



Agri Drain Smart Irrigation Controller

Group DEC1602

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

Client: Agri Drain

Adviser: Nicola Bowler

Project Plan



Problem Statement

Farmers want to remotely:

- Control crop irrigation/drainage
- Observe water flow rates, communication issues, etc.
- See the water irrigation data represented graphically
- Visually see the location of their devices
- Be notified via text and/or email of device alarm

How do we do this efficiently?

Solution

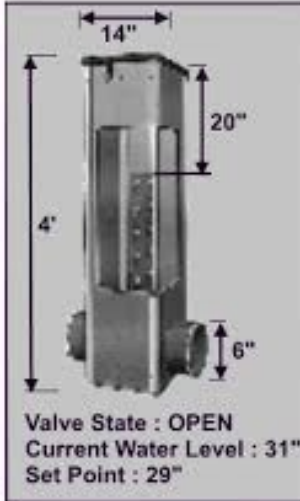
- ABE capstone group: hardware implementation
- Our team (DEC1602): web interface to connect farmer and device

DASHBOARD (REEVO RANCH)

Control Structure

Maintenance

ID : CS4-10-06-AB001



Technical Information

Structure Type : Inline - Manual
 Structure Height : 4'
 Structure Width : 14"
 Top Stop Log Distance From Top : 20 inches
 Top Stop Log Type : Standard
 Pipe Size : 6"
 Valterra Size : 6"
 Actuator Control : 1 Relay
 Ground Level : 30' Feet Above Sea Level
 Minimal Drainage Time (Water Below Set Point) : 30 minutes
 Calculate Flow : No

Water Level Configuration

Set Point : 29 inches

Above Set Point Level Information		
Level Number	Level Height (")	Wait Timer (Hours)
1	30	6
2	33	2
3	36	1
4	39	1
5	42	1

Client mockup (conceptual sketch)

Functional Requirements (User)

- The product shall...
 - Allow users to remotely read and update drainage controller settings
 - Present the user with the current device sensor data
 - Graphically present the user with historical device sensor data
 - Present the user with an interactive map that shows physical locations of their devices
 - Alert users (via text/email) when an alarm is triggered

Functional Requirements (Administrator)

- The product shall...
 - Allow administrators to add new users to the system as well as edit user information
 - Let administrators add new devices with initial configurations
 - Present administrators with all devices, and their information, within the system
 - Allow administrators to modify settings of any device within the system

Non-functional Requirements

- The product shall
 - Be functional and visually appealing on both web browsers and mobile devices
 - Have a user intuitive interface
 - Elicit useful documentation for future developers
 - Fit Criteria: Another team shall be able to pick up where our team left off with minimal communication with our team

Risks & Mitigation

- Development time constraints
- No funding from client
- Communication hardware yet to be determined by client
- User requirements inaccurately defined initially

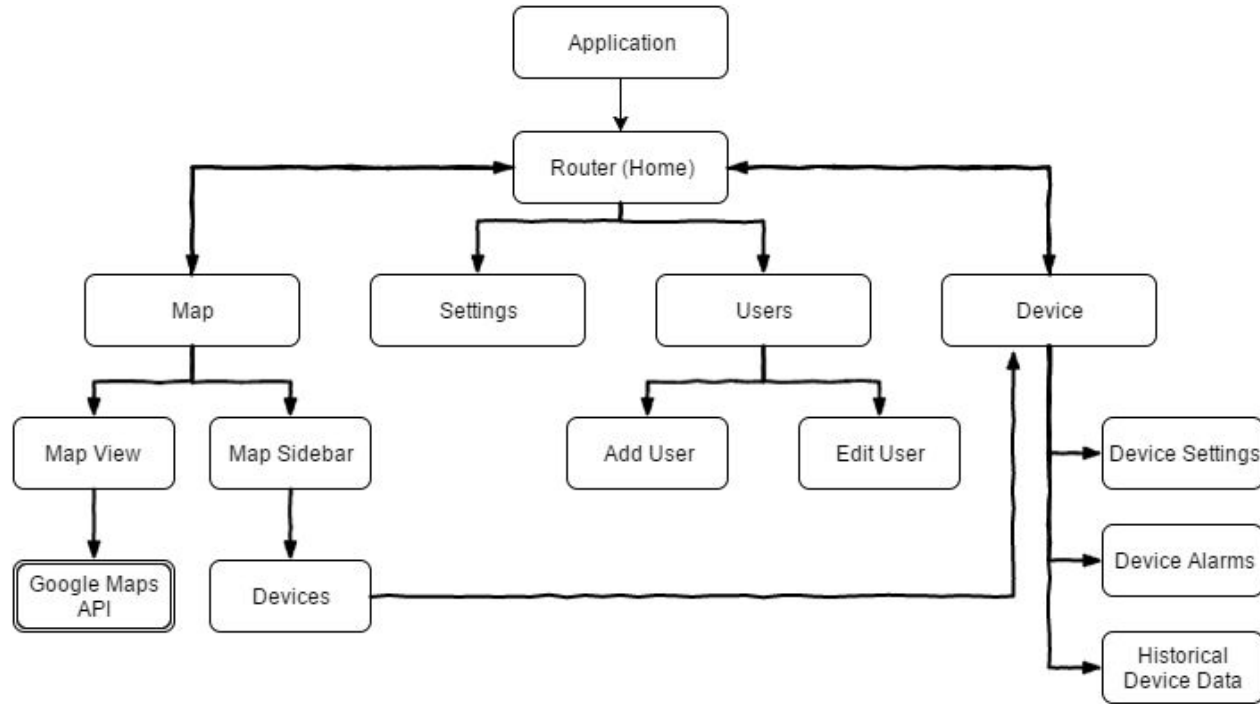
Costs

- Initially considered using paid services like Coveralls, Jira, and Travis CI
 - Decided to reject these services
 - Used free services instead (Instanbul.js, CircleCI)
- Deployment Cost Discussion with Client
 - Amazon Web Services

System Design



Front End Screen Flow



Detailed Design

- Functional Modules Design
 - Separate JavaScript modules for specific functions
 - Necessary for readability and maintenance
- Use React paradigm of creating small components
- User interface design based somewhat on Agri Drain examples
 - We believed it could be improved



Technologies Used

Node.js

- Server-side JavaScript runtime environment
- Open source and cross platform
- Event-driven, runs on single async event loop
- Highly scalable
- Used by several companies in the software industry
 - Microsoft
 - Facebook
 - eBay / Paypal
 - Netflix

React

- Library for creating user interfaces
- Emphasizes modular and component-based design
- Open source, written and maintained by Facebook
- Extremely popular and used by several companies
 - Facebook
 - AirBnB
 - Netflix

Mocha / Chai

- Testing framework and assertions library
- Behavior Driven Development (BDD)
- Tests are easy to read and write
- Allows us to test server-side hapi code and React components

Other Technologies

- Semantic UI
- SASS
- Sinon
- Webpack
- DynamoDB
- Swagger API

Testing Plan



Unit Testing

- As mentioned, we used Mocha/Chai for testing
 - Each component is tested
- We used Istanbul.js to show test coverage
 - Allows us to see statement/branch/function coverage
 - Shows exactly what is and what isn't tested

Mocha/Chai Test Report

Settings Page

- ✓ should set the initial state to display a loading indicator to the user

- ✓ should display a loading indicator until the content has loaded

Given the component has mounted to the DOM

- ✓ should make a request to the server to get the current user when mounted

Given the request to get the current user was successful

- ✓ should set the state of the component to contain the user information

Given the component state contains the user information

- ✓ should render a root DOM element with the correct markup

- ✓ should contain a Settings header

Form

- ✓ should exist with the correct markup

- ✓ should contain a Change Your Password header

- ✓ should contain a password change instructions

- ✓ should contain a Save Changes button

First and last name fields

- ✓ should exist within a field container

- ✓ should contain a first name field

- ✓ should contain a last name field

Email field

- ✓ should exist within a field container

- ✓ should contain an email field

Phone number field

- ✓ should exist within a field container

- ✓ should contain a phone number field

Phone number field

- ✓ should exist within a field container

- ✓ should contain a phone number field

Current and new password fields

- ✓ should exist within a field container







- ✓ should contain a current password field

- ✓ should contain a new password field

Istanbul.js

All files

97.86% Statements 137/148 83.33% Branches 28/24 98.72% Functions 77/78 97.58% Lines 121/124

File ▲		Statements ▾		Branches ▾		Functions ▾		Lines ▾	
client		100%	26/26	100%	2/2	100%	20/20	100%	20/20
client/components		100%	25/25	50%	2/4	100%	17/17	100%	21/21
client/pages		100%	25/25	100%	4/4	100%	14/14	100%	23/23
client/pages/device		100%	19/19	100%	6/6	100%	10/10	100%	17/17
client/pages/map		93.02%	40/43	75%	6/8	94.12%	16/17	92.68%	38/41
server/controllers		100%	2/2	100%	0/0	100%	0/0	100%	2/2

Continuous Integration

- Used Circle CI to build on every push
 - Run tests
 - Run linting tools
 - Ensure all committed code is verified

Demo

