

Kelowna Irrigation Parks Management System

By Derrick Pelletier

Honours Thesis Submitted in Partial Fulfillment of
the Requirements for the Degree of
Bachelor of Arts in Computer Science, Honours

University of British Columbia - Okanagan

April 2013

Supervised by Dr. Ramon Lawrence

Abstract

Kelowna is a fast growing city, with a large number of parks of varying size and maintenance requirements. The City of Kelowna Parks Department's current data solutions are inefficient, become outdated quickly, and are difficult to utilize. The focus of this project is to provide the City of Kelowna Parks Department with an application to augment their current systems, allowing them to better manage their data, visualize water usage trends, and manage resources. Their rich dataset contains detailed GPS data coordinates, water usage over time, detailed equipment information, and irrigation scheduling. This new application will provide the department with tools to visualize their parks on interactive maps, monitor usage trends and efficiency through interactive charts, manage their internal resources and employees, and generate reporting information for better budgeting. By facilitating greater accessibility to their current data infrastructure, the department can work towards a more sustainable future and have a positive impact on our city.

Table of Contents

1 Introduction.....	3
1.1 Motivation.....	3
1.2 Thesis Statement.....	3
1.3 Definitions.....	3
2 Background.....	3
3 System Overview.....	3
3.1 Backend.....	3
3.2 Front-end.....	3
3.3 Hosting.....	3
3.4 Visual Walkthrough.....	3
4 User Manual.....	3
4.1 Sign in.....	3
4.2 Main screen - listing.....	3
4.3 Park Details.....	3
4.4 Maps.....	3
4.5 Reports.....	3
4.6 Alerts.....	3
4.7 Employees.....	3
4.8 Tools.....	3
4.9 Mobile List.....	3
4.10 Mobile Maps.....	3
4.11 Mobile Alerts and Employees.....	3
5 Code Elaboration and Explanation.....	3
5.1 Database.....	3

5.2 Backend..... 4

5.2.1 Models 4

5.2.2 Controllers 4

5.2.3 Views 4

5.2.4 Mobile detection 4

5.3 Frontend 4

5.3.1 Application 4

5.3.2 Modules 4

5.3.3 Templates 4

6 Feedback..... 4

7 Conclusions 4

8 Future Work 4

9 Acknowledgements..... 4

10 References 4

1 Introduction

1.1 Motivation

Kelowna Parks Department monitors several hundred parks throughout the Kelowna area and they inherently track a very rich dataset that is, unfortunately, underutilized and undervalued. Having this information organized effectively, automated where necessary, and accessible to all employees, will provide a great opportunity to effectively utilize their vast data resources.

By providing the department with the tools to organize, visualize, and analyze this data, it opens up opportunity to reduce water usage, better allocate resources, personnel, and ultimately

facilitates a more sustainable future for the Parks Department. Furthermore, should the application prove a viable solution to the growing needs of the irrigation department, the system will be developed with location-flexibility in mind, so that it can be adapted to the needs of other markets beyond the Kelowna Parks Department. Not only can we help make an impact on sustainability in our city, but we can facilitate the same goals in other cities as well.

1.2 Thesis Statement

My thesis is that this system will improve the sustainability of the Kelowna Irrigation systems. Our system is a robust and modular tool that allows the city to adapt it to their needs efficiently as they discover new ways to utilize their data. My goal is to develop a system that not only meets their needs in the current state of data understanding, but also allows them to maintain their trend of further technological advancement by not limiting their possibilities.

1.3 Definitions

Users are parks employees and can consist of managers, on-site workers, and administrative staff members.

The **Application** is generally used to reference the primary web application developed for this honours thesis. This is an abbreviated form of “Kelowna Irrigation parks and water management application.”

View is typically used to define each different section of the application. In traditional websites, this would be considered a page, but within the context of an application, it will be called a view. View may also be used in the context of a class of object found in Backbone.js.

2 Background

UBCO has worked with the City of Kelowna for the past several years. In 2011, UBCO computer science graduate students developed and installed moisture sensor nodes in a park in order to track irrigation patterns with the goal to reduce water usage. The cost of installation and

maintenance proved too high for the payoff of this technology. The City of Kelowna opted to pursue an alternate means of technology advancements that could have a more immediate impact on the members and resources of the parks department. In 2012, work began on approaching their existing, and outdated, data collection system with the hopes of developing an application that could be used both in-office and in the field and provide immediate efficiency payoffs, as well as long-term payoffs as water usage analysis and resource management was made more approachable. In the summer of 2012, graduate students collected sample data on some parks, including mapping GPS coordinates, and prototyped interfaces for this project.

3 System Overview

3.1 Backend

MySQL – All data is now stored and managed in a MySQL database. MySQL is an open-source, relational database system that provides the city with the ability to store varying types of interrelated data in a logical manner. MySQL is tightly integrated with PHP and CodeIgniter.

PHP - The backend of the Application is entirely developed in PHP 5.x. This is a fully featured open-source, server-side language with an extensive developer community, stability, and maturity. This language was chosen over other, modern, and perhaps more cutting-edge, server-side languages due to the ease of use for all developers. Selecting a language which future developers can easily carry on developing without having to learn a new, difficult, and perhaps under-documented system in order to make future maintenance modifications or develop new features, increases the potential for this application to remain in active development and eventually succeed in the goal of providing the city with the tools to see long-term change in their department systems.

CodeIgniter - A lean MVC application framework designed to be flexible to and provide a very lightweight structure to applications. The simplicity CodeIgniter provides compared to more rigid systems, allows new users to adapt quickly without being forced to learn a robust and strict

framework before being productive. We initially started with Symfony2, a mature, Java inspired framework. However, it proved too heavy to support multiple developers of varying skillsets.

3.2 Front-end

HTML5 - Rather than develop a native application for a particular mobile application ecosystem, or develop a mobile application dependent on a pre/cross compiler, we chose to develop the Application as a mobile, HTML based application which only depends upon having a browser available and is thereby capable of being used on any number of devices.

Bootstrap - This library is developed by Twitter and provided as open-source. It provides numerous style provisions to give developers the ability to style web applications quickly and easily. This allowed us to prototype at a quick pace while simultaneously styling the application cleanly and effectively as to convey the goals and intentions of the software efficiently.

Backbone and Marionette - This JavaScript application framework combination allows developers to build rich, HTML based applications with data persistence, and native-application feel on both desktop and mobile browsers.

3.3 Hosting

The web technology stack chosen for the Application has very few requirements in hosting dependencies and therefore is flexible in both the chosen development and production environments. City regulations call for specific details of the host's operation (for example, it must be Canadian) so keeping the stack simple and mostly dependency-free affords the ability to keep the Application nearly server-agnostic.

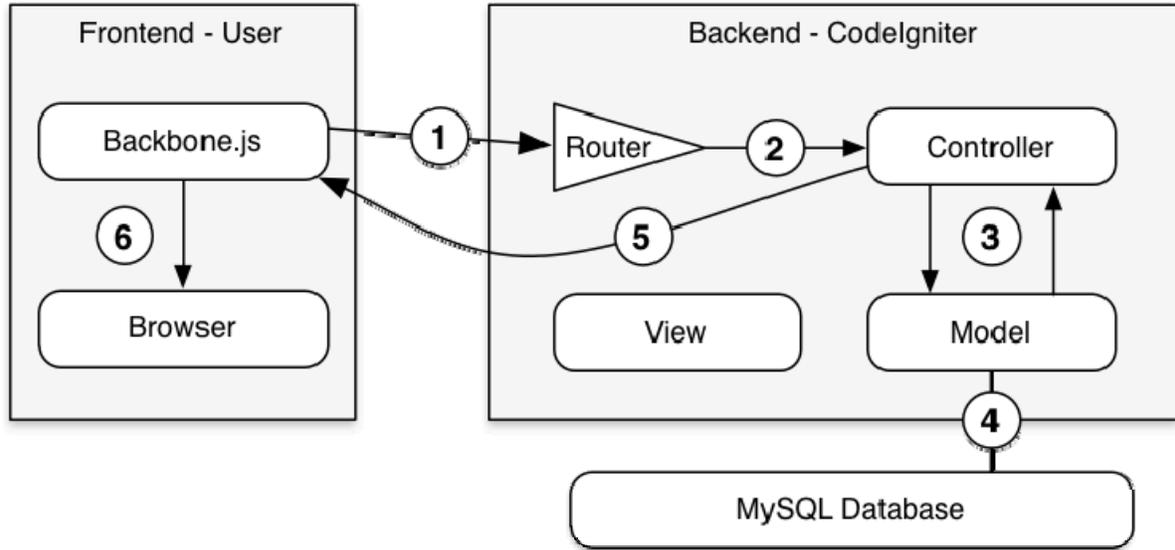


Figure 3.a - Application request/response flow

3.4 Visual Walkthrough

Step 1 - User clicks to view the details view of a park. This causes Backbone to initiate a request to the server for the park data to display. This request, in this case, the URL `http://domain.com/api/parks/14`, goes to the CodeIgniter Router.

Step 2 - The Router parses the URL to determine which Controller to use. Normally this URL would map to an `Api.php` controller with a `parks()` method. However, due to the size of the API, we have overridden this behavior, and it would check the `controllers/api` directory for a `Parks.php` file, and call the `byId(14)` method.

Step 3 - `byId()` will then determine that it requires a set of site data from Model: `sitesmodel.php`. This model contains a similar method, `byId()`

Step 4 - `SitesModel::byId()` generates an SQL query through the ActiveRecord class, queries the database, and then returns the data result back to the model. The model then validates the returned data is correct, and returns either the data or `false` to the Controller.

Step 5 - The Controller formats the returned data as a JSON-encoded object and generates a response which is sent to Backbone.

Step 6 - Backbone parses this data, inserts it into the appropriate frontend models, generates some HTML templates based on the data, and finally shows them to the user.

4 User Manual

The following steps will walk through the core aspects of the application, providing a quick understanding of the available features. Implementation details are discussed in Section 5.

4.1 Sign in

The Application may contain sensitive employee information or incorrect data, and as such exists entirely behind an authentication wall. Employees sign in with their department email address and password.



The screenshot shows a sign-in form for the City of Kelowna. At the top, the logo consists of a colorful circular pattern followed by the text 'City of Kelowna'. Below the logo, the text 'Please sign in' is displayed. There are two input fields: the first is for the email address, containing 'derrick@dpelletier.com', and the second is for the password, containing a series of dots. Below the password field is a checkbox labeled 'Remember me'. At the bottom of the form is a blue button with the text 'Sign in'.

Figure 4.a - Application Sign-in view

4.2 Main screen - listing

Upon signing in, the first screen is the main listing. This view lists all parks currently in the database. We determined that this listing is the most commonly accessed information by parks staff, and therefore should be accessible the quickest. The listing provides some core details, such as the location, the irrigated area, the type of park, supplier information, number of outstanding alerts, and the year to date usage.

ID	Site Name	Location	Irrigated Area	Type	Water Supply	Alert	YTD Usage
734	Abbott Street Boulevard	2178 Abbott Street	0.06	Boulevard	CITY		
728	Abbott Street Park	2955 Abbott Street	0.18	Beach Access	CITY	0	
011	Anchor Park	1691 Ellis Street	0.07	Green Space	CITY		225.025806451613
652	Anhalt Road Cul-de-sac	Anhalt Road	0.01	Cul-de-sac	CITY		
613	Apple Bowl	1800 Parkinson Way	2.31	Sports Field	CITY		
061	Art Walk (North Spine)	1355 Water Street	0	Beach Access	CITY		
39	Athans Court Cul-de-sac	Athans Court	0.01	Cul-de-sac	CITY		25.1784946236559
240	Augusta Court Cul-de-sac	Augusta Court	0.01	Cul-de-sac	CITY		
263	Avonlea Park	274 Avonlea Way	0.19	Park	CITY		764.183870967743
445	Bach Road Cul-de-sac	Bach Road	0.01	Cul-de-sac	BMD		
041	Bankhead Green Space	1400 Bankhead Crescent	0.32	Green Space	CITY		
503	Barlee Community Gardens	1898 Barlee Road	0.99	Garden	CITY		
517	Baron Road Boulevard		0.19	Boulevard	RWD		
697	Barrera Recreational Corridor		0.69	Green Space	CITY		4165.21505376344
905	Belgo Park	895 Belgo Road	5.27	Park	RWD		
647	Bellevue Creek Park		0	Green Space	CITY		
649	Bellevue Creek Trail	703 Vance Avenue - 781 Varney Court	0.24	Green Space	CITY		23.9838709677419
687	Belmont Park	4444 Belmont Road	1.89	Park	CITY		3013.9390807879
467	Ben Lee Park	900 Houghton Road	14.8	Park	BMD		
	Bennett Bridge Southside Linear Park		0	none	CITY		
071	Bennett Memorial Clock	1435 Water Street	0	Beach Access	CITY		
437	Benson Court Cul-de-sac	Benson Court	0.01	Cul-de-sac	BMD		
513	Benvoulin / Cooper - Median/Boulevard		0.61	Median	CITY		
047	Bernard Avenue Nodes		0	Node	CITY		
575	Birkdale Park	363 Prestwick Street	1.42	Park	BMD		

Figure 4.b - Main Sites listing view

Above the listing is the filter bar. Clicking “display filters” will expose a panel with several filtering options. On the right end of the filter bar is a live query box. Typing in this box triggers a filtering option that reduces the dataset by comparing the site names against the search text.

Manipulating these options will reduce the list view in order to navigate to desired results quickly. Additionally, this reduced dataset can be used to create custom water usage charts simply by clicking “chart” in the filter bar.

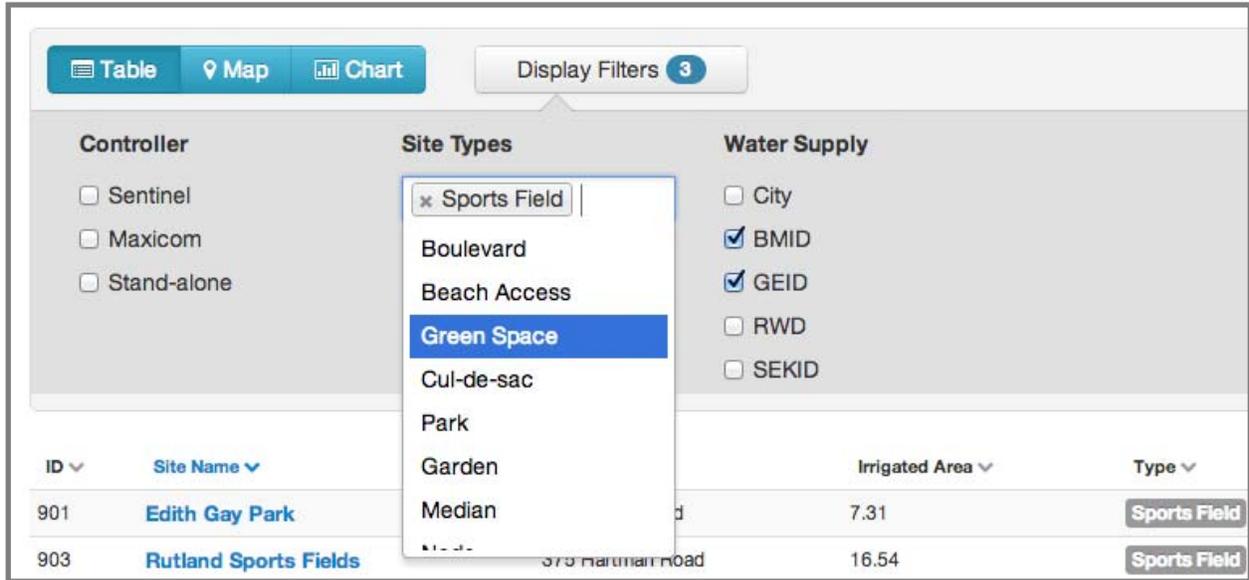


Figure 4.c - Closeup of main listing filtering options

4.3 Park Details

Clicking a row in the main listing will bring up the Site Detail view. This view is essentially an aggregation of all data relating to a particular site, split into four categories (as represented by the tabs along the top): site details, alerts, maintenance logs, and reporting charts. The first row consists of a small map of the location, dynamically generated by GPS coordinates, as well as a listing of the core details of the site. These can be edited directly in place by clicking the edit details button. Following this is a list of the employees responsible for this site, as well as a listing of controllers, meters, and points of connection with details for each item. Clicking edit on a piece of equipment brings up a modal window in which the equipment can be edited instantly.

The screenshot displays the 'Site Details' view for Mission Recreation Park (MRP). At the top, there are navigation tabs for 'Site Details', 'Alerts', 'Maintenance Logs', and 'Reporting Charts'. The 'Site Details' tab is active. On the left, a map shows the location of Mission Recreation Park, with labels for 'Mission Dog Park' and 'Mission Recreation Park'. The map data is attributed to Google. On the right, the 'Edit Details' button is visible above a list of site attributes:

- Data ID: 184
- Site Name: Mission Recreation Park [MRP]
- Site ID: 915
- Location: 4105 Gordon Drive
- Irrigated Area: 31.79 acres
- Zones: 0
- Water Supply: CITY
- YTD Usage: 1568473.99143591
- Category: Sports Field

Below the details panel, the 'Assigned Employees' section lists two employees:

- bob_johnson (seasonal)
- derrick (admin)

The 'Controllers' section contains a table with the following data:

ID	Serial	Nickname	Model	Version	Zones	Available Stations	Channel	Power Source	Company	Type	
212	775592	North	ESP-24SAT	0		24	1	0	Rainbird	Maxicom	edit
213	01259	North	ISC-SAT-12B	0		12	2	0	Rainbird	Maxicom	edit
214	00785	North	ISC-SAT-12B	0		12	3	0	Rainbird	Maxicom	edit
215	00799	North	ISC-SAT-12B	0		12	4	0	Rainbird	Maxicom	edit
216	1343461	North	ESP-24SAT	0		24	5	0	Rainbird	Maxicom	edit

Figure 4.d - Site expanded detail view

The **alerts** tab brings up a view of all of the alerts currently outstanding for the chosen site, as well as details for those alerts and which employee they are assigned to.

The **maintenance** tab is intended to contain a record of maintenance for the particular site. This section is currently unimplemented and is placed here as concept only.

The **reporting charts** tab initializes an interactive charting tool. By default it will display all water usage for the park currently in the database. Users have the ability to filter the results according to various options to generate appropriate visualizations for their needs.

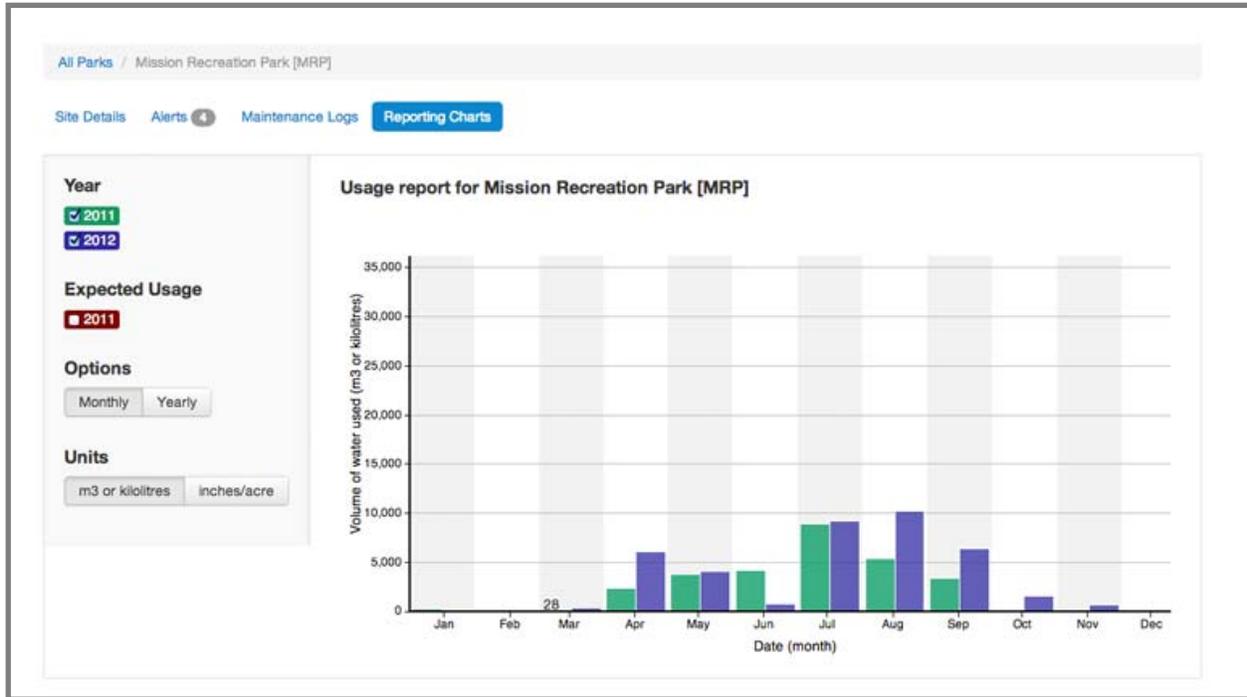


Figure 4.e - Site Detail water usage chart

4.4 Maps

The Maps view of the application displays the exact dataset as the main listing view, albeit in a map format. The filters from the main listing persist across to the map view as well, allowing the user to toggle between views in order to best meet their needs. Each park is represented by a pin with the park's unique ID as well as a polygon representing park's boundaries.

Kelowna Irrigation Parks Management System

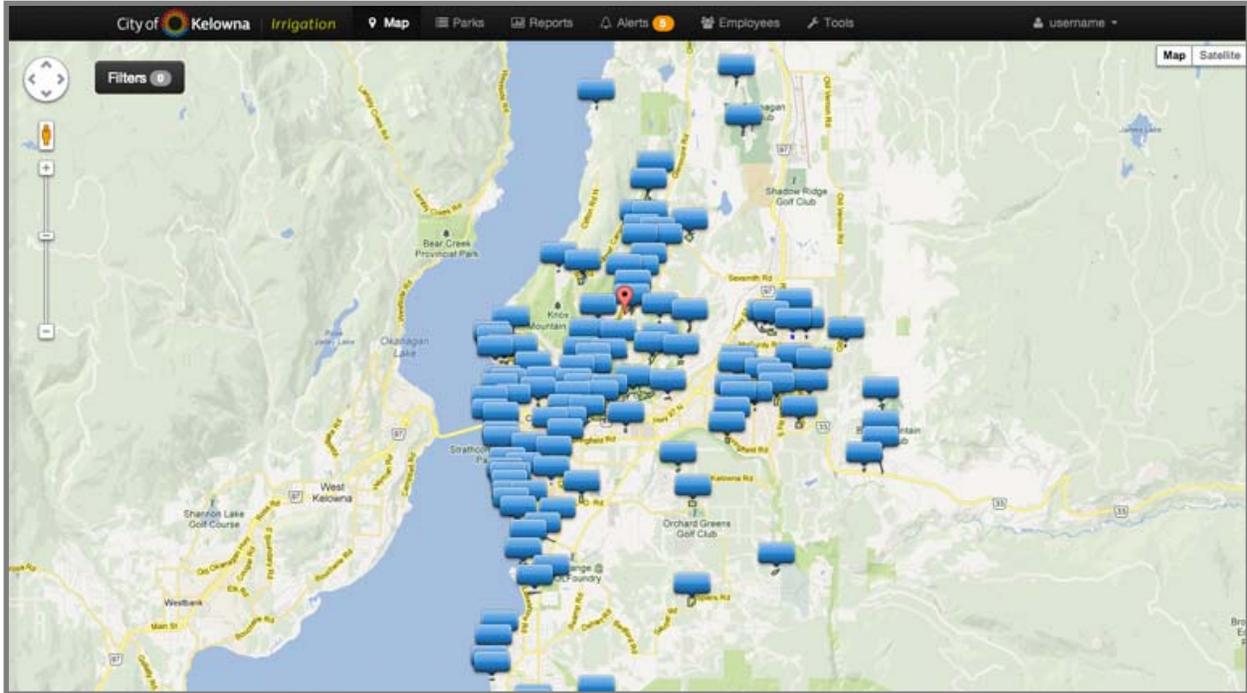


Figure 4.f - Site Map view

Clicking on a pin or park boundary will zoom in on that park, slide open a details panel, and display associated equipment on the map. The details panel contains only the core information that is required most often, including: location information, essential site details, recent park alerts, and assigned employees. Clicking the title in this panel takes the user to the extended detail view previously discussed.

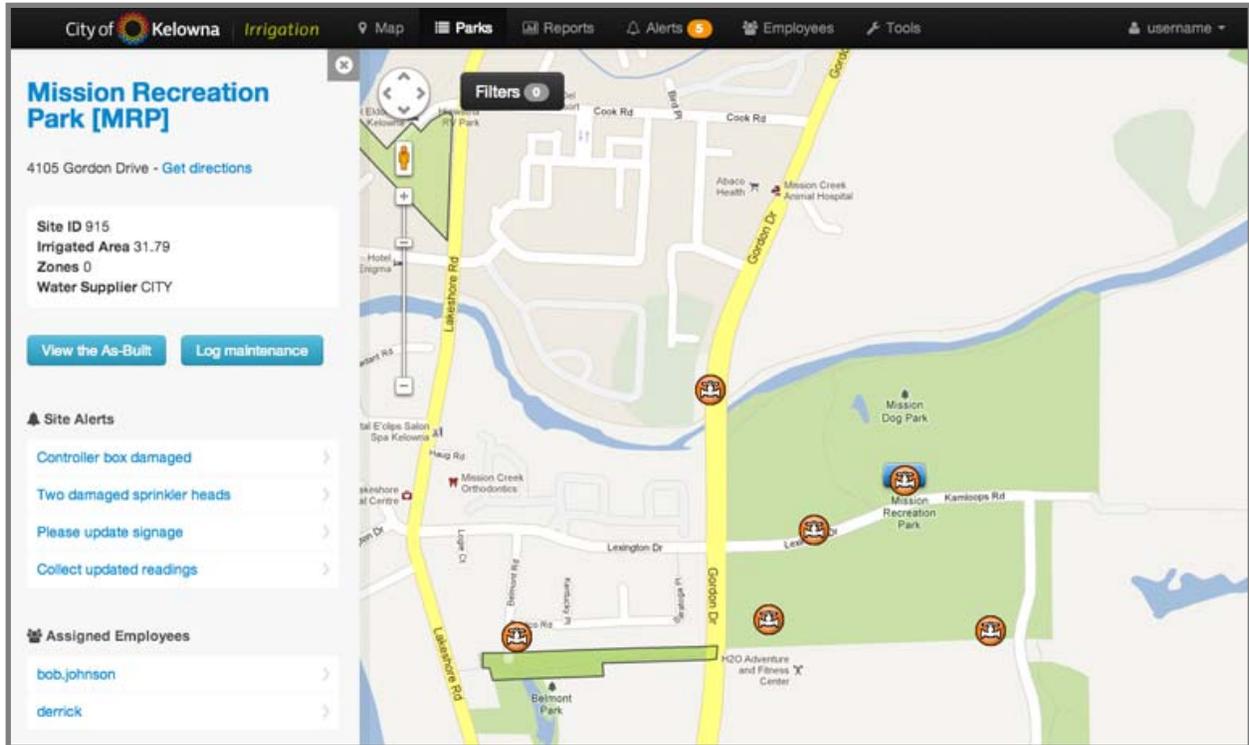


Figure 4.g - Site Map view with expanded site details panel

4.5 Reports

The reports view of the application is intended to give managers a quick overview of the parks system. Different visualizations of data can be accessed through a tab system. The charts are completely independent from one another, and as such, intractability will vary. This view is currently primarily in conceptual phase, and is only listing the Top Ten Water Users. As the Parks Department discovers analytical needs, this space will be adapted and expanded to meet them.

4.6 Alerts

The alerts view is the central location where the current user can view all alerts assigned to them. Alerts are formatted in a standard email style arrangement, with a listing and truncated body in the left column, and the expanded click-through view in the right panel.

Each alert can have any of the following properties: a title; an assignee; an associated park; an alert type, such as warning or info; and a message body. Users have the ability to complete or delete an alert in the detailed panel. Completed alerts remain accessible via the archive button in the main toolbar for this view. The alerts system was an exploratory attempt to remedy a management system that relied on a single employee to manage all communications. Each employee is tasked to deal with these communications in their own manner, which often involves note taking and little archival attention.

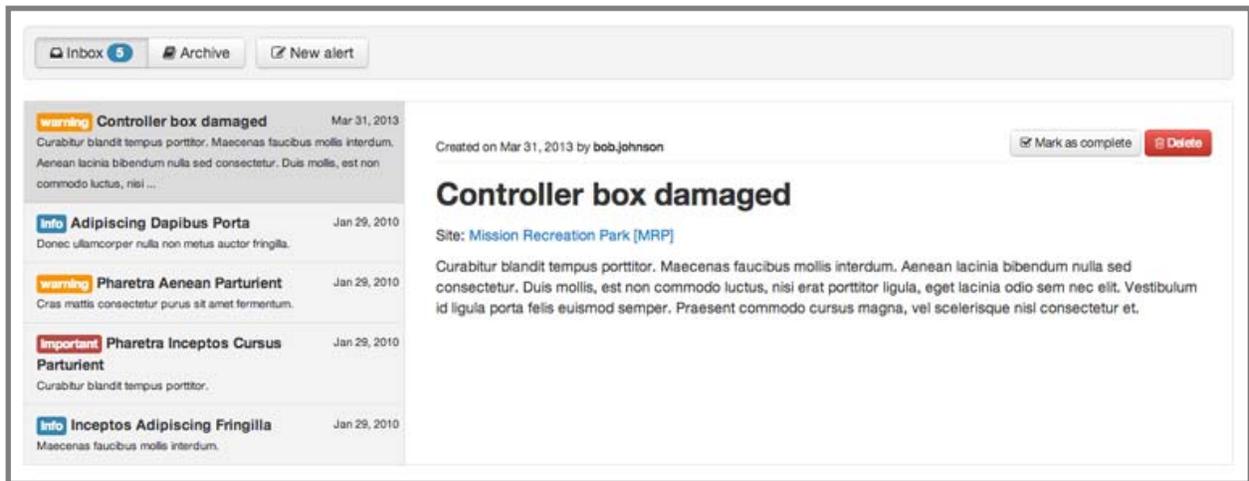


Figure 4.h - Main Alerts view

Clicking New Alert will bring up a form where a user can submit an alert to other users. This form has the ability to quickly select a site to associate, as well as to assign the alert to multiple users in one message.

4.7 Employees

The employees view is a listing of all employees registered in the system. Administrators can assign employees to sites and edit general employee information. Standard users can utilize this view as a lookup for contact information. Currently this page is undeveloped due to the exact requirements still being determined.

4.8 Tools

From the tools view, employees and administrators can manage application-wide data. Application settings can be accessed, new sites can be added, and minor options such as employee types and alert types can be modified here. The majority of this system is undeveloped due to the application not being finalized



Figure 4.i - Tools options

The readings tool is currently the only one in active development. Here an employee can import new water meter usage data in CSV format into the database, as well as access previous imports in order to edit inconsistencies or incorrect readings.

Tools / Past Imports			
<input type="button" value="Import a CSV file"/> <input type="button" value="Upload"/>			
Import Date	User	Filename	Ro
Feb 10, 2013 at 9:33 AM	derrick	test.csv	30
Feb 6, 2013 at 22:13 PM	derrick	august.csv	0

Figure 4.j - Past Imports listing view

Once a user imports a new CSV or selects a previous import, they are presented with a view that includes details as to the employee who imported, the filename of the CSV, and two tabs which house both clean rows and flagged rows. Flagged rows are those that, on import, were dynamically flagged with various messages based on rules in our import system. If a reading fails to pass these tests, it remains inaccessible to usage data queries in the main application. A

user can view these flagged rows, and edit them in place or remove them. Once edited, these rows are tested again, and if passed will be considered clean and appear in the standard usage results.

Tools / Past Imports / Feb 10, 2013 at 9:33 AM

Upload date: Feb 10, 2013 at 9:33 AM
 Filename: test.csv
 Uploaded by: derrick

Clean Rows 21 **Flagged Rows 0**

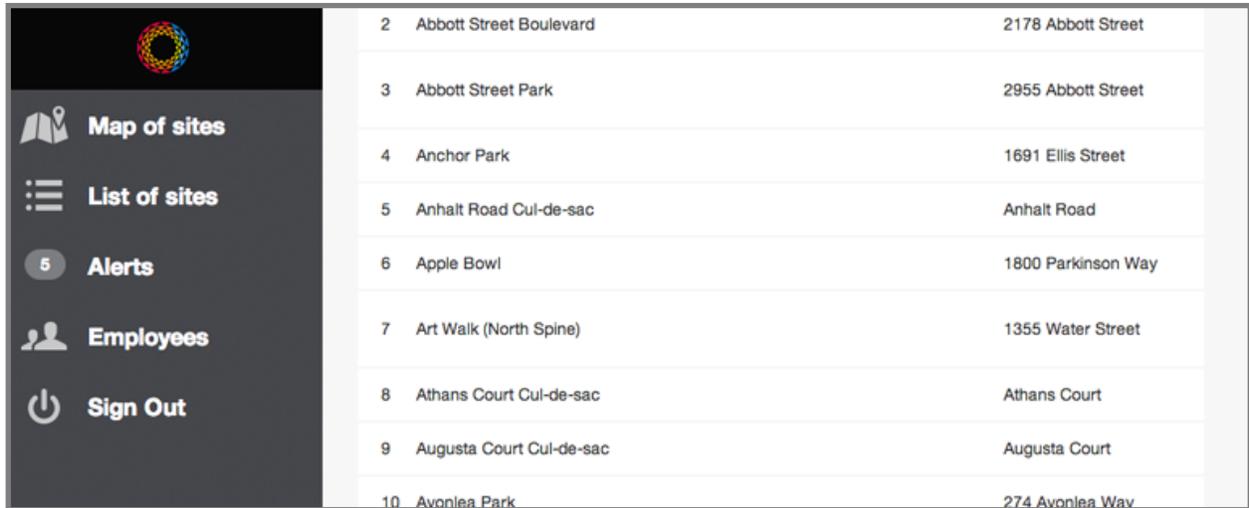
ID	Read Date	Reading Value	Read By	Meter	
1725	2012-06-26 00:00:00	965	Corix	58290779	Save Cancel
⚠ This data is clearly wrong - remove flag					
1726	2012-06-16 00:00:00	34343	Corix	69076187	Edit ⓧ
⚠ This is a warning - remove flag					
⚠ Donec ullamcorper nulla non metus auctor fringilla. - remove flag					
1727	2012-06-08 00:00:00	9103	Corix	65557829	Edit ⓧ
⚠ Sed posuere consectetur est at lobortis. - remove flag					
1728	2012-06-26 00:00:00	52657	Corix	62742918	Edit ⓧ
⚠ Maecenas faucibus mollis interdum. - remove flag					
1729	2012-06-14 00:00:00	52151	Corix	65707074	Edit ⓧ
⚠ Cras mattis consectetur purus sit amet fermentum. - remove flag					
1730	2012-06-07 00:00:00	24385	Corix	69076132	Edit ⓧ
⚠ Aenean lacinia bibendum nulla sed consectetur. - remove flag					
1731	2012-06-07 00:00:00	45826	Corix	62312251	Edit ⓧ
⚠ Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. - remove flag					
⚠ Aenean lacinia bibendum nulla sed consectetur. - remove flag					
1732	2012-06-14 00:00:00	81861	Corix	65707075	Edit ⓧ
⚠ Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. - remove flag					

Figure 4.k - Usage Import details view of flagged alerts, with first entry being edited

4.9 Mobile List

The mobile version of the Application runs on a shared codebase with the main application. There are some minor tweaks to functionality in various instances, but primarily relies on utilizing mobile friendly templates in order to style the application for a mobile, tablet-friendly application experience.

The main site list, for example, has stripped away many of the extra detail columns in order to reduce clutter and increase text and tap-size. This allows employees in the field to quickly scan and filter the list without worrying about extraneous information. By tapping a row, the application will load a similar detail view as that in the desktop application, albeit without editing functionality.



2	Abbott Street Boulevard	2178 Abbott Street
3	Abbott Street Park	2955 Abbott Street
4	Anchor Park	1691 Ellis Street
5	Anhalt Road Cul-de-sac	Anhalt Road
6	Apple Bowl	1800 Parkinson Way
7	Art Walk (North Spine)	1355 Water Street
8	Athans Court Cul-de-sac	Athans Court
9	Augusta Court Cul-de-sac	Augusta Court
10	Avonlea Park	274 Avonlea Way

Figure 4.1 - Mobile Application main site listing

4.10 Mobile Maps

The mobile map view functions identically to the desktop map view. This view was initially developed for speed and ease of use on mobile, and this functionality was ported to the desktop version in order to maintain consistency and deliver the same benefits of brevity and browsability. By providing field employees with quick access to the core details of a park, as-built overlays, and equipment locations, they can be much more efficient and knowledgeable in the field rather than relying on outdated and unreliable datasets.

4.11 Mobile Alerts and Employees

The mobile alerts and employees views function identically to the desktop counterparts with some minor style variations to better suit touch interfaces.

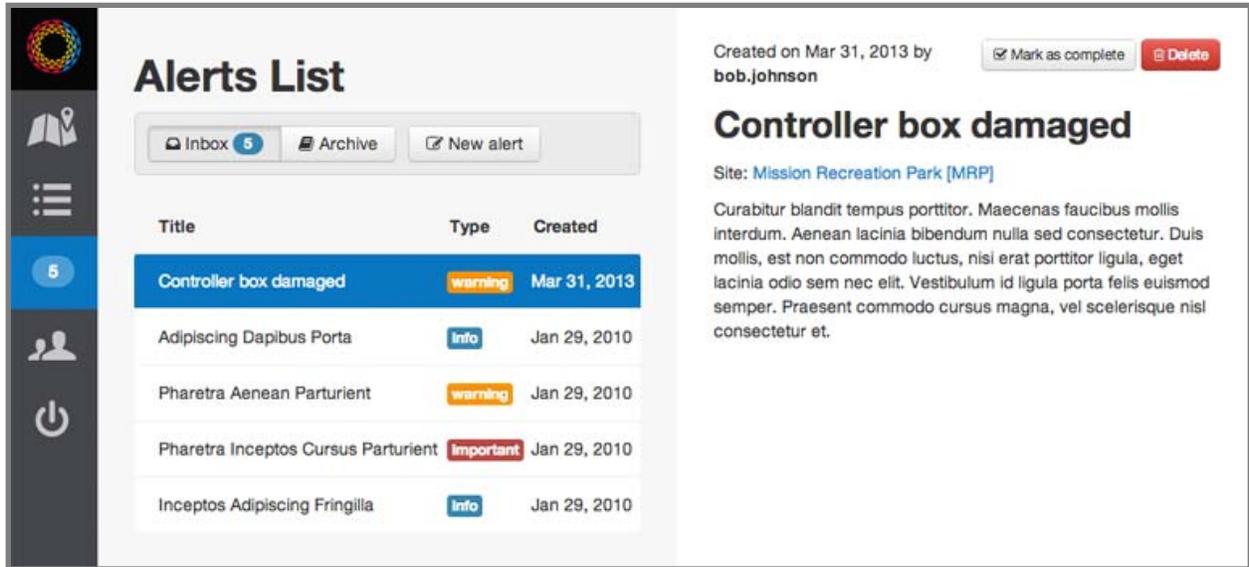


Figure 4.m - Mobile Application Alerts view

5 Code Elaboration and Explanation

5.1 Database

The Application utilizes a MySQL 5.x relational database. MySQL was selected due to its maturity, stability, and easy entry point for new developers. MySQL is a de facto component of the standard web application LAMP (Linux, Apache, MySQL, PHP) architecture, and as such, system configuration and hosting becomes a trivial issue in the development process. The database structure is in the second iteration since having been derived from a collection of flat CSV files.

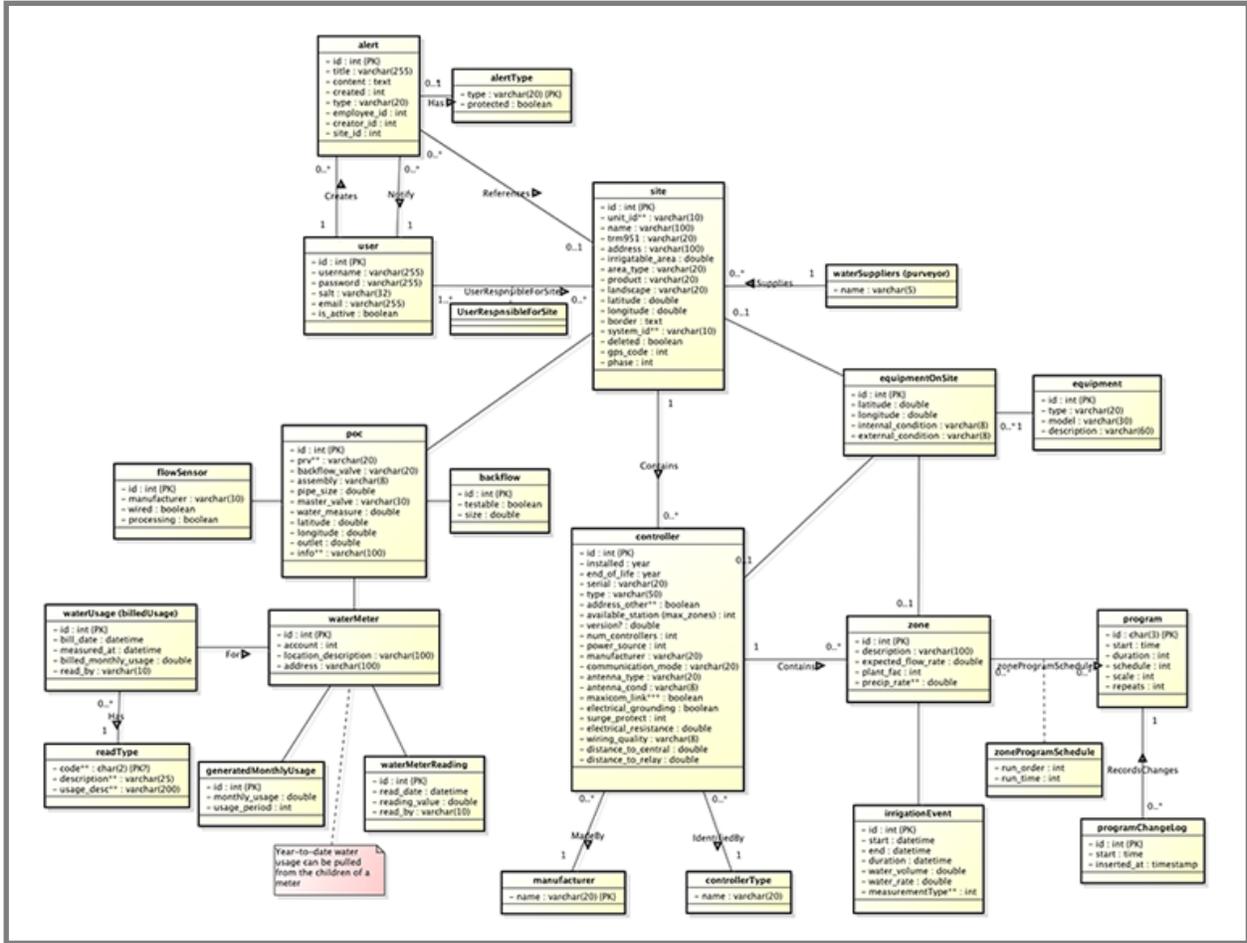


Figure 5.a - Current iteration of database schema

5.2 Backend

The backend portion of the Application functions mainly as the interface between the MySQL database and the frontend, as well as to deliver the application files to the user.

The backend utilizes and MVC (model, view, controller) framework, a design pattern aimed at isolating application logic into core behaviours. The models act as data interfaces, in that all data is queried or inserted through a model. Controllers handle all business logic, performing tasks like data manipulations,

The Application is running on a PHP application framework called CodeIgniter. Initial versions of the application were developed on the more rigid Symfony2 framework, but the steep learning curve was a large barrier to developer collaboration and continuity of the project beyond this thesis. CodeIgniter offers a very lean structure while still providing the flexibility benefits of PHP.

5.2.1 Models

Models are located in the `application/models` directory. Each file in this directory is a PHP class that interacts primarily with one table and any number of joined tables in the database. Each method in a model is logically named according to the request from the database it performs. For example, `sitemodel.php` currently has four methods: `all()`, which returns an array of all sites; `byId()`, which returns a single site referenced by its unique id; `types()`, which returns a list of all site-types; and `suppliers()`, which returns a list of all water suppliers for sites. These files currently use the CodeIgniter ActiveRecord class, which handles database interactions providing numerous features, including: database independence, easier query-building, and auto-escaped queries. However, because the only expected return value from these models is an associative array, the developer may use any method they are comfortable with to request data.

5.2.2 Controllers

Controllers are located in the `application/controllers` directory. Controllers can serve various purposes. The controllers in the Application perform three primary tasks: the first is the aforementioned API, used to relay JSON-formatted data to and from the frontend; second, authenticating users through session analyses and form submission; and third, rendering of HTML views to the browser.

CodeIgniter uses an automatic routing system, wherein URLs are parsed to load specific controllers and methods according to the default pattern: `domain.com/<controller>/<method>/[parameters...]`.

All controllers within the Application that require authentication extend the **MY_Controller** class located in **application/core**. This class ensures that every attempt to load a page is first authenticated against a user in the database. Adding a subdirectory to controllers lengthens the routing pattern by adding a directory segment before the controller. Because the frontend portion of the Application makes numerous data requests, these have all be subdirectored under **api**, creating url patterns such as: **/api/sites/get_all**, which will map the the **get_all()** method of the **Sites.php** file under **controllers/api/**. This allows for a very logically organized API, rather than collecting dozens of methods with elaborate method signatures under one API file.

5.2.3 Views

Views are located in the **application/views** directory. When a controller method is run that is expected to return a page, the application will utilize a view load to return HTML to the browser. These views take an associative array of data as an object, from which they parse and generate the final HTML. This behaviour is utilized heavily in more static, traditional request and response type web applications. In the Application, views are only used to render the authentication forms and the core application files delivery.

While static applications typically split views into modular chunks, the simplicity of the static portion of our application did not warrant this. Instead, we only have two views, **application.php** and **signin.php** which contain full HTML documents.

5.2.4 Mobile detection

The Application required a method of detecting whether the user was accessing the site via mobile or desktop in order to choose the corresponding display logic. In order to accomplish this, the Application utilizes Serban Ghita's PHP Mobile-Detect library. Mobile detection happens server-side, in the **application.php** view, setting an in-page boolean JavaScript variable: **is_mobile**. The frontend portion of the application uses the value of this variable to determine which templates it should load. Loading of the mobile version for development purposes can be done by either manually spoofing the browser UserAgent or by appending **?m** to the URL.

5.3 Frontend

The Application frontend is comprised of a JavaScript application with HTML views and relies primarily on the Backbone.js application framework and several other minor dependencies. Backbone provides a robust application architecture on top of what is considered a very loose language. The library divides the application logic in a similar fashion as CodeIgniter, but uses a model-template-view-router framework. The addition of Marionette.js simplifies this pattern by providing a stronger event framework and module-based organization. All frontend application logic is stored in `assets/js/m`.

5.3.1 Application

The Application is the core of the Backbone/Marionette structure and handles all of the initialization, routing, and global logic. The Application class and global logic is found in `assets/js/m/main.js`:

Line 1 to 70 - This is custom loader that lets the developer store the templates in individual files during the development phase, rather than all in the HTML, which can be hard to manage and slow to edit. Once all of the templates have been loaded, the callback initializes an asynchronous request to `api/payload` which returns all of the core data needed to run the Application effectively, including: a barebones listing of all sites, site types, suppliers, controller types, alert types, and a list of users. Once these payload items have been initialized as Backbone Collections, the App object is started on **line 64**.

Line 73 - `loadTemplate()` is the method Marionette uses to find templates. By default, templates are located on the HTML document and this method would find them with a DOM selection engine. However, to speed development, we load them individually, as mentioned above, and store them in an array and therefore we require that `loadTemplate()` is overwritten to pull them from this array.

Line 79 - instantiates a Marionette Application object.

Line 81 to 123 - `routeController` is a hash array of methods that the application calls depending on the route accessed by the application. Each of these methods loads and starts a particular Module.

Line 125 to 141 - `MyRouter` defines a listing of routes accessible by the application and maps them to the aforementioned `routeController` methods. This hash map defines all URL patterns that the application will recognize as valid and allows for dynamic elements in the URL such as site ids, which are passed into the controller to load and render the proper details.

Line 144 to 149 - a callback to the initialization of the Application object. When the Application is started, this method will run and will initialize the router class and start the history tracker. Here `emulateHTTP` and `emulateJSON` are both set to true, due to the fact we are using a legacy server technology which does not support modern REST headers like PUT and DELETE and can't accept POST data in JSON encoding. Modern servers, such as Node.js, support this functionality by default.

Line 152 to 155 - adds two available display regions for use by the application. These elements, `menu` and `main`, exist in the initial HTML document and are used to render the core frame of the Application.

Line 160 to 215 - `MenuModule` - modules will be explained in more detail below, but it should be noted that this module was included with the `main.js` logic because of its inherent global behaviour.

Line 218 to 243 - `SiteModel` defines the Model class that will represent an instance of site-specific data. Two custom methods have been added to this model: `elaborate()` and `getCenter()`. `Elaborate` will check to see if this is a barebones representation of the site, as used by the main listing, and if it is, it will query the API to receive a full data object which includes all of the equipment, alerts, and joined data associated with

the site for use in the more detailed view renderings. **GetCenter** is used to retrieve the center point of the site by first checking the center attribute, and if that is empty, then by determining the central point of the site based on the geographic coordinates used to draw its boundaries.

Line 261 - `isAdmin()` is a method used to check if the current user is an admin, and is used for various rendering options as to not provide visibility to portions of the Application that are intended for only administrators.

Line 267 to 277 - `IrrigationCollection` defines a collection object used to store special data collections that are intended to be rendered as select-box lists. This is used in such storage instances as sites, site types, and controller types.

5.3.2 Modules

Modules are self-contained MCV entities that contain all models, views, and controllers relating to a specific, potentially modular, portion of the application. In the Application, modules are used to contain each individual view of the application. **SitesListModule**, found at **`assets/js/m/modules/list.js`**, will serve as an example for how modules function within Backbone:

Line 1 - This line is the initialization for the containment of the module.

Line 3 - `this.startWithParent = false` ensures that the application will not attempt to run this module when the application itself starts up. Because the Application shares data between views, this ensures that the module is only active and in memory during use.

Line 7 - `ListItemView` represents the view logic for an individual row in the main listing table. The events hash specifies events that will automatically be attached to the view when shown, in this case a click event that triggers the **`rowClicked()`** method.

template specifies the loaded template file to be used to render the view. `rowClicked()` simply informs the router that it should navigate to a new page, in this case the site detail page, which will contains further information on the site clicked.

Line 21 - `MainView` represents the view logic for the primary view of this module which is the table element and filters views. The `MainView` extends the `Marionette CompositeView`, which is a special view type used to contain a collection of child views, which are specified here to be of the type `ListItemView`. The events hash in this view simply attaches a click event on the button to show and hide the filters view. `initialize()` attaches an event to the `filter:count` message, which updates the number of filters displayed on the button. `appendHtml()` controls the manner in which each `ListItemView` object is added to the `MainView` by dictating that they be added into the `tbody` HTML element of the template for this view. `redrawTable()` re-renders the contents of this views collection.

Line 65 - `Controller` is the class in which all of the logic for a module exists. In this case, `initialize()` defines where on the page the templates will be loaded, which dataset to use, and then initializes the views. `show()` tells the views to render, and ensures that the proper filter settings are activated. `onClose()` is triggered when the controller is closed, which typically occurs when changing pages and simply ensures that all regions and events associated with the module are removed and unbound accordingly. `redraw()` is a method triggered by the `filters:processed` event, which occurs each time the dataset is filtered. When run, it tells the container view to redraw all of the child elements (in this case, table rows) it contains according to the new dataset.

Line 104 - `Mod.addInitializer` - adds a callback method to the initialization sequence of the module. When the application runs this module, this callback is run which essentially just creates and initializes the controller with the region it should be using to display its views.

5.3.3 Templates

Template files are stored in `assets/js/m/templates` and contain small portions of HTML to be inserted into the page by the views. Each view automatically passes its assigned model into the template as a JSON object, at which point each attribute of that object becomes a global variable. The templates use a syntax known as ERB and made popular by Ruby on Rails. Within the template, when an attribute should be printed out, the template uses syntax such as: `<td><%= address %></td>`. This will dynamically generate a table cell with an address within it.

Template file names can that are specific to either desktop or mobile should be named with either `*.d.html` or `*.m.html`, respectively. By appending the global variable `tpl_type` to the template definition in a view, the view will load the appropriate template automatically.

6 Feedback

In developing this application for the Parks Department, we utilized a pragmatic process. The department was unsure of the most effective way to utilize, visualize, and translate their existing data, therefore our process was largely exploratory. We began by prototyping early, straight translations of the data so the department could begin to visualize the data beyond spreadsheets. Out of this process, we discovered new avenues of use for the data and new needs that could be solved. For example, by creating proof-of-concept prototypes of visualized parks on an interactive map, we discovered that the application should be divided into two core systems with shared functionality: one for desktop, in-office use, and the other for mobile, in-field use. The Parks Department worked closely to ensure the application continued down a path that met their needs with a consistent vision.

7 Conclusions

In its current state, the department has expressed that they are pleased with the progress of the Application and the clarity it brings to their dataset. The department is excited about the

possibilities it holds for visualizing their usage trends and maintenance patterns, as well as optimistic about the projected growth of the Application. They are eager to test the current stable version in the field in the upcoming season. The ease with which reports can be generated, as well as the immense increase in accessibility has had a great response from the members of the department. It was an exciting opportunity to work on a project that makes a sizable difference in the way they approach their data with each small iteration, and it was exciting to discover new needs that could be met for the department with each of these iterations. The possibilities for this application to help facilitate a sustainable future are incredible, not only Kelowna, but also for markets where water resources are even scarcer.

8 Future Work

The Application was designed using module patterns in order to simplify future additions by ensuring that the code for each module was completely encapsulated, while the core application functionality remained available across the entire system. By utilizing a shared code base on both the desktop and mobile applications, we ensured that future progress on both ends of the application will maintain a consistent direction and avoid any complications and technical debt from constantly diverging paths. Future plans for the application that have been discussed are more detailed reporting analytics, public-accessible reporting, maintenance logging, enhancing the map views with more data and detailed equipment, and scheduling-system integrations. The modular approach that has been taken in development thus far makes these updates and modifications seamless.

9 Acknowledgements

In conclusion of this project, I would like to thank Dr. Ramon Lawrence for his supervision, direction, and for acting as the project manager and liaison between UBCO and the Parks Department. I would like to thank Ryan Trenholm for his previous work with the Parks Department, the prototype phase of this project, and for assisting with portions of the build for this application. I would also like to thank Cody Clerke for his work parsing existing data formats and developing new database structures for the Application.

10 References

Backbone.js - <http://backbonejs.org>

Bootstrap - <http://twitter.github.io/bootstrap>

CodeIgniter - <http://ellislab.com/codeigniter>

Marionette.js - <http://marionettejs.com>

MySQL - <http://mysql.com>

PHP - <http://php.net>