iSprinkle: Design and implementation of an internet-enabled sprinkler timer

Carlos Adrian Gomez, Adam Sedziwy, Michael Soltys

July 20, 2017

Abstract

This paper presents the details of the design and implementation of an internet-enabled sprinkler scheduling system, undertaken as a senior level capstone project at the California State University Channel Islands. The end result is a functioning prototype, but, more importantly, the project has tremendous pedagogical value as it combines advanced programming, introduction to embedded systems, application, as well as aspects of sustainability and California State law.

1 Introduction

The aim of this paper is to report on the design and implementation of a sprinkler timer for home usage, as well as expound on the pedagogical value of the exercise. The aim is to provide enough details and references so that the reader can replicate the final product, and thus obtain a working sprinkler timer, as well as report on the insights for instructors who may wish to repeat this project and draw learning value for the student.

iSprinkle is a Raspberry Pi-powered irrigation controller which will allow a user to set an initial irrigation schedule for a sprinkler system using a web interface, after which it will use the local weather forecast to adjust the base watering schedule as-needed. By doing so, iSprinkle will be able to irrigate more efficiently compared to a fixed schedule; by programmatically modifying the user's watering schedule, iSprinkle will increase/decrease the amount of watering that the schedule dictates depending on data that it receives from a weather API. iSprinkle hopes to make it easier for homeowners to conserve water by automating adjustments to their irrigation schedule.

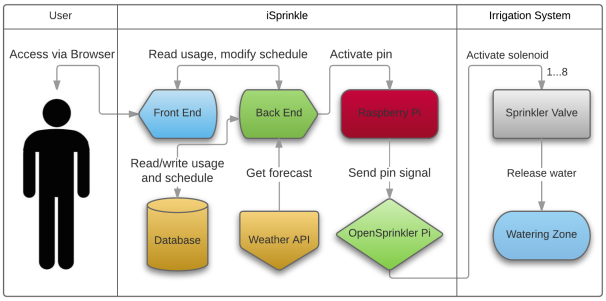

Figure 1: Diagram of the project.

2 Hardware components

The hardware is inexpensive, and all the components can be purchased for about $150. This is the price of a standard "smart" WiFi sprinkler controller; however, the standard products have a very limited set of capabilities compared to our solution, and they are not open source.

Table 1 contains an itemized list of the hardware components, including the URLs of venues that sell the items online, and Figure 2 contains pictures of the items. We do not include the sprinkler system itself, that is the valves, the pipes, the sprinklers, etc., as iSprinkle is only intended to be a replacement for the sprinkler scheduling system.

	Component	**URL**	**Price**
(a)	Raspberry Pi 2 (RPi)	`amzn.to/1Yv2nrL`	$39.99
(b)	OpenSprinkler Pi (OSPi)	`rayshobby.net/cart/ospi`	$77.99
(c)	Samsung 16Gb MicroSD card (SD)	`amzn.to/20UnEw6`	$7
(d)	Edimax USB WiFi adapter (Edimax)	`amzn.to/1WDvo5d`	$8.50
(e)	24V Transformer (T)	`thd.co/1NzwXKf`	$12.97
		Total:	$146.45

Table 1: Itemized hardware components

For the module to be practical we may also need to add an Uniterrupted Power Supply (UPS)[1] to deal with power outages.

3 Software components

3.1 Operating System

Once all the hardware components are out of the box, we must start by downloading and installing an Operating System (OS) for for the RPi. The OS resides

[1] `https://www.pi-supply.com/product/pi-ups-uninterrupted-power-supply-raspberry-pi`

(a) Raspberry Pi 2 (b) Open Sprinkler Pi

(c) MicroSD card (d) Edimax WiFi (e) 24V Transformer

Figure 2: Pictures of hardware components

on the SD card, and so the SD card must be formatted and the image of the OS copied to it. On a Mac this can be done as follows[2]:

1. Download the SD card formatting tool, "SDFormatter," from:

   `https://www.sdcard.org/downloads/formatter_4/eula_mac/`

   and once it is installed, insert the SD card[3], and run the formatting tool by selecting the "Overwrite Format." (Here it is important to make sure that the right card is selected.)

2. Download the New Out Of Box Software (NOOBS) from:

   `downloads.raspberrypi.org/noobs`

3. Unzip the files, and copy them to the SD card that was just formatted. Eject the SD card from your computer, and insert it into the RPi.

4. Connect the RPi to a monitor (with an HDMI cable), a keyboard, and a mouse. Insert the WiFi adapter (Edimax) into one of the USB slots.

5. Connect to RPi to the power supply (T), which will boot it, and select the OS from the given list. We recommend installing Raspbian, a free OS optimized for the Raspberry Pi hardware and based on Debian Linux, and will describe the setup required for it in the proceeding sections.

6. Once the system boots in the appropriate OS, connect your RPi to the local WiFi, and now you will be able to disconnect all the peripherals and control your RPi by ssh'ing into it.

[2] `https://www.raspberrypi.org/files/legacy/qsg.pdf`

[3] Note that the SD card comes with an extension that makes it the appropriate size for a Mac card slot.

3.2 Environment

3.2.1 System Packages

In order to keep Raspbian updated with the latest software patches, we first update the system's package list by entering the following command in LXTerminal or from the command line:

```
sudo apt-get update
```

and next, upgrade the installed packages, firmware, and kernel to their latest versions with the command:

```
sudo apt-get dist-upgrade
```

3.2.2 Raspbian Configuration

Before setting up iSprinkle, one of the most critical parts of configuration is the RPi's timezone settings; they must be configured to local time so that the watering schedule start times are accurate. As of this writing, timezone settings are located under `Internalization Options`.

The Raspberry Pi configuration tool can be accessed with the command:

```
sudo raspi-config
```

Other configuration options include the ability to remotely access the Raspbian's desktop environment via Virtual Network Computing (VNC), disabling the desktop environment altogether, and modifying the amount of memory allotted to the Graphical Processing Unit (GPU)[4].

3.3 Installing iSprinkle

1. Before installing iSprinkle, it is good practice to setup a virtual Python environment so that Python modules required for iSprinkle are isolated to the project itself and not available globally throughout the system [5]. First, install the `virtualenv` module via `pip`, Python's package manager.

   ```
   sudo pip3 install virtualenv
   ```

2. In the pi user's home directory `~`, create a new virtual environment in a directory called `venv` with the command:

   ```
   virtualenv venv
   ```

3. The `venv` directory contains a copy of the Python interpreter, pip package manager, as well as other tools, which can be used as iSprinkle's isolated environment. Continue by activating the environment with the command:

   ```
   source venv/bin/activate
   ```

[4] `raspberrypi.org/documentation/configuration/raspi-config.md`
[5] `packaging.python.org/installing/#creating-and-using-virtual-environments`

After activation, the command prompt should show the following:

```
(venv) pi@raspberrypi:~ $
```

4. Clone the iSprinkle repository with the command:

   ```
   git clone https://github.com/cagmz/iSprinkle.git
   ```

5. Change directory into the newly downloaded repository:

   ```
   cd iSprinkle/
   ```

6. Install iSprinkle's Python dependencies using pip using the command:

   ```
   pip install -r requirements.txt
   ```

7. Run iSprinkle using the command:

   ```
   nohup python3 iSprinkle.py &
   ```

   The command `nohup` and BASH's control operator `&` make sure that the program continues running even after we log out from the RPi (recall that we assume that we control the RPi by ssh'ing into it).

8. Access iSprinkle via any web browser on the local network via:

   ```
   <Raspberry Pi IP>:8080
   ```

   where `<Raspberry Pi IP>` is the IP address of the RPi on the network.

4 Running iSprinkle

Here are the functional requirements for running iSprinkle.

1. Manual run
2. Set irrigation schedule
3. Retrieve weather data
4. Optimize schedule
5. View historical irrigation logs

4.1 Manual run

Allow the user to manually selected water zones for a specified amount of time. This can be done on any device (on the same WiFi as the RPi), by accessing the following address on any web browser:

```
http://sprinklerrp.local:8081
```

where "`sprinklerrp.local`" is the name of the RPi on the local network (this can be also replace with its IP, or any other name that was given to the RPi).

4.2 Set irrigation schedule

Allow the user to set a watering schedule for up to 8 stations. In a typical home sprinkler system, each station pertains to one sprinkler valve. This watering schedule, as well as future adjustments, will be stored in the database.

4.3 Retrieve weather data

Poll the weather API in order to gather the latest weather forecast and adjust the watering schedule as needed.

4.4 Optimize schedule

Optimize the user's watering times based on a sliding scale according to the forecasted temperature from the weather forecast.

4.5 View historical irrigation logs

Display historical data from the database to the user, allowing them to display watering duration totals.

5 Water usage

In California, the current drought has become so severe that the state and local governments have begun regulating water consumption, imposing sanctions and even passing legislation in order to curb water usage. On average, the statewide ratio for water usage is about 50% environmental, 10% urban, and 40% agricultural [MFL14].

Irrigation is one of the widest uses of water nationwide, accounting for more than 60% of water withdrawals in our state [MKH+10]. Fortunately, the amount of water used in both urban and agricultural irrigation has been reduced through a variety of measures, including a strong trend in the use of precision irrigation techniques (e.g., drip irrigation) [Han07].

However, research suggests our current unprecedented drought is only expected to get worse; for this reason, it is imperative to be proactive and reduce our water consumption further [CAS15]

5.1 Residential over irrigation

In a random survey of single-family water customers sponsored by the California Department of Water Resources, results showed that 87% of homes appeared to be irrigating with only 54% doing so in excess [DMM+]. However, the surveyors also mentioned that most water customers were irrigating at or below average levels.

The survey found that 62% of excess usage occurred on 18% of all irrigating lots, leading the surveyors to conclude that "the majority of savings from

outdoor use will be found from around 15% of the customers." For this reason, they suggested that a solution to reduce outdoor water usage should be focused on those households which over irrigate, so that households who are irrigating at appropriate levels are not affected.

Of the survey respondents, only 4% were said to be using weather-based irrigation controllers (or WBIC), despite some municipalities offering rebates towards commercially available products. The low levels of adoption surrounding WBIC's is especially concerning given the potential savings; it is estimated that a WaterSense-labeled irrigation controller, or one that meets the EPA's requirements for watering without doing so in excess, can save the average home almost 9,000 gallons of water per year [Age]. The EPA estimates that if every US home replaced their sprinkler timer with a WaterSense labeled controller, the potential savings "could save $435 million in water costs and 120 billion gallons of water across the country" [Age].

Over the past 15 years, there have been numerous studies intended to evaluate the reduction of water usage of WBIC's, compared to traditional timers, when retrofitted at an over irrigating household. Most studies suggest that "savings of 40-50 gallons per household per day, or roughly 10% of total use can be expected from a residential WBIC retrofit program assuming such programs target high water users"[Res].

The aforementioned studies differed in the criterion for targeting over irrigators, citing difficulty in devising a methodology that was effective. However, as WBIC's become more commonplace and more households in general begin to adopt the technology, we can be sure that at least a percentage of these will be over irrigating households and will reap the benefits of "smart" irrigation.

6 Pedagogical value

Although this project involves both hardware and software, the learning potential leans greatly on the latter. We will expand on the learning value of the project from the initial setup to the functioning prototype.

6.1 Hardware

Assuming the user has an existing sprinkler system complete with sprinkler valves, iSprinkle is intended as a drop-in replacement for a home irrigation timer. The OSPi expansion board has female GPIO pins; all that is needed is to plug in the RPi and securely close the OSPi enclosure so that both boards are protected. After this, the sprinkler valve solenoid leads must be connected to the OSPi, and the 24 volt power supply connected, providing power to both the OSPi and RPi.

6.2 Operating System

The Linux environment offered by Raspbian offers a wealth of learning opportunities. While Raspbian is easy to use due to the graphical user interface (GUI), users are free to explore the underlying system using the Bash shell.

As mentioned previously, assuming the user interacts with the RPi via a remote shell using the SSH, they will become intimately familiar with Linux commands, if they are not already. New users are encouraged to review the Console Basics section in the Debian Reference [Aok16]. In addition, Linux provides convenient documentation for Unix utilities via the `man`.

6.3 Python

The server-side software of iSprinkle is written solely in Python, a versatile scripting language which has been adopted by many learning institutions for teaching programming to beginners. Python is an excellent choice for new and experienced programmers due to its ecosystem. Because Python is open-source, there is a wide variety of freely-available learning materials as well as online communities providing support.

Although Python's standard library is already quite extensive, offering features such as built-in support for networking and interfacing with the underlying OS [Fou16b], there are also thousands of third-party libraries available via PyPI, the Python Package Index [Fou16a]. Due to Python's open nature, users will find it easy to modify and extend iSprinkle's existing codebase.

6.4 Web Development

The user-interface for iSprinkle combines HTML, CSS, as well as JavaScript. Bootstrap, a popular framework for developing web sites, is used to easily create stylish pages, many of which can contain components such as buttons, forms, and icons, and are responsive (i.e. adaptable to the viewer's screen size and platform) [OT]. Bootstrap provides a gentle introduction to web development in that it makes it easy to create pages which follow a convention rather than spending time on configuration; however, because the documentation is so extensive, users will easily be able to learn about HTML and CSS for both structuring and styling websites, as well as JavaScript, for adding interactivity. However, because Bootstrap only provides structure, style, and a limited set of dynamic features for the user interface, another component is required.

Angular, a client-side JavaScript framework maintained by Google, is used to "extend the vocabulary" of HTML by adding templating, bi-directional data binding, and scope to the traditional static HTML page [Goo]. Users will learn to use "scopes", in the traditional computer science sense, as they declare variables and use logic to add even more dynamic features to web pages. In addition, Angular makes it easy to start learning about asynchronous programming, such as when making requests to a server, be it local (such as iSprinkle's backend) or remote (such as an external API), without affecting the user's experience by

waiting for the reply. Finally, JavaScript itself has seen enormous changes within the last decade; countless web frameworks have been built upon it, making it an enormously versatile language to learn.

6.5 Software Engineering

The design of a system such as iSprinkle requires a holistic approach that is very different from most class assignments. The former usually span a few files that are to be turned in within a week or two, making it difficult to implement a system with many 'moving parts.' However, iSprinkle's functionality is divided between the front-end and backend, both of which need to communicate so that the user's requests are fulfilled. Designing such a system requires taking into consideration many aspects; from major decisions such as deciding on a backend language to use, to minutiae such as the date and time formats to use across the backend and front end to maintain consistency.

7 Implementation

Throughout the design and implementation of iSprinkle, a major consideration was to emphasize modularity, allowing for separation of concerns between components while maintaining a standard way to communicate between them. For this reason, iSprinkle's functionality is split between the front end, allowing the user to interact with the software in a user-friendly way, and the back end, which does most of the work.

In this section, we will go through the typical work flow for a user, starting from the front end, and trace the request to the back end, while detailing implementation details and design decisions.

7.1 Home

The dashboard is where users are able to view historical watering usage. Users select a date range, click on a button, and graph displaying past watering usage is shown. As of this writing, iSprinkle displays the cumulative watering times for all stations given a date range.

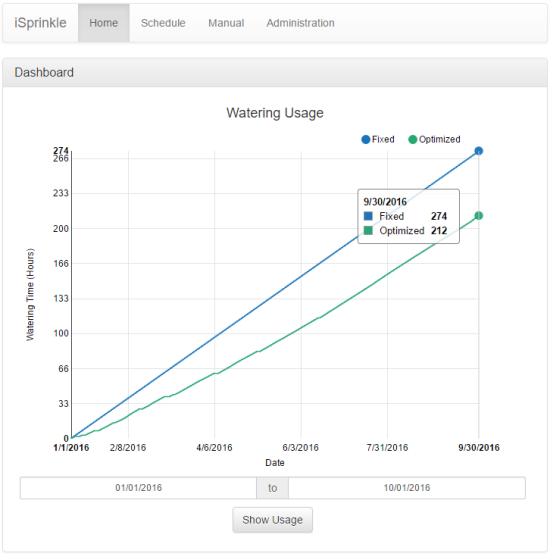

Figure 3: Both original and optimized durations are plotted for comparison.

Starting from the user input, the bootstrap-datepicker library is used to provide a user-friendly way to input dates [dW]. After clicking the button, Angular begins validating the user input by preparing attempting to craft proper request before sending it to the backend. For instance, the start and end dates are converted to an ISO 8601 date time string in UTC time, a format which is used consistently throughout the front and back end.

After a proper request containing start and end dates, as well as the stations to request usage from is crafted, the request is sent to the back end using Angular's `$http` service. The `$http` returns a promise, which allows the back end to reply at a later time.

Once the back end receives the request, it queries the database for entries in the historical usage table that are between the start and end dates and that pertain to the selected stations. Once the rows are found, they are sent back to the front end in JSON format.

When the reply from the back end is received, a success callback function in Angular passes the payload to another function for parsing before displaying in a graph.

Parsing the payload is done by creating two cumulative watering arrays; one for the fixed schedule and another for the optimized watering times. The data returned by the back end is iterated through and the arrays are populated by inserting tuples containing the date as well as the running total of their respective watering times. These tuples represent (X, Y) coordinates, and both of these arrays are then used by `d3.js` for plotting [Bos].

7.2 Schedule

Users are able to create, read, update, and delete watering times for stations on the schedule page. The entire weekly schedule can be seen at a glance, with one station per row and every weekday as a column. Each station can have multiple start times in a day.

When iSprinkle executes, a job scheduler provided by the Advanced Python Scheduler module is instantiated [Gr]. As the watering schedule is loaded into memory from disk, new jobs are added to the job scheduler for each station. Each job can be thought of as a tuple which contains the watering function, start time, and watering function arguments, where the latter consists of the station number and watering duration. The job scheduler runs in a separate thread from the main application and executes the watering function when a start time for a job is reached.

When the start time for a job is reached (i.e. a station is due to start watering), iSprinkle retrieves weather data from the past month. The average temperature, along with the current temperature and desired watering duration, are used to produce an optimized watering duration which is then used in place of the user's original value.

When serving the schedule page, Angular makes a request to the backend for the current schedule using the aforementioned `$http` service. Once the back end returns the schedule as a JSON object, Angular saves it in the schedule controller's scope. HTML is dynamically generated by Angular, containing textfields for start times and durations present in the schedule. These textfields have bidirectional data binding; once the schedule has been displayed, the user is able to modify start times and durations in the schedule table, and the JSON object in the scope is modified at the same time. Angular then displays the schedule in a table.

When the user saves an updated schedule, the existing schedule object in the scope already contains the modified values due to the two-way data binding. Angular removes start times with durations equal to 0 are removed from the schedule, and Angular's `$http` service makes a POST request with the updated schedule to the back end after which it is saved to disk.

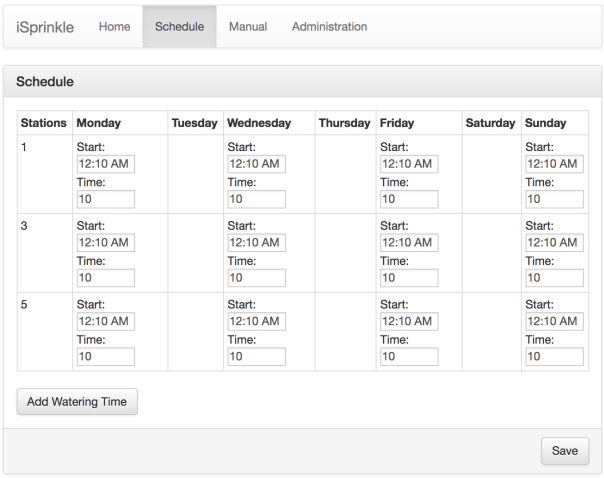

Figure 4: Text fields are bound to a JSON object in Angular.

Users are able to add watering times for multiple stations and days at the same time by using dropdown menus for stations and days, making it easy to quickly set up an initial schedule. Once the user has filled the form, Angular creates start time entries and inserts them into the schedule JSON object that is already present in the schedule controller's scope. This updated schedule is then sent to the back end in a similar fashion as an updated schedule would be.

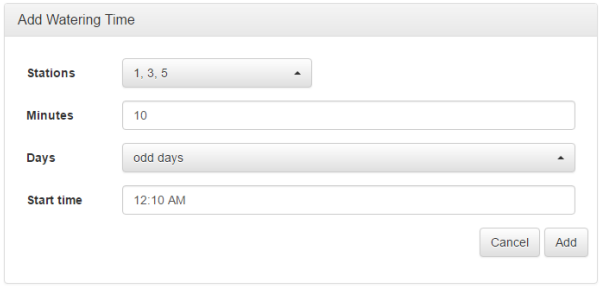

Figure 5: A flexible way to add watering times.

7.3 Manual

Many times, it is necessary to manually activate sprinkler valves. iSprinkle provides on-demand station activation on via the Manual page. The user is presented with a form showing their stations along with a textfield input to specify the manual watering duration in minutes for each.

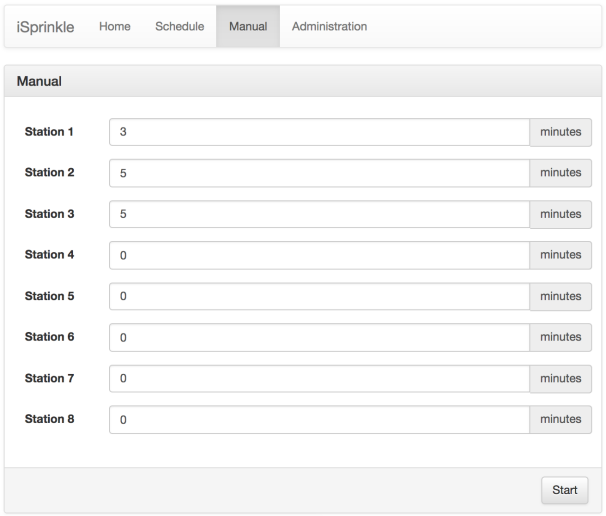

Figure 6: Users are able to manually irrigate on demand.

When the user accesses the Manual page, Angular requests a list of active stations from the backend and then dynamically generates a list of stations along with a textfield belonging to each. Users are then able to input the number of minutes that they would like to activate a station for. After pressing the Start button, Angular validates the user's input making sure that they are valid watering duration (i.e. a positive value), crafts a JavaScript object which contains the request, and then uses the `$http` service to POST a request with that object to to the backend.

Once the backend receives the watering request object, the user's normal operating schedule is paused. New jobs are created for each of the stations that the user wants to water and are added to the schedule, and a final job responsible for resuming the previously paused jobs in the user's schedule is also added. After the stations have finished watering in a serial fashion, the jobs associated with the original schedule are resumed.

7.4 Administration

Users are able to view system information and update settings on the Administration page. Currently, users are able to view the the local IP address and uptime, as well as update their "active stations". Active stations allow users to enable/disable stations system wide, displaying only these selected stations in the user interface and scheduling their watering jobs. One use case is for active stations is if a user has, for example, 3 sprinkler valves installed and thus only needs 3 stations; by setting those 3 stations as "active", they can remove clutter in the user interface, especially in the Schedule page.

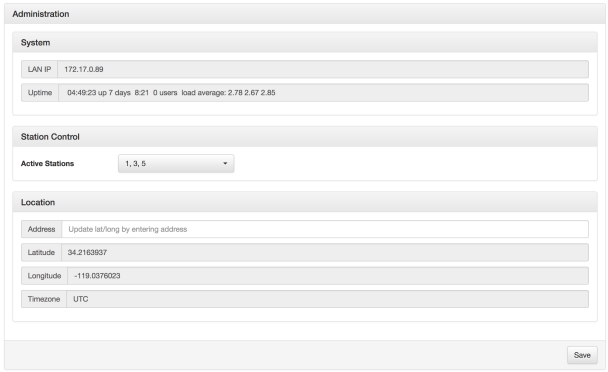

Figure 7: System wide status and configuration.

For system information, the front end simply queries the iSprinkle backend with a GET request for the requested information using the appropriate API endpoint. For example, the uptime field is populated by Angular by using the `$http` service to perform a GET request to the `/api/rpi/ip`.

However, updating settings is performed by crafting a JavaScript object when the user presses the Save button. The JavaScript object contains key, value pairs whose keys map directly to the user's existing settings file. Angular's `$http` service uses a POST request to deliver this object to the backend, after which the backend overwrites previous values associated with the keys.

8 Conclusion

This article presented a senior capstone project aimed at preparing a low cost, open source-based sprinkler timer capable of performance adjustment on the basis of weather forecast data being gathered in the background. The project covered assembling both hardware and software components. The latter ones were based on open source solutions and technologies: the Raspbian operating system, Python-based components, Angular, Bootstrap and others. The project required the student to develop skills in several areas: assembling together all hardware items; installing and setting up the OS and the software environment; programming using advanced web technologies. As mentioned in Section III, the basic assumption was reducing the cost of the solution. Meeting this requirement and relying on the open software, make iSprinkle to be easily replicable. This fact together with the environmental context, i.e. the common water shortages in California, open good prospects for commercial application of iSprinkle.

References

[Age] United States Environmental Protection Agency. Watersense labeled irrigation controllers — watersense — us epa. `https://www3.`

epa.gov/watersense/products/controltech.html. (Accessed on 12/17/2016).

[Aok16] Osamu Aoki. Debian reference. https://www.debian.org/doc/manuals/debian-reference/, September 2016. (Accessed on 10/15/2016).

[Bos] Mike Bostock. D3.js - data-driven documents. https://d3js.org/. (Accessed on 12/16/2016).

[CAS15] Benjamin I. Cook, Toby R. Ault, and Jason E. Smerdon. Unprecedented 21st century drought risk in the american southwest and central plains. *Climatology*, February 2015.

[DMM+] William B. DeOreo, Peter W. Mayer, Leslie Martien, Matthew Hayden, Andrew Funk, Michael KramerDuffield, and Renee Davis. California single family water use efficiency study — aquacraft. http://www.aquacraft.com/2015/07/28/california-single-family-water-use-efficiency-study/. (Accessed on 12/17/2016).

[dW] Joris de Wit. Timepicker for twitter bootstrap. https://jdewit.github.io/bootstrap-timepicker/. (Accessed on 12/16/2016).

[Fou16a] Python Software Foundation. Pypi - the python package index : Python package index. https://pypi.python.org/pypi, 2016. (Accessed on 10/15/2016).

[Fou16b] Python Software Foundation. The python standard library ? python 3.5.2 documentation. https://docs.python.org/3/library/, September 2016. (Accessed on 10/15/2016).

[Goo] Google. Angularjs a superheroic javascript mvw framework. https://angularjs.org/. (Accessed on 11/09/2016).

[Gr] Alex Grnholm. Advanced python scheduler ? apscheduler 3.3.0.post4 documentation. https://apscheduler.readthedocs.io/en/latest/. (Accessed on 12/16/2016).

[Han07] Blaine Hanson. Irrigation of agricultural crops in california. Technical report, University of California Davis, 2007.

[MFL14] Jeffrey Mount, Emma Freeman, and Jay Lund. Water use in California. Technical report, Public Policy Institute of California, July 2014.

[MKH+10] Molly A. Maupin, Joan F. Kenny, Susan S. Hutson, John K. Lovelace, Nancy L. Barber, and Kristin S. Linsey. Estimated use of water in the united states in 2010. Circular 1405, United States Geological Survey, 2010.

[OT] Mark Otto and Jacob Thornton. Bootstrap the world's most popular mobile-first and responsive front-end framework. http://getbootstrap.com/. (Accessed on 11/09/2016).

[Res] Western Policy Research. Weather based irrigation controllers. http://www.cuwcc.org/Research-Portal/Weather-Based-Irrigation-Controllers. (Accessed on 01/09/2017).