

# ISU Bikeshare Project

---

APRIL 24, 2016

ADVISOR: MARK KARGOL AND MANI MINA

BRIAN ANDERSON  
MOHAMMAD DARWISH  
KAITLYNN KRUSE  
JEFF NEEL  
KYLE PERKINS  
MOLLY SCHAEFFER  
JORDEN STUDER

## Table of Contents

|   |    |
|---|----|
| Introduction .....                                    | 2  |
| Problem Statement.....                                | 2  |
| Purpose .....   | 2  |
| System Design.....                                    | 2  |
| Detailed Design .....                                 | 2  |
| Key Functional Requirements .....                     | 2  |
| Key Non Functional Requirements .....                 | 3  |
| Functional Decomposition .....                        | 3  |
| Software.....   | 3  |
| Hardware .....  | 4  |
| Assessment of Proposed Solution.....                  | 4  |
| Process Details .....                                 | 5  |
| Interface.....  | 5  |
| Interface Description .....                           | 5  |
| Materials .....                                       | 7  |
| Project Expenses .....                                | 7  |
| Risks .....   | 7  |
| Feasibility .....                                     | 8  |
| Testing.....  | 8  |
| Hardware .....  | 8  |
| Web Application.....                                  | 9  |
| Android Application .....                             | 13 |
| Schedule.....   | 15 |
| Project Schedule .....                                | 15 |
| Conclusion.....                                       | 16 |
| Appendix I: Operation Manual.....                     | 16 |
| Appendix II: Alternative Versions of the Design ..... | 25 |
| Hardware Alternatives .....                           | 25 |
| Software Alternatives .....                           | 26 |
| Appendix III: Other Considerations.....               | 26 |

## Introduction

### Problem Statement

The Iowa State Bike Share project is an initiative to provide a bike-sharing service to Iowa State University and the Ames community. The project requires considerable work from a collection of multi-disciplinary teams to create a working solution. As Electrical, Computer, and Software Engineers, we must develop and expand a working program that can fit in with the work of other teams working on this project. This Bike Share project would be providing a solution at or below the cost of existing solutions that have been implemented in large cities. Our project would be the first of its kind. The electrical team has worked to redesign and perfect the docking system electronics that help control and manage the bikes. The software team has worked on the backend administration web application and frontend user-facing mobile and web applications. This project has currently been worked on for a few years and is proposed to be implemented on Iowa State's campus beginning fall of 2017 if funding is reached.

### Purpose

The purpose of this document is to outline the scope, timeline and specifications for the electrical and software teams involved in the Bike Share project. What follows is a comprehensive plan of how these teams' objectives were achieved over the past year.

## System Design

Our hardware team focused on reworking most of the previous bike-stand from last year's senior design team in order to create a more efficient and reliable system. In doing this, the power supply system was replaced by a typical computer power supply unit in order to avoid stepping the power down ourselves for different components, thus making a more reliable system and utilizing fewer custom manufactured components. A lot of the wiring in the previous system was simply a mess, not well protected, and nearly untraceable; to get rid of excess wiring we replaced the previous LED system to lessen the amount of wires, and we also ran the wires in a more efficient manner.

Our software team has focused on building the web applications. A simple application was initially put in place by last year's senior design team, but we decided to overhaul for a few different reasons. We have built new frameworks for mobile, administrator, and user systems. For the web applications we decided to implement a Django python web application. We decided to utilize Django python because of its ease of use and its security. The other portion of the software that was developed was the Android mobile application. We worked to develop functional applications that were easy to understand and further develop in the future.

## Detailed Design

### Key Functional Requirements

The bike will unlock from the dock when a user swipes their card.

The docking stations must be simple and easily produced

The system will keep track of user interactions for each dock.

Users can report that a bike is damaged.

Users can view all bike stations on a map on the mobile app.

### Key Non Functional Requirements

Only registered users should be able to check out a bike

All bike transactions will be retained for a minimum of 1 year

The check-in and check-out process should be less than 5 seconds.

The map on the mobile app must load within 5 seconds.

### Functional Decomposition

#### Software

##### Android Application

1. Map of Docking Stations
  - a. Available bikes, empty docks
2. Ride History
  - a. See all transactions of your bike rentals
3. View Statistics
  - a. See ride statistics including total rides and rides per week
4. Report Bike Maintenance
5. Submit a Contact Request

##### User Web Application

1. Map of Docking Stations
  - a. Available bikes, empty docks
2. Transactions
  - a. See all transactions of your bike rentals
3. Report Bike/Dock Maintenance
4. Submit a Contact Request

##### Admin Web Application

1. Map of docks
2. Security
  - a. Login page for Interface
  - b. REST Implementation
3. Adding more functionality to the existing REST API
4. Tracking bike maintenance
5. See all users/bikes (done)/docks (done)
  - a. Users including riders & admins

## Raspberry Pi

1. Check-out confirmation
  - a. Check if valid User ID
  - b. Ensure registered bike is in lock and secured
  - c. Indicate to user the status of the stand
  - d. Send and receive use-data to and from the database
2. Check-in confirmation
  - a. Check if valid Bike ID
  - b. Ensure bike is placed in the lock and is secure
  - c. Indicate to user the status of the stand
  - d. Send and receive use-data to and from the database

## Hardware

1. Wiring
  - a. Choose hardware that enables a more simple wiring configuration
  - b. Clearly document and route wiring to enable easier assembly/maintenance
  - c. Reduce splicings, soldered wires, and other labor intensive harness structures
2. Lights
  - a. Introduce lights that use a simpler wiring configuration
  - b. Use brighter LED's and/or direct the lights to make them brighter
3. Power supply
  - a. Remove the 'homebrew' voltage regulator
  - b. Use only off-the-shelf components to reduce construction costs
  - c. Provide a reliable and easy way to set up power system
4. RFID Reader
  - a. Make sure RFID reader has necessary range and reliability
  - b. Reduce chances of unwanted user interactions
5. Display
  - a. Provide clear feedback for users
  - b. Allows for future development
6. GPS
  - a. Determine if GPS is desired way to keep track of bikes
  - b. Verify plausibility of mounting onto bike
7. Mechanical
  - a. Documenting possible mechanical changes to the docking station
  - b. Labelling component mounting locations and specifications

## Assessment of Proposed Solution

Our proposed solution and requirements were approved by Professor Kargol and were feasible for the project time restraints and group size working on this project. Professor Kargol approved our proposed plans to improve the current stand's hardware to make it more efficient and less costly, as well as, in terms of software, adding everything necessary in order for the system to communicate and behave effectively.

Originally, Professor Kargol had proposed that there be a bike reservation feature. We decided after looking into this proposed addition further that this would not be realistic. Our main concern arose for the situation that one could walk up to a bike rack full of bikes and they would all be reserved by someone else. We want to have a system that is available to use for the users; the possibility of all bikes being reserved at a given time was not appealing. We also knew that other bike share systems that exist do not allow reservations, so we didn't think this would be a necessary addition.

Anti-theft will be handled using an RFID reader and lever switch on the bike-stand lock. These two pieces will work in tandem to ensure that a bike is being submitted to the bike-stand and also that the bike actually gets put into the stand and locked.

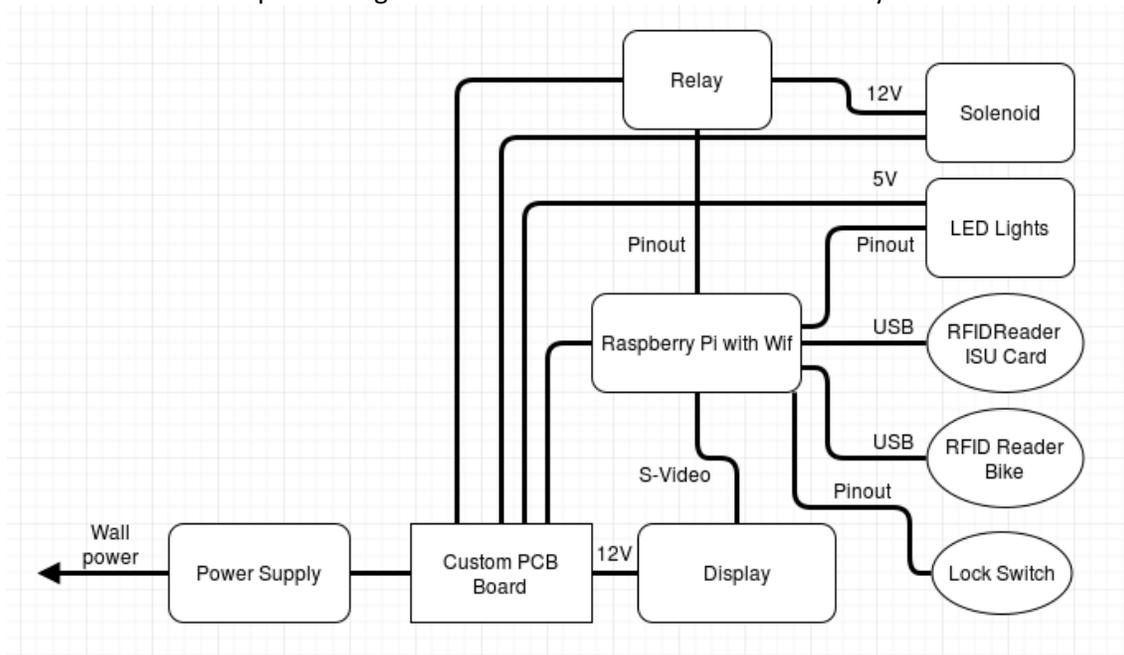
### Process Details

This bike-share project is split into two groups: hardware and software. These two groups have worked in parallel to each other in order to maximize efficiency on the project. The hardware team worked on a new bike-stand in order to improve upon the older system - the details of this are described above in the detailed design section as well as the operation manual portion of the appendix. At the same time, the software group split up even further in order to work on the Android application and the Administrative and User web applications. In this way, we were able to produce thought-out functional applications for both users and administrators.

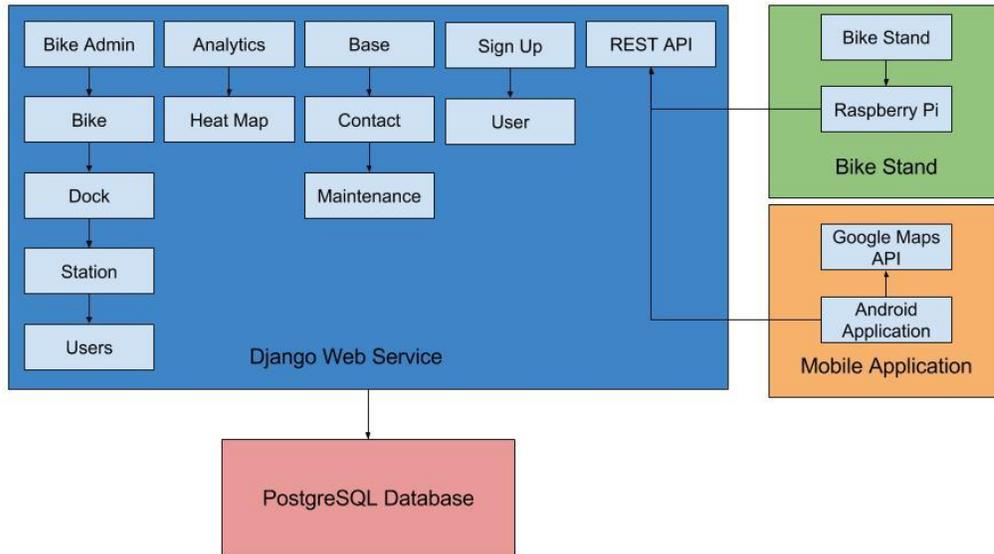
## Interface

### Interface Description

A simplified diagram of the hardware for this bike-stand system



A simplified diagram of the software for this entire bike-share system



## Materials

### Project Expenses

| Resource       | Quantity | Estimated Cost/Unit | Purchasing location |
|----------------|----------|---------------------|---------------------|
| Raspberry Pi 3 | 1        | \$35                | MCMElectronics      |
| Memory Card    | 1        | \$5                 | Amazon              |
| Relay          | 1        | \$10                | Amazon              |
| Card Reader    | 1        | \$115               | Newegg              |
| Tag Reader     | 1        | \$60                | SparkFun            |
| Power Supply   | 1        | \$30                | Amazon              |
| LED Strip      | 1        | \$12                | Adafruit            |
| Display        | 1        | \$15                | Amazon              |
| Solenoid       | 1        | \$15                | SparkFun            |
| PCB            | 1        | \$10                | OSH Park/Digikey    |
| Cables/Wire    | N/A      | \$10                | Amazon              |

## Risks

We were unable to access the database information of the Iowa State students. We mocked a version of the database in order completely eliminate any sensitive data is in our environment. We worked with ITS to mock the data in the exact format of this data that is used at Iowa State. We were required to implement comprehensive access control on the server. We designed this in such a way that users had separated actions so they are not able to perform any actions that should not be available to them. We were unable to get hardware ordered and received in a reasonable time frame which presented a risk of having to add parts to the stand that were not thoroughly tested. This could lead to failing parts and

even dangers to the operators. Another risk was finding the best hardware to suit all of the needs of the project.

## Feasibility

The Bike share project was started over a year before our team approached the project which should give an idea for the scope of the project. There has been a multitude of groups ranging over a variety of disciplines ranging from graphic design to engineering, and therefore the project required strong communication between groups to ensure the entire project came together as intended. To ensure that our work was as productive and successful as possible we worked closely with our project director Mark Kargol as well as other faculty involved in the project such as James Heise and Mani Mina. We utilized the ideas and work of previous groups to create a foundation for our work.

The largest feasibility issue we encountered was the large scope of this project. When we first met with our advisor, he requested a rework of the docking station, Android and iOS phone applications, and user and admin facing web applications with a multitude of features in each application. As we spent time in the first couple of weeks evaluating the project and what still needed to be done we decided this was much too large in scope. We narrowed this scope to make the project more feasible by agreeing, with our advisor, to only produce an Android application and to narrow the range of features that would be available in both applications. We also considered GPS tracking of the bicycles which ended up being too complicated with the limited power available on each bike. Other hardware hurdles we had to overcome was determining what type of RFID readers to use and where to get them as there were a plethora of options but many were much too costly or were from questionable sources. Another problem we encountered was incomplete bike stand hardware designs. This made it difficult to finalize component placement on the docking stations.

Another important feasibility consideration was accessing sensitive Iowa State student information. We assumed we would not have access to this information and, after meeting with ISU ITS, our assumption was confirmed. Therefore we worked with them to determine that all the data that we would be able to use and mocked all of the data using the exact format that Iowa State uses on their servers.

## Testing

### Hardware

For the testing of electrical hardware, we subjected the components to overvoltage tests as well as extreme temperature tests to simulate the harsh Iowa weather as well as fluctuations in input voltage. We conducted stress tests in which we left the system on for extended periods of time and later verified that the system was still functional. This was necessary as the system will be operating 24/7 when in use.

## Web Application

We were able to test the security of the server through manual and automated scans. We only found one vulnerability in the application. This vulnerability (below image) was that over HTTP connections the passwords were sent over plain text. We were able to fix this issue by using backend Django authentication methods instead of using HTML POST requests. This would also not be an issue since the final server would be running over HTTPS instead of HTTP. Iowa State Technology Services was unwilling to issue SSL certificates for a senior design project. We were also able to perform automated security assessment on the host that was running the web application and we verified that there was no security risks on the host itself. Our automated security scans were also able to detect a few configuration issues that were not a security risk but helped to make our server work better.

**LOW** Web Server Transmits Cleartext Credentials >

---

**Description**

The remote web server contains several HTML form fields containing an input of type 'password' which transmit their information to a remote web server in cleartext.

An attacker eavesdropping the traffic between web browser and server may obtain logins and passwords of valid users.

**Solution**

Make sure that every sensitive form transmits content over HTTPS.

**Output**

```
Page : /signup/login/  
Destination Page: /signup/login/  
  
Page : /signup/register/  
Destination Page: /signup/register/
```

| Port ▼           | Hosts  |
|------------------|--|
| 8000 / tcp / www | bikeshare-server.student.iastate.edu <a href="#">🔗</a> |

The web application was tested primarily with use of Django unit tests. This form of testing was used to test the REST API, website flow, and functionality. We used python's built in coverage report to verify coverage of 90% of code lines and manually tested lines that were difficult or impossible to test with Django unit tests. Within the Django unit tests we used Model Mommy, a tool to easily create Django models for testing. Below is shown the output of our test cases and a coverage report for the testing that was completed at the time of this report. The signup and rest\_api apps were yet to be tested which is where most of the missed lines are seen. We also tested load time on the Google Maps API that we used on the home page within the web application. We found this to be very slow which led us to discovering a bug in our code where the web app was making many more REST API calls than necessary. This was fixed and the map now loads virtually instantaneously.

```

=====
ERROR: test_login_get (signup.tests.test_signup.BikeViewsTestCase)
-----
Traceback (most recent call last):
  File "/home/bikeshare/bikeshare/django_webapp/signup/tests/test_signup.py", line 34, in tes
t_login_get
    self.assertTrue('form' in resp.context[-1])
  File "/home/bikeshare/bikeshare/django_virtual_env/lib/python3.4/site-packages/django/templ
ate/context.py", line 71, in __getitem__
    raise KeyError(key)
KeyError: -1
-----

Ran 63 tests in 9.039s

FAILED (errors=1)
Destroying test database for alias 'default'...
(django_virtual_env)bikeshare@debian:~/bikeshare/django_webapp$ █

```

| Name                                | Stmts | Miss | Cover |
|-------------------------------------|-------|------|-------|
| analytics/__init__.py               | 0     | 0    | 100%  |
| analytics/admin.py                  | 4     | 0    | 100%  |
| analytics/apps.py                   | 4     | 4    | 0%    |
| analytics/forms.py                  | 10    | 0    | 100%  |
| analytics/models.py                 | 33    | 2    | 94%   |
| analytics/tests/__init__.py         | 0     | 0    | 100%  |
| analytics/tests/test_history.py     | 44    | 0    | 100%  |
| analytics/tests/test_transaction.py | 42    | 0    | 100%  |
| analytics/urls.py                   | 3     | 0    | 100%  |
| analytics/views.py                  | 12    | 0    | 100%  |
| base/__init__.py                    | 0     | 0    | 100%  |

|                                  |     |   |      |
|----------------------------------|-----|---|------|
| base/admin.py                    | 5   | 0 | 100% |
| base/forms.py                    | 23  | 0 | 100% |
| base/models.py                   | 43  | 4 | 91%  |
| base/tests/__init__.py           | 0   | 0 | 100% |
| base/tests/test_contact.py       | 44  | 0 | 100% |
| base/tests/test_maintenance.py   | 71  | 0 | 100% |
| base/urls.py                     | 3   | 0 | 100% |
| base/views.py                    | 66  | 3 | 95%  |
| bike_admin/__init__.py           | 0   | 0 | 100% |
| bike_admin/admin.py              | 7   | 0 | 100% |
| bike_admin/forms.py              | 14  | 0 | 100% |
| bike_admin/models.py             | 50  | 1 | 98%  |
| bike_admin/tests/__init__.py     | 0   | 0 | 100% |
| bike_admin/tests/test_bike.py    | 65  | 0 | 100% |
| bike_admin/tests/test_dock.py    | 99  | 0 | 100% |
| bike_admin/tests/test_station.py | 69  | 0 | 100% |
| bike_admin/urls.py               | 3   | 0 | 100% |
| bike_admin/views.py              | 136 | 9 | 93%  |
| bikeshare/__init__.py            | 0   | 0 | 100% |

|                             |             |            |            |
|-----------------------------|-------------|------------|------------|
| bikeshare/settings.py       | 22          | 0          | 100%       |
| bikeshare/urls.py           | 16          | 0          | 100%       |
| bikeshare/wsgi.py           | 4           | 4          | 0%         |
| manage.py                   | 6           | 0          | 100%       |
| rest_api/__init__.py        | 0           | 0          | 100%       |
| rest_api/admin.py           | 1           | 0          | 100%       |
| rest_api/apps.py            | 4           | 4          | 0%         |
| rest_api/serializers.py     | 37          | 0          | 100%       |
| rest_api/tests.py           | 1           | 0          | 100%       |
| rest_api/views.py           | 87          | 43         | 51%        |
| signup/__init__.py          | 0           | 0          | 100%       |
| signup/admin.py             | 39          | 11         | 72%        |
| signup/forms.py             | 28          | 10         | 64%        |
| signup/models.py            | 24          | 3          | 88%        |
| signup/tests/__init__.py    | 0           | 0          | 100%       |
| signup/tests/test_signup.py | 26          | 0          | 100%       |
| signup/urls.py              | 3           | 0          | 100%       |
| signup/views.py             | 38          | 19         | 50%        |
| <b>TOTAL</b>                | <b>1186</b> | <b>117</b> | <b>90%</b> |

## Android Application

### Load Time

With any mobile application, load times of different functionality is very important to success of the application because something every user values is time. There were a few different cases that we wanted to test load times as well as the load times of individually pages to insure that the application was running smoothly for our users. We used a network analysis tool to verify the calls were being made to our server and verifying correctly. After that, we stubbed out the functionality that we wanted to test and were able to time when it started, until an all clear signal was sent back from our server or everything was loaded onto the page. We wanted to keep the load time of individual pages under 5 seconds and the map loading all markers within 5 seconds as well.

| Test Case  | Load Time   | Met Criteria |
|--|-------------|--------------|
| Click on map, time for markers to become visible on the map    | 2.7 seconds | Passed       |
| Click on map marker, time for information to pop up            | .4 seconds  | Passed       |
| Submit maintenance request, time until server receives request | 1.2 seconds | Passed       |
| Submit contact request, time until server receives request     | 1.3 seconds | Passed       |
| Login to application, time until verified by server            | 1.2 seconds | Passed       |
| Load time of map page  | 2.7 seconds | Passed       |
| Load time of ride data page                                    | .4 seconds  | Passed       |
| Load time of account info page                                 | .3 seconds  | Passed       |
| Load time of contact us page                                   | .3 seconds  | Passed       |
| Load time of maintenance request page                          | .3 seconds  | Passed       |

### Flow of Application

In order to get complete coverage of application flow, we used Android unit tests with a mix of manual testing to make sure everything was working together with each other. The two main areas we checked for flow was the navigation and forms. With the navigation we created tests that would start at a certain

page in the application, click on another page and verify it was on the correct page and that page had loaded. We also did manual testing to make sure changing pages did not cause errors or issues. With the forms we wrote tests to make sure that the data in the form was verified by doing data validation unit tests. We also used manual testing to make sure the forms submitted correctly and the correct errors popped out. At the end of the testing process, we made sure to check every page to make sure they looked good and complete.

| Test Case  | Unit Test Results                        | Manual Test Results |
|--|--|---------------------|
| Can navigate from station map to all other pages         | Passed                                   | Passed              |
| Can navigate from ride data to all other pages           | Passed                                   | Passed              |
| Can navigate from account info to all other pages        | Passed                                   | Passed              |
| Can navigate from contact us to all other pages          | Passed                                   | Passed              |
| Can navigate from maintenance request to all other pages | Passed                                   | Passed              |
| Cannot navigate until logged into application            | Failed, Fixed, Passed before Manual Test | Passed              |
| Can only select actual issues for maintenance dropdown   | Passed                                   | Passed              |
| Can only select actual reasons for contact requests      | Passed                                   | Passed              |
| Only enter bike number of a bike in the system           | Passed                                   | Passed              |
| Only enter dock number of a dock in the system           | Passed                                   | Passed              |

|  |  |        |
|--|--|--------|
| Empty extra information<br>pops up a warning on submission | Passed   | Passed |
| Submit request clears all information on page              | Failed, Fixed, Passed before<br>Manual Testing | Passed |
| Map Page Visual Check                                      | -  | Passed |
| Ride Data Page Visual Check                                | -  | Passed |
| Account Info Page Visual Check                             | -  | Passed |
| Contact Us Visual Check                                    | -  | Passed |
| Maintenance Request Page Visual Check                      | -  | Passed |

## Schedule

### Project Schedule

- September 2015
  - Evaluated the Project Status
  - Met with Multidisciplinary Groups
- October 2015
  - Gathered Requirements
  - Worked on Project Plan
- November 2015
  - Circuit Design & PCB Design
  - Google Maps Prototype in Android Application
  - Server Setup
  - Finalize Design Document and Project Plan
- December 2015
  - Hardware Connected
  - Troubleshooting/Testing
  - Developing web application code
- January 2016
  - Working hardware
  - More Advanced Prototypes of Android and Web Applications
  - API Communication from Dock to Server
- March 2016
  - PCB Design Complete and Sent in for Manufacturing

- Completed testing plan
- April 2016
  - Code reworked on the Pi to better suit our hardware
  - Testing plan executed
- May 2016
  - Final documentation
  - Finalized, working hardware
  - Finalized Web and Android applications

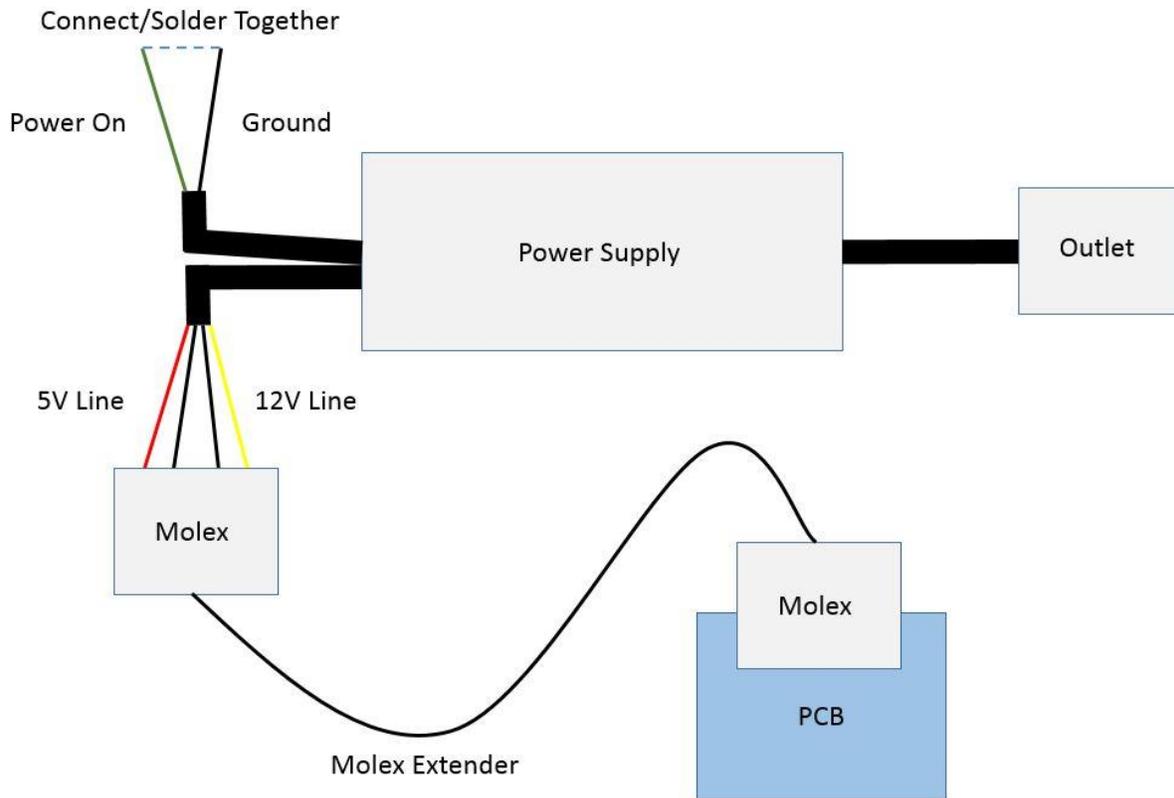
## Conclusion

The Bike Share project is the first system of its kind to be designed completely by students and at a cost that is significantly less than any system that currently exists. We have taken a simple prototype and developed it into a complex functioning system. The Bike Share will provide the University and Ames with a low cost alternative to existing bike share options while the students, faculty, and the community can utilize the free and easy to use campus and community service.

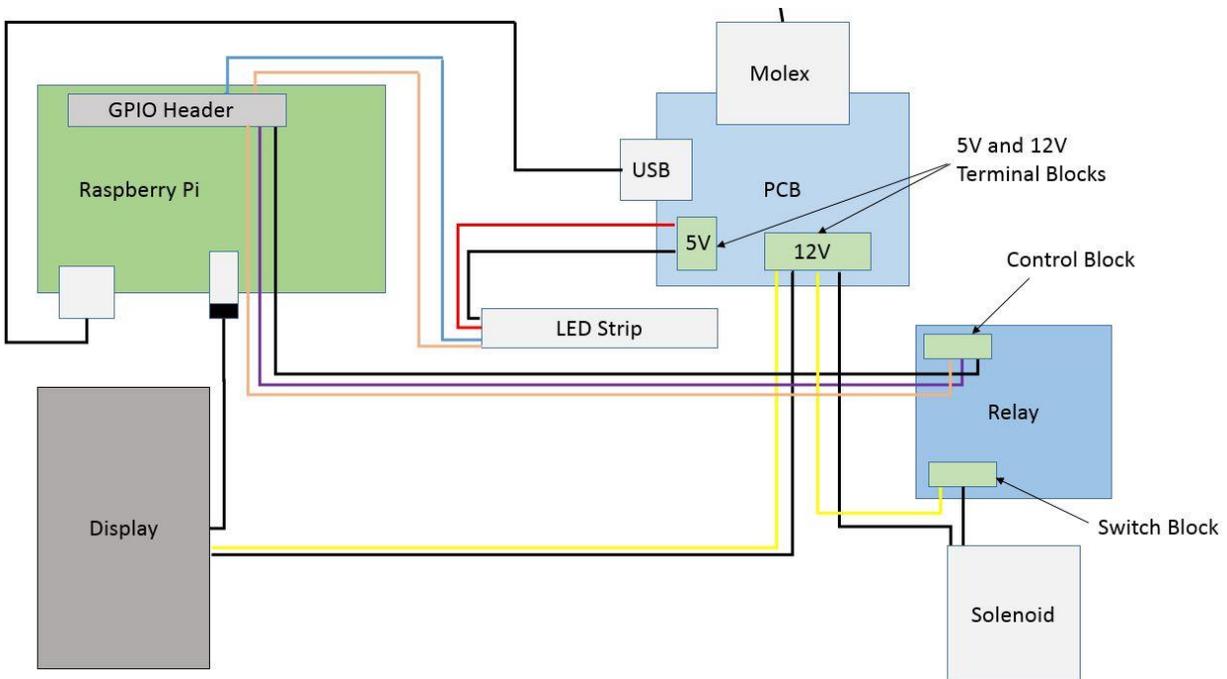
The project was an overhaul of last year's Bike Share project. The existing hardware was improved upon to be more efficient, cleaner, and cheaper. Additional software was added for functionality and usability. After two semesters, we now have a working bike-share system in place where bikes can be checked in and out by students, faculty, and members of the community. Administrators have their own web-page, users can check their account and access services online. An android application is also available for extended functionality. The Bike Share project developed from a simple prototype to a functional bike sharing system for the University and the Ames community.

## Appendix I: Operation Manual

We will begin with setting up the hardware; this includes everything that would go inside the bike docking station. First things first: power. Make sure the outlet you are plugging the power supply into has a good ground connection, otherwise the lights in the system could behave erratically. Also make sure that the power-on pin of the power supply is hooked to a ground pin so that the supply runs as soon as it's plugged in. Attach one end of the MOLEX extender cable to the MOLEX port of the power supply and the other end to the custom PCB. This PCB routes power to the rest of the system. This setup is shown by the following diagram:



The LED strip, raspberry pi, solenoid/relay, and display all hook up to the PCB. The display and solenoid/relay (one wire of the solenoid and one wire of the relay) hook up to the 12V-out on the PCB, while the LED strip hooks up to the 5V-out on the PCB. The raspberry pi hooks up to the USB. This setup is shown in the following diagram:



Also shown in this diagram is that the display, LED strip, and relay must all be hooked up to the raspberry pi as well. The display simply hooks up to the A/V out on the raspberry pi. The LED strip has a data pin and clock pin in addition to its two power cables, which must be hooked up to GPIO pins 23 and 24 on the raspberry pi respectively. The LED strip must also only have 24 LEDs, and thus a strip of 144 LEDs will be cut into 6 pieces for 6 stands. Finally, the relay has three connections which go the raspberry pi: a supply pin, a ground, and a trigger pin. The supply pin and ground pin can go to any 3.3V and GND pin respectively on the raspberry pi, while the trigger pin must be hooked to GPIO 21.

Not shown in either of these diagrams are the two USB readers and the level switch (the switch is not shown as we never received the piece and were therefore unable to test it, but the switch would simply plug into the GPIO header on the Pi). These can simply be plugged into any unused USB port on the raspberry pi. Make sure to have these two USB readers as well as all of the other pieces that route to the raspberry pi plugged into the raspberry pi before powering the system.

Now that we have the hardware all set up for the stand has been powered on, we can set up the software portion of the dock. From here we can plug a keyboard in the raspberry pi and use the built in display to configure the dock. The display will have a shell pulled up on the screen; if necessary, you will log in with the username "pi" and the password "raspberrypi". You are now logged in as the default user. You will now need to run the command "sudo su root" in order to get a root shell on the system. The first thing we will do is update the software on the raspberry pi to do this we will issue the command "apt-get upgrade && apt-get update". We will edit the file /etc/network/interfaces to in order to set up the networking on the dock. To do this we will issue the command "nano /etc/network/interfaces". We will assume for sake of the demo that we are using wired ethernet. We will need to modify the device so it will get an IP address on the appropriate network. We will need to know the IP address (remember this, we will need it later) - we intend to use the DNS server, the netmask, and the gateway. When your

config file looks like the example below, hit `Ctrl`+`x` to save your work and hit `y` to confirm these changes.

```
# The primary network interface
allow-hotplug eth0
iface eth0 inet static
    address 10.10.231.123
    netmask 255.255.255.0
    network 10.10.0.0
    broadcast 10.10.231.255
    gateway 10.10.231.254
```

Now that we have the networking set up, we can test this by pinging an outside server with the command “ping -n 8.8.8.8”. This should show successful responses from the server. Now we need to download the code to the raspberry pi. Take the flash drive with the code on it and run the command “cd /home/pi && mkdir rpi”. This will take us into the pi users home directory and make a folder called rpi. We will now issue the command “cd /mount/pi/<NAME OF FLASHDRIVE>” where <NAME OF FLASHDRIVE> is the name of your flashdrive. We will then issue the command “mv \* /home/pi/rpi” this puts all the files we need on the pi, and we can remove the flashdrive. We will also need to edit the file bike\_id in the rpi folder and type the bike id with the command “nano /home/pi/rpi/bike\_id”. Now all we need to do is configure to have the program start at boot. To do this we will issue the command “crontab -e” this will pop open a text editor where we will have to add the line “@reboot python /home/pi/rpi/main.py &” now all we need to do is reboot the pi with the command “reboot” and the pi will be running the code. After verifying that all communications are working correctly you can put the dock hardware in its housing.

## **Bike Dock**

### Check-In Bike

Scan Iowa State card on RFID scanner

Lights will flash green if successful

Lights will flash red if unsuccessful

Screen will report dock status

Direct bike into the locking mechanism

Lock will close behind bike

### Check-Out Bike

Scan Iowa State card on RFID scanner

Lights will flash green if successful

Lights will flash red if unsuccessful

Screen will report dock status

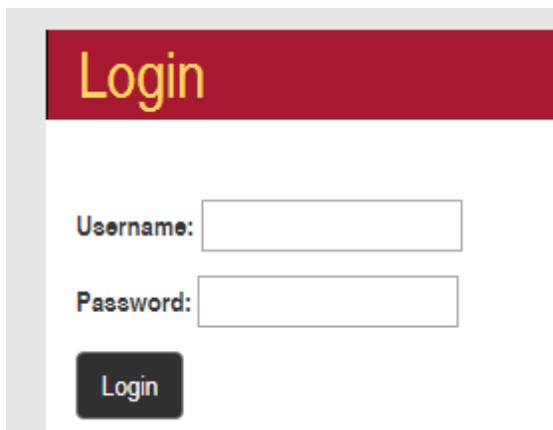
Locking mechanism will unlock

Direct bike out of locking mechanism

Enjoy bike

### **Admin Web Application**

#### Login to System

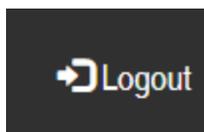
A screenshot of a web application login page. At the top, there is a dark red horizontal bar with the word "Login" in a yellow, sans-serif font. Below this bar, on a light gray background, are two white input fields. The first is labeled "Username:" and the second is labeled "Password:". Below the password field is a dark gray button with the word "Login" in white text.

Enter username

Enter password

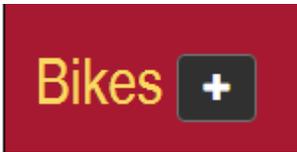
Click Login Button

#### Logout of System



Click Logout button in top right of application

#### Add Bike



Click on the "+" button after Bikes

Add name of Bike

Add status of Bike

Click save

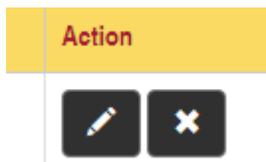
#### Delete Bike



Click on the "X" for the selected bike

Confirm deletion

#### Edit Bike



Click on the pencil icon for the selected bike

Change name of bike

Change status of bike

Click save to confirm changes

#### Add Dock



Click on the "+" button after Docks

Enter name of dock

Enter station dock belongs to, or add station

Enter bike currently in dock, or add bike

Enter status of bike

Click save

### Delete Dock



Click "X" for the selected dock

Confirm deletion

### Edit Dock



Click pencil icon for selected dock

Edit name

Edit station

Edit bike

Edit status

Click save

### Add Station



Click the "+" after stations

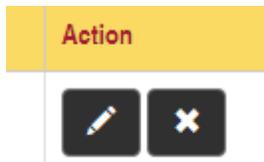
Change the station name

Change the latitude of location

Change the longitude of location

Click save

### Delete Station



Click "X" for selected station

Confirm deletion

### Edit Station



Click the pencil icon for selected station

Change the name

Change the latitude

Change the longitude

Click save

### View Users



Click on Users under the Data page to view the Users

### View History



Click on History page to view the History of transactions

### Add Maintenance Request



Click on the “+” after Maintenance Requests

Change the item of request

Change the bike/dock needed to change

Change the issue

Add details

Click save

### Change Maintenance Request Status



Click on the pencil icon after the selected request

Change the status drop down to new status

Click save

### Add Contact Request



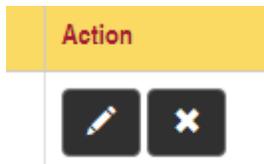
Click on the “+” after contact requests

Change the reason of the request

Change the details for the request

Click save

### Change Contact Request Status



Click the pencil icon of the selected request

Change the status drop down to new status

Click save

### Search Individual Pages



Type the name of what you are looking for on selected page

Website will filter out the information to what you are looking for

## Appendix II: Alternative Versions of the Design

### Hardware Alternatives

We went through several instances in which we purchased parts in which we felt were not quite up to the standards of our project. We initially wanted to go with a touchscreen rather than a display, but the touchscreen proved too difficult to waterproof and still be usable. It also costed much more than our current display and used a bulky ribbon cable that rendered most other pins on the raspberry pi's GPIO header unusable. We decided our new display which simply plugged into the A/V out on the raspberry pi was a much better fit.

We also considered the idea of putting a flex sensor on the locking mechanism of the dock in order to have an extra measure to tell that a bike was checked into the dock. Ultimately, we were concerned that a flex sensor would not provide a well-defined cutoff for telling if a bike was in or not, and we decided

that a lever switch would be better for this. Unfortunately due to shipping problems, we were not able to receive the lever switch in time for testing and implementation.

Lastly, we had many discussions on where the RFID tag on the bike should be placed for the most consistent reading. Ultimately we decided on two possible locations. One possible location was on the side of the main steering column, and the other was extended a bit out from the column that runs down by the front wheel. We decided on the latter in the event that a well for the wheel was introduced to the stand to ensure proper alignment and reading of the bike ID. We also had concerns about the former being easy to scrape against the dock or possibly break off when docking the bike into the station.

### Software Alternatives

When we took on the project there was currently a Java spring web application being hosted on Heroku server. Heroku is an online web hosting service. After talking to ITS and other Iowa State employees it was obvious that we needed to have the code hosted here at Iowa State and built on a virtual machine so that it was easily portable. Through working with ETG we were able to get a secure server setup to host our application. We then evaluated the existing web application which had minimal functionality and would be trivial to re-implement to the current state. Since none of us were familiar with the current Java Spring Framework we took the time to evaluate its security built into spring and other frameworks. We decided that spring has issues that make something hard to implement in a professional and secure way. We were aware of the Django Python Framework that is secure by design and is hard to code in an insecure way.

### Appendix III: Other Considerations

This year of senior design caused great growth within our team. Prior to starting the project, no one on our team had experience with the Django web service which is the complete backbone of our system because it handles all the views, data, and API calls coming from the bike dock and Android application. This caused a good amount of stress in the first semester learning everything and getting all the environments set up. However, once those were all setup and the tutorials were went through, we were able to start gaining ground on the project.

Another thing we learned was how implementing a project into the Iowa State system can cause a lot of problems because of the tight security measures they need to follow and the censored information that our application wants to use. This brought a lot of meetings with our client, Iowa State and figuring out the correct direction of how different data should be stored or what technology we should use. The project itself is a proof of concept design, which helped provide a little more experimental ways to attack the project and getting things to work.