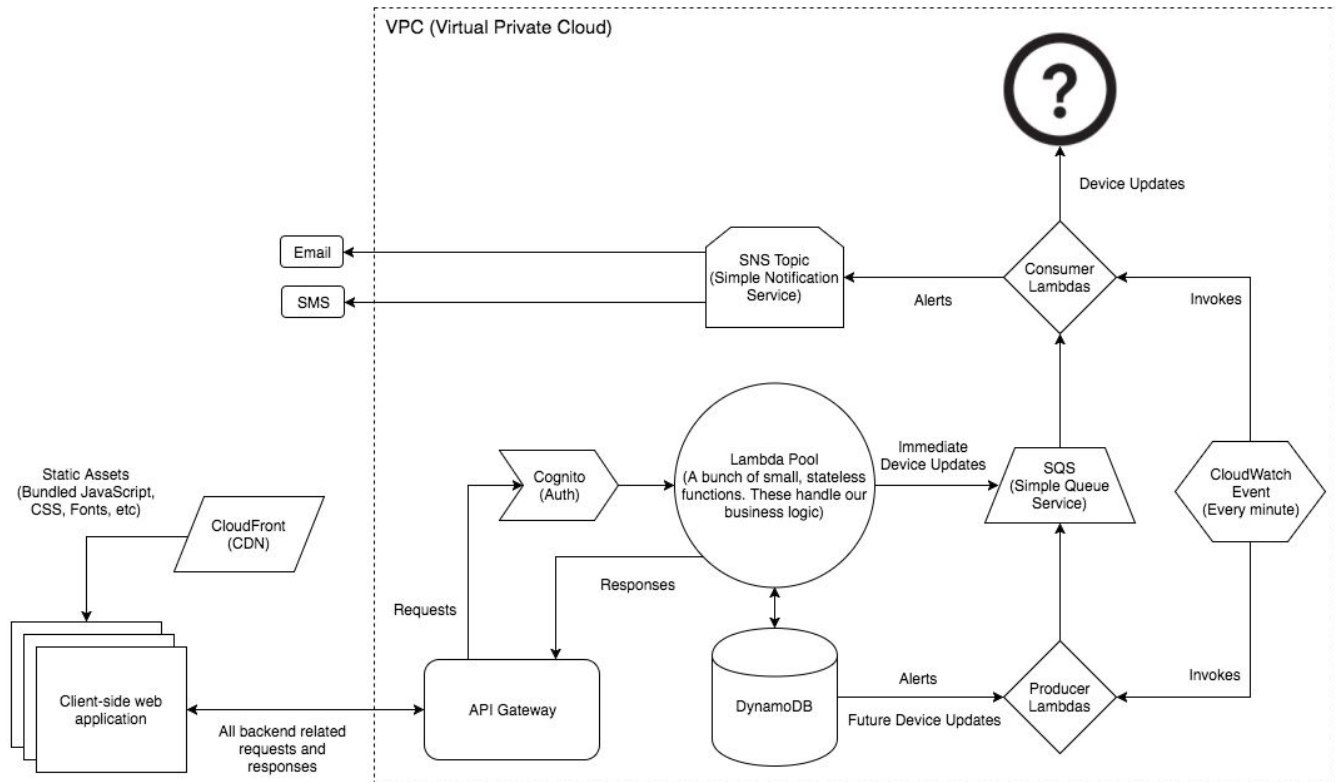
### Hapi

We first planned to use Hapi, a Node.js server-side framework, to implement the backend. However, after considering the lightweight nature of our application's backend design, we decided that Express.js would be a better framework to use for our server code, since it provides a much simpler API. Hapi is more appropriate for applications with complex backends, due its features such as its plugin system that allows for modularizing the backend code.

## Amazon Web Services

In an attempt to add complexity to our project and experiment with up-and-coming cloud technologies, we looked at Amazon Web Services (AWS) for the deployment of our app. AWS offers several different services that allow developers to architect “serverless-backends” for their applications; each service works well together to produce a single cohesive system. These services could have been utilized to provide key functionality to our product, in particular, the ability to send emails and text notifications.

Here is an outline we prepared to demonstrate what our serverless architecture would have looked like:



All the requests made from our client-sided code would have hit our backend through API Gateway, which could have been paired with Cognito to provide authentication for API calls.

Our service layer would have been handled by multiple Lambda functions, which could perform read and write operations to DynamoDB. These reads/writes could trigger downstream jobs handled by more Lambda functions, which could push items (such as a new device update) to SQS, or kick off a SNS topic to send an email or text message.

While we still believe that a serverless backend setup would have been impressive, cost effective for our client, and a great learning experience for the team, it was too complicated to implement in the time we had. We also had limited resources provided by our client: AWS is not free to use, and we did not have funding for it.

## Appendix III: Other Considerations

Our project provided a learning opportunity that has exceeded all of our expectations and has been unlike any other class project in size and scope. Each person in our group learned a great deal throughout the planning and development process that will be very valuable for our futures as software engineers. We had a good deal of success with our project, but it was not without roadblocks and problems.

In the initial stages of our project, we were focused on defining the requirements from our client. Our first meeting took a longer than expected to get planned, because of availability constraints on both our end and the client's. As students with different class schedules, work schedules, and commitments we had to attend to during normal business hours, and our client having many business obligations to attend to during business hours, it was impractical to have many meetings face to face or remotely. However, we managed to have several meetings in person and remotely despite this challenge to define our requirements and project goals. We learned to make the most of our time with our meetings and communicated over email for pressing questions that we had. Despite this, our requirements changed often between meetings, which is not uncommon for software projects, but proved to be something that we had to work through as a team.

Our advisor, Dr. Nicola Bowler, was very helpful throughout our project. She brought an extensive background knowledge of procedures and commonalities in large projects. Our team is grateful that she was always willing to review our progress, documents, and give us helpful advice on how to work with our client to create a successful project.

The choices we made for technologies posed a learning curve that potentially prevented faster and more efficient development. However, these choices also greatly increased the quality of what we created. It also allowed our members to have a learning experience working with technologies that are widely used in industry and not taught in other classes. We believe that this learning was one of the most important things that we achieved from this project.



# References

1. <http://githut.info/>
2. <http://www.modulecounts.com/>
3. <https://facebook.github.io/jsx/>
4. <http://semantic-ui.com/usage/theming.html>
5. Beck, Kent. *Test-driven Development: By Example*. Boston: Addison-Wesley, 2003. Print.