BIKE SHARE

FINAL DOCUMENT

TEAM

May 15-09

DATE

Tuesday, April 28, 2015

MEMBERS

Austin Adam
Mohammed Alkatheeri
Derrick Anderson
Sean Cavanaugh
Jeremy Curtiss
David Kominek
James McGinnis
Derek Otoo
Caleb Schulze
Brian Simons
Joe Ternus
Nick Ziegeweid

ADVISOR

Mani Mina

CLIENT

Mark Kargol

TABLE OF CONTENTS

INTRODUCTION

The project we were tasked with was proposed by the Government of the Student Body (GSB) at Iowa State University called Bike Share. The GSB Bike Share project is the first known collegiate level bike sharing program in the country. It is an initiative to provide a bike-sharing service to Iowa State University and Iowa communities. The project requires considerable work from a collection of multi-disciplinary teams to create a working solution. Mechanical, electrical and software systems must be developed to provide a robust and expandable program. Our portion of the project consisted of creating the code and infrastructure to manage the checking in and checking out of bikes in the system. The job of integrating the bikes and docks that other team's created into a fully functional system also fell upon us. This integration took a lot of time collaborating and meeting with the other teams as well as our advisor, Mark Kargol. There are already solutions on the market, so the GSB BikeShare project is expected to provide a solution at or below the cost of alternative solutions. We hope that this will provide alternate transportation to students, reduce traffic congestion, and lessen emissions that are harmful to the environment.

One of the main focuses of this project consists of creating a system that is very user friendly. Our implementation uses the existing infrastructure provided by the ISU card system. This allows us to implement a "one touch checkout" which is a great improvement over existing bike sharing systems which implement temporary codes and other complicated checkout methods. We firmly believe that simplicity of use is the greatest factor in success of any project. In our system a student need only touch their card to any dock with a bike available. No kiosk. No additional input.

What we have seen from other bike share projects is that they have to perform a long analysis of a working system and make constant changes to the number of bikes and docks at any given location at any time. In other systems this can be difficult and requires a technician to add and remove docks from a location. Our system will strive to avoid that problem as well. Our system has independent docks, meaning that every dock is self contained and can be moved independently of the others. In other systems the kiosk method is implemented which causes docks to have to be connected to a singular input/output device which allows checkout. This device has advantages when a complicated checkout process needs to be accommodated. However the drawbacks are needing to run connections to all the docks from the kiosk which makes redistribution of the docks a difficult process. Our system overcomes this drawback by being "Plug-N-Play". Meaning any dock can be placed in any location at any time and be redistributed at will, no kiosk required. Each dock needs only be provided electricity and WiFi and can run independently. This allows non technical savvy administration of the system. Including winter or other weather removal, dock redistribution, and unlimited expansion at any time.

PROJECT DESIGN

SYSTEM REQUIREMENTS

FUNCTIONAL REQUIREMENTS

- When a user swipes their ID card then the system will unlock a bike for the user.
- When a user puts the bike into the station then the system will lock and check in the bike.
- When a user checks in a bike and marks it as damaged then the system will make the bike unavailable to be checked out.
- When an unauthorized user attempts to check out a bike then the system will not unlock a bike.
- When an unauthorized user attempts to check in a bike that does not belong then the system will not lock the bike.
- When an administrator performs maintenance on a damaged bike then the system will make the bike available to be checked out.
- An administrator will be able to view bike transactions and filter it by bike ID, station ID, student or faculty ID, or date.
- Keep track of the time at which a bike was checked out and in along with the users information.

NON-FUNCTIONAL REQUIREMENTS

- Only students and faculty should be able to checkout a bike
- The system shall be weather proof.
- The checkout process should take less than 5 seconds.
- All bike transactions should be kept for 2 years for auditing purposes.

FUNCTIONAL DECOMPOSITION

HIGH LEVEL DECOMPOSITION

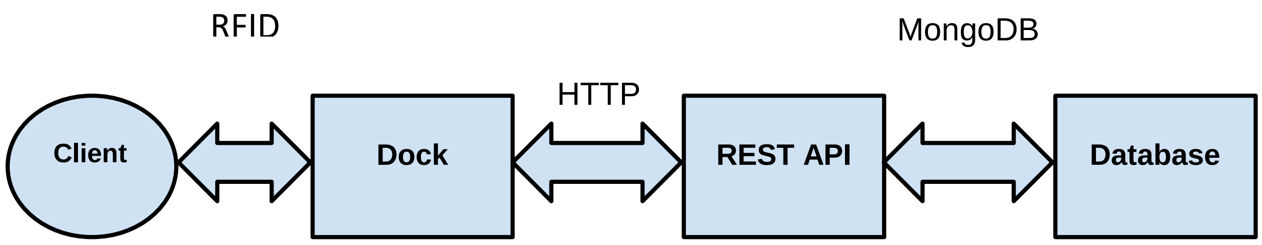


FIGURE 1. HIGH LEVEL DEPICTION OF THE SYSTEM

At a high level, the bike share system can be decomposed into several modules. At heart, the system is a fairly standard distributed network. The physical station interacts with users through RFID cards and with the central server through HTTP requests to the REST API. The REST API responds to the docking stations through HTTP and accesses the Database through MongoDB protocol.

HARDWARE/SOFTWARE SPECIFICATIONS

SYSTEM OVERVIEW

The system will incorporate various RFID readers for inputs, and use RGB LEDs in various colors to indicate dock status and transaction success/failures to the user. When the user swipes an ISU card, the system will go through three steps before unlocking the bike to the user. If one of these steps is not successful, the system will not unlock the bike. First, the system will check if the bike had been marked as damaged or if it is useable. Second, the system will check with the server if this user is allowed to check out a bike, as users may only check out one bike at a time. The third step is to ensure that the user is authorized to use the system, and is enrolled in the Iowa State University user database. Only Iowa State students and facility will be allowed to use the system. If all this steps are successful, the green LED will light up for five seconds and the system will unlock the bike. Otherwise, a red LED will light up indicating an error.

When a user returns a bike to the dock, the system will follow a similar process to check the bike in. Once the bike is completely inserted into the dock, the RFID reader inside the dock will read the bike's unique ID number from the RFID tag on the bike. The bike ID number will then be sent to the server, where the bike ID is verified belonging to our system, and not already checked into a different dock. If the check in is allowed by the system, the dock LEDs flash green and the bike is locked. If there is an anomaly detected with the bike ID being checked in, such as the bike already being checked in, the system will lock the bike in the lock and freeze the dock disallowing the bike to be checked out. The dock color will change to yellow, and a flag will be set on the admin web page indicating that dock is in a locked state.

HARDWARE SPECIFICATIONS

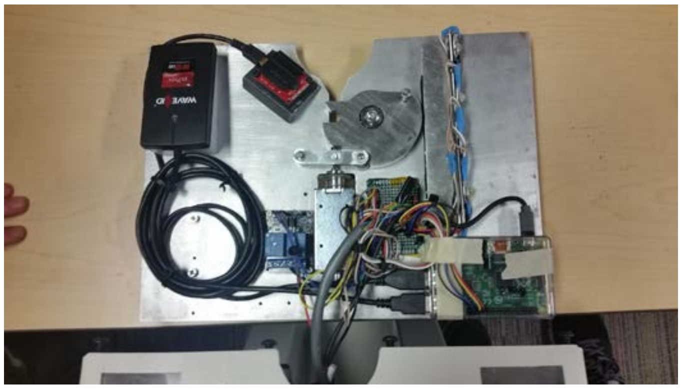


FIGURE 2. FINAL DESIGN AND LAYOUT ON THE DOCK

The system will incorporate various RFID readers for inputs, and use RGB LEDs in various colors to indicate dock status and transaction success/failures to the user. When the user swipes an ISU card, the system will go through three steps before unlocking the bike to the user. If one of these steps is not successful, the system will not unlock the bike. First, the system will check if the bike had been marked as damaged or if it is useable. Second, the system will check with the server if this user is allowed to check out a bike, as users may only check out one bike at a time. The third step is to ensure that the user is authorized to use the system, and is enrolled in the Iowa State University user database. Only Iowa State students and facility will be allowed to use the system. If all this steps are successful, the green LED will light up for five seconds and the system will unlock the bike. Otherwise, a red LED will light up indicating an error.

When a user returns a bike to the dock, the system will follow a similar process to check the bike in. Once the bike is completely inserted into the dock, the RFID reader inside the dock will read the bike's unique ID number from the RFID tag on the bike. The bike ID number will then be sent to the server, where the bike ID is verified belonging to our system, and not already checked into a different dock. If the check in is allowed by the system, the dock LEDs flash green and the bike is locked. If there is an anomaly detected with the bike ID being checked in, such as the bike already being checked in, the system will lock the bike in the lock and freeze the dock disallowing the bike to be checked out. The dock color will change to yellow, and a flag will be set on the admin web page indicating that dock is in a locked state.

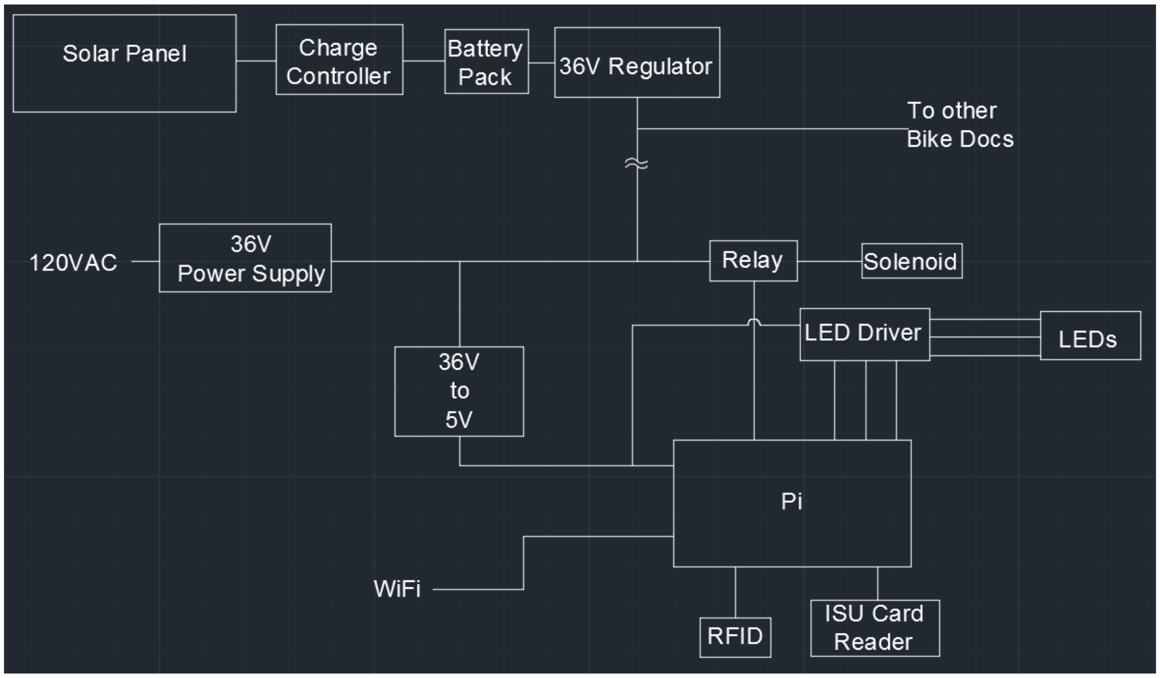


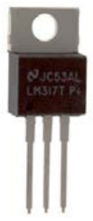
FIGURE 3. BASIC CONCEPTUAL ELECTRICAL DIAGRAM

SYSTEM POWER

120 VAC TO 36 VDC POWER SUPPLY

We chose to use a 9.7A Regulated Switching power supply because this model will allow us to step down and convert the 120 VAC from a grid tie to 36 VDC which will be used to power solenoid and the rest of the circuit.

36 VDC TO 5 VDC REGULATOR CIRCUIT



For the regulator circuit we chose to use the OKI-78SR series in order to bring 36 VDC to a usable 5 VDC to power the Raspberry Pi. The reason we decided to build a regulator circuit instead of a voltage divider was because they automatically maintain a constant voltage level independent of how much power is drawn from the line, thus ensuring that our solenoid will always have enough power to open the lock. The 78SR can be very efficient, even at high differential input-output voltages, and can maintain this voltage without overheating.

SOLAR PANEL

As an alternative energy option, we chose to spec out the needed solar power components in order to have a green alternative power option. We decided that the panels should be able to power eight bike locks at a time and will be great for rural areas with limited access to the electric grid.

BATTERY BANK

The battery bank will store the power that is collected from the solar panels and will be stored within the battery cells. The three 12VDC battery bank will be a great fit for the solar side of this project while also being able to hold enough charge for several hours, enough to power the locks and all of the electrical components.

CONTROL

CHARGE CONTROLLER

The primary function of this charge controller is to protect the battery from an overcharge and block reverse current. Without the charger, the life of the batteries will mostly likely be compromised quick.

INVERTER

An inverter is an electronic device or circuitry that converts direct current (DC) to alternating current (AC). This would change the power coming from the solar panels/ battery banks into a usable AC voltage/current for the bike rack components.

RELAY

The relay we are using is an electrically operated switch. This will be used when it is necessary to control a circuit by a low-power signal. In our case the relay will be sending a signal to the solenoid when the bike needs to be unlocked.

RASPBERRY PI

The Raspberry Pi is a credit-card sized computer board that we have programmed to run our bike transactions, control the bike lock, and allow access using the RFID card reader. The Raspberry Pi has a voltage input of 5V and is able to run several outputs.

PUSH BUTTON

For user interaction a push button will be used as a simple switch to control when the user wants to report damage..

RFID

RFID otherwise known as Radio Frequency Identification, meaning that it is a device or a technology that allows communication between two components when they are connected or come within a proximity. These will be placed on the bike and locks to let us know when the bike is being locked.

HID-RFID

These are proprietary RFID readers, which use electromagnetic fields to transfer data for keeping track of objects automatically. These RFID readers allow for ISU students and faculty to use their ISU cards to rent these bikes.

OUTPUT

SOLENOID

A solenoid is a type of electromagnet with the purpose of generating a controlled magnetic field in order to operate a switch which in our case, will operate the lock mechanism. We chose to use a 36V solenoid because of its durability, size, and its strength to pull open the lock.

LEDS

The LEDs are being used for the user interface telling them the status of the bike rack. Within this system we are using an set of RGB LEDs which can only emit red, green, and blue light. However through the combination of those lights all colors can be made. On the dock the different colors represent different statuses. There are two indicators which temporarily show and general status colors which show while the dock is powered and now showing an indicator. The indicators are red and green which represent success and failure. The status colors are white and blue. White shows when a bike is present in the dock and blue shows when a bike is absent from the dock. The final function is that if any light is present then the dock is powered on.

IMPLEMENTATION OF PROJECT

CENTRAL SERVER

OVERVIEW

The software systems consists of two pieces of code: the Representational State Transfer (REST) API and the UI. Both of these portions exist in the same code base, but each's respective duties provide a natural separation. Due to this separation we will discuss each portion's implementation and design individually.

DATA STORAGE/DATABASE

The CyBike system contains a database of Transactions, Bikes, Docks, and ErrorReports to model our system. The use of these models helps to keep the systems complexity low by enforcing a strict set of properties that each object must conform to.

The Dock model represents a dock that is placed on campus. To keep track of these, we give this model a unique dockID, current bikeID, location, and status. When a bike is checked into the dock, we update the dock's bikeID property to match the bike that was checked in. The location is a string representation of where the dock is located on campus and can be updated by administrators. The status represents whether the dock is currently active and allowing bike transactions to occur.

The Bike model represents a bike that is in our system. To keep track of these, each bike is given a bikeID, studentID, dockID, state, and locked attribute. The bikeID is the uniquely identifying property that each bike will have. The studentID is the ID of the current student that has the bike checked out (or null if the bike is in a dock). The dockID is the ID of the dock that the bike is currently checked into (or null if the bike is checked out). The state of the bike can be one of "in", "out", or "maintenance". If that state is currently "maintenance", that means the bike has been removed from the system temporarily, so a staff member can perform maintenance on the dock. Lastly, the locked attribute of the bike represents whether transactions can be done on the bike. A bike may be locked for reasons such as weather, season, or time of day.

The Transaction model represents a checkin or checkout of a bike in our system. Each transaction is given a transactionID, studentID, dockID, bikeID, date, action, and success attributes. The transactionID is the uniquely identifying property that each transaction will have. The studentID is the ID of the student that requested the transaction. The dockID is the ID of the dock where the transaction happened. The bikeID is the ID of the bike that the transaction is occurring on. The date is the date and time in which the transaction occurred. The action is an indication of whether the transaction is a checkin or a checkout. Lastly, the success attribute will show if the transaction completed successfully or if it failed. A transaction could fail if the bike is in the locked state, if the student has a biked checked out and tries to checkout another bike, or any other situation where a student was unable to get the bike checked out or checked in.

The ErrorReport model represents an error that has occurred somewhere in the software system. Each Error Report is given a stackTrace, dockID, date, and type. The stack trace contains the information about the state of the system when the error occurred. The dockID is used to identify which dock errored in case of a client error. The date is when the error occurred. The type is either "server" or "client". If the server code throws an error, it is considered a "server" error. An example server error would be if a staff member attempts to add a bike to the system but the ID they provide already exists on a bike. If anything running on a Raspberry Pi errors, it is considered a "client" error. An example of a client error would be if the card reader on a dock received an invalid card.

These models can be seen below:

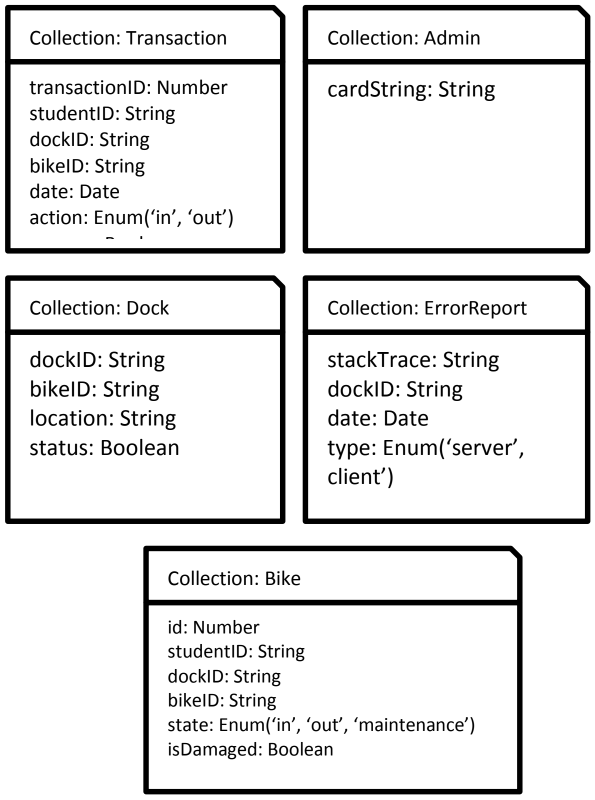


FIGURE 2. WEB SERVICE DATA MODELS

REST API

The main goal of this portion of the code is to control and record all bike transactions in the system. In order to keep record all of these transactions several models were created to represent this information (Dock, Bike, and Transaction) and to control these models several HTTP endpoints were created for each model. To manage the system as a whole, additional endpoints were added. All of these models and endpoints can be seen Figure 2. To implement this REST API design, a Full-Stack JavaScript solution comprised of MongoDB, Express, Angular, and Node was chosen - MEAN. Furthermore, using GitHub and Heroku allowed the creation of a continuous integration pipeline to easily deploy our application.

HTTP ENDPOINTS

The CYBIKE system contains many HTTP endpoints to manage and manipulate the above data storage modules. The REST API and its endpoints follow the REST API standard in several ways. First, when using the URL without a trailing "/ID" it will interact will all of the given models and when a trailing "/ID" is given it will interact with the model with the given ID. Secondly, any get endpoints will get all model(s) of the requested type, a post will create a new model instance of the requested type, and finally, a put will replace a model with a different version of the model of the requested type. There are other HTTP methods that we chose to not implement, because they were not needed or could disrupt the system. There were several benefits to following the

REST API standard such as allowing anyone who is familiar with REST APIs to quickly use our system and to create a set of concise endpoints with a large variety of functionality.

The documentation for each endpoint contains several crucial pieces of information. First is the name of each end point. Secondly, is the URL at which the endpoint can be interacted with, so the use case "Get Docks" can be accessed at *www.domain.com/api/dock*. Next, the HTTP Method used to interact with the endpoint is listed, so to access the endpoint "Get Docks" a get would be needed, while a post would be needed for "Create Dock". Next is the Parameter(s) for each endpoint as each endpoint will need to be provided different information and sometimes in different ways. Finally, the response is listed, whether it be a HTTP status code, a single JSON object, or an array of JSON objects. All of these endpoints can be seen below:

TABLE 1: REST API ENDPOINTS

Name	URL	Method	Params	Response
Check in	/api/checkin/	POST	dockID: string bikeID: string	200/500 HTTP
Check out	/api/checkout	POST	dockID: string bikeID: string cardString: string	200/500 HTTP

TABLE 2: DOCK ENDPOINTS

Name	URL	Method	Params	Response
Get docks	/api/dock/	GET	N/A	{ { status: boolean, dockID: string, bikeID: string, location: string}, ... }
Get dock by ID	/api/dock/	GET	Inline Param	{ status: boolean, dockID: string, bikeID: string, location: string }
Create dock	/api/dock/	POST	dockID: string location: string status: boolean	200/500 HTTP
Create dock	/api/dock/	PUT	dockID: string, location: string status: boolean	200/500 HTTP

TABLE 3: BIKE ENDPOINTS

Name	URL	Method	Params	Response
Get bikes	/api/bike/	GET	N/A	{ { cardString: string/null, bikeID: string, dockID:string/null, state: enum: ['in', 'out' 'maintenance'] , isDamaged: boolean}, ... }
Get bike by ID	/api/bike/:ID	GET	Inline Param	{ cardString: string/null, bikeID: string, dockID: string/null, state: enum: ['in', 'out' 'maintenance'] , isDamaged: boolean }
Create bike	/api/bike/	POST	bikeID: String	200/500 HTTP
Update bike	/api/bike/	PUT	dockID: string cardString: string isDamaged: boolean	200/500 HTTP

TABLE 4: TRANSACTION ENDPOINTS

Name	URL	Method	Params	Response
Get transactions	/api/transaction	GET	N/A	{ { bikeID: string, dockID: string, studentID: string, date: date, action: string, success: boolean }, ... }

TABLE 5: ERROR REPORTING ENDPOINTS

Name	URL	Method	Params	Response
Get error reports	/api/errorreport	GET	N/A	{ { stackTrace: string, dockID: string, date: date}, ... }

Get error reports by dockID	/api/errorreport/:dockID	GET	Inline Param	{ { stackTrace: string, dockID: string, date: date}, ... }
Create error report	/api/errorreport	POST	trace: string dockID: string	200/500 HTTP

ADMIN UI

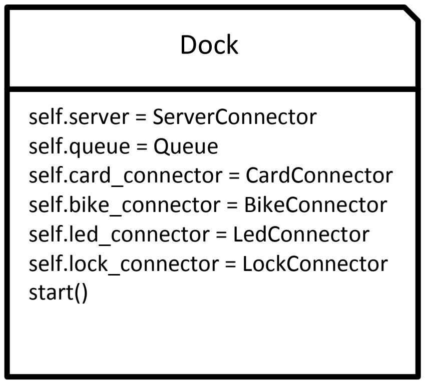
The goal of the UI was to create a simple and intuitive web application that administrators could visit to monitor and manage bikes, docks and transactions. It is designed to be a single-page application to reduce load times between different views and to promote simplicity. Using AngularJS allowed us to implement this easily using AngularJS Directives. We also wanted to allow the users to use almost all of the HTTP endpoints through the UI, so they are able to add, edit, or lock bikes and docks.

DOCKING STATIONS

OVERVIEW

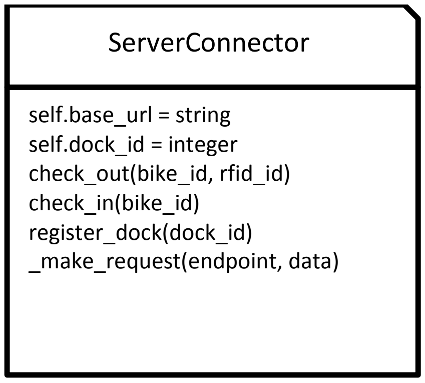
The dock software is written in python, and runs on the Raspberry Pi. The Raspberry Pi is setup with the Raspian Linux distribution, a Debian based distro designed specifically for use with the Pi. The RFID readers are plugged into the USB ports on the Pi, and are accessed using a serial connection. The data is read by each specific connector, and then added to a queue for processing by the main thread. This allows for requests to be made to the centralized server asynchronously of hardware components being accessed by users. The RGB LEDs are controlled by three separate GPIO pins on the PI, one pin for each color. The GPIO pins connect to transistors on 5V voltage regulator, allowing us to easily control the colors of the LEDs from the Pi without drawing too much current directly from the GPIO pins. There is also a single GPIO pin used as a control signal for the relay, which will trigger the 36V solenoid to pull the locking mechanism and release the bike to the user.

MODULE DECOMPOSITION


Dock
self.server = ServerConnector
self.queue = Queue
self.card_connector = CardConnector
self.bike_connector = BikeConnector
self.led_connector = LedConnector
self.lock_connector = LockConnector
start()

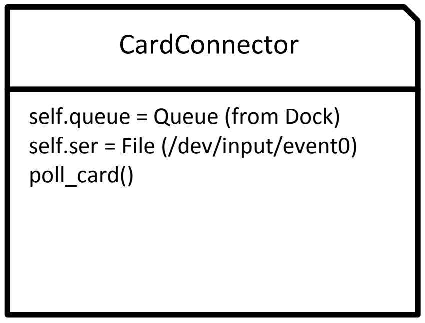
DOCK CLASS

The Dock class is the main entry point of the dock software. The Dock class creates instances for all the different connectors: server, card, bike, led and lock. The class also contains a queue which is passed to the CardConnector and BikeConnector and is used to inform the Dock class that an event occurred. These connectors populate the queue with events as they occur which are then handled in the main loop of the dock. When an event occurs, the dock's main loop checks the sender and responds appropriately. If the event is from the CardConnector, the event is a checkout (the user swipes their card to use a bike), while an event from the BikeConnector is interpreted as a checkin event (the bike is slid into the dock). If the dock is in the prerequisite state for the event, the server is contacted through the ServerConnector. Regardless of success or error, the dock informs the LedConnector to notify the user.



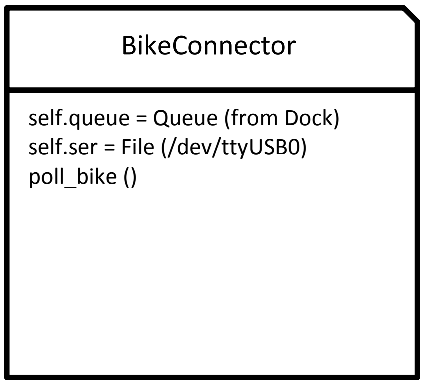
SERVERCONNECTOR CLASS

This class is responsible for transmitting all check in and checkout events to the server, as well as registering the dock on startup. The main Dock thread is given a handle to this class, which allows for web requests to be made asynchronously and allow for smoother user interaction.


CardConnector
self.queue = Queue (from Dock)
self.ser = File (/dev/input/event0)
poll_card()

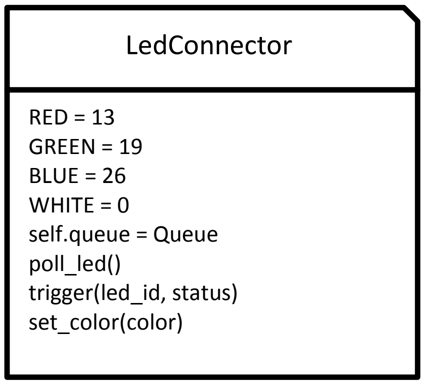
CARDCONNECTOR CLASS

When this class is initialized, it will kick off a new thread to read the card data from the ISU card reader. When a new card read event occurs, the card ID is pushed onto the dock queue for processing.


BikeConnector
self.queue = Queue (from Dock)
self.ser = File (/dev/ttyUSB0)
poll_bike ()

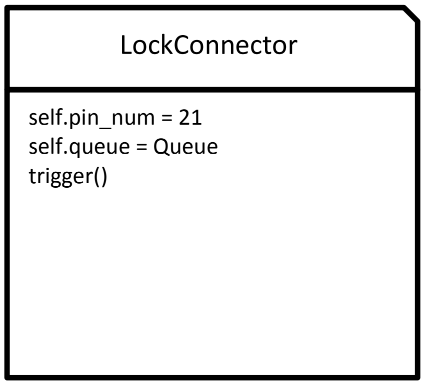
BIKECONNECTOR CLASS

This class is almost identical to the CardConnector, except it reads the RFID tags of bikes which get checked into the dock. These bike IDs are then pushed onto the dock queue for processing.



LEDCONNECTOR CLASS

The LedConnector controls output to the user in the form of Leds on the dock. The output is set to green or red for 3 seconds (depending on an event success or failure) and then returns to the idle state, which is either white or blue depending on the state of the dock.



LOCKCONNECTOR CLASS

The LockConnector is responsible for triggering the solenoid when a checkout occured. This is the only time this module is used.

TESTING PROCESS

WEB API TESTING

The REST API was tested using Mocha, Chai, and SuperTest. Mocha is a JavaScript testing framework, Mocha is a behavior driven development and test driven development assertion library, and SuperTest is a HTTP assertion library. With these three frameworks, we are able to easily test our API's HTTP endpoints and verify they are responding with the intended response by using functional tests. What these functional tests allow us to do is have the tests actually go in and interact with the database. By having tests do this, it allows us to view the system working as a whole and ensuring our endpoints and UI are functioning correctly. Also, another positive is that we can easily test our requirements through behavior driven development. For example, a test to verify that creating a bike and adding it to the system could be done. This test would hit the correct HTTP endpoint for creating a bike, create the bike and add it to the database. After that is done, the test would then verify that the created bike has the correct data and that the api call responded with the correct response (a 200 in this case).

CONCLUSION

The Bike Share project is a ground breaking idea that has the potential to become implemented at campuses around the nation. It includes a spin of an already successful system that will easily allow communities access to efficient transportation.

By making each dock modular, new stations can be easily implemented as the demand grows. Additionally, mapping and application based programs for mobile devices could be included as the program develops. As the docks spread, solar power can allow for placement in remote locations.

It has been fun to see where this project has grown and can continue to thrive. There is likely room for cost savings, but soon enough, the system should be available for mass production and further additions.

APPENDIX 1: OPERATION MANUAL

USER – DOCK

Interaction with the docking stations is achieved primarily through the sensors on the dock system. As the dock has no display other than the LED, it was important to create a simple UI which is easy for the user to understand through color feedback.

BIKE CHECKOUT

1. To checkout, the user places card on dock activating HID card reader
    a. The dock LED should flash green to indicate that the server has accepted the checkout
2. The solenoid is engaged, pulling the locking pin and disengaging the locking mechanism
    a. The user is able to remove the bike from the docking station

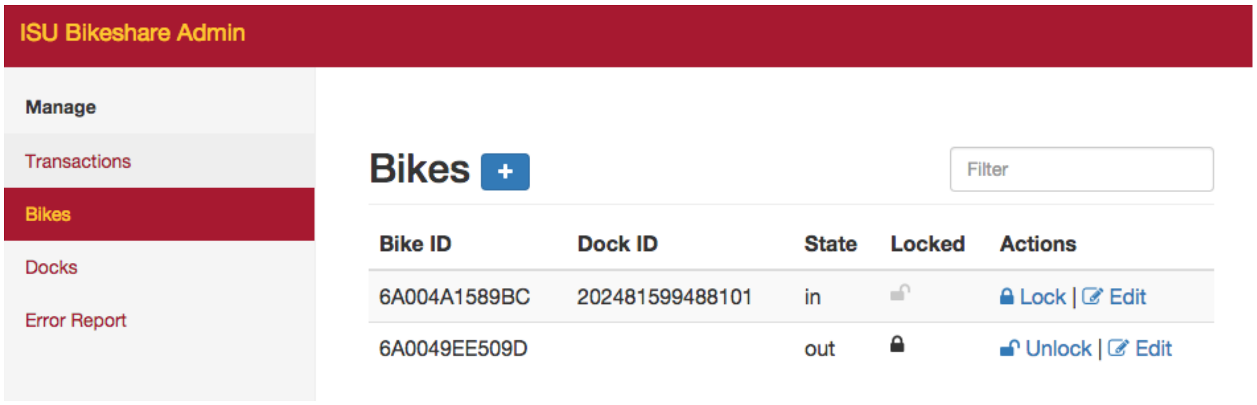
BIKE CHECK IN

1. User inserts the bike into any empty dock
    a. Green LED Flashes to indicate that the bike was successfully checked in

ADMIN - WEBSITE

ADD BIKE

1. Navigate to Bikes page via the sidebar.
2. Select blue button with "+" next to Bikes header. *See Figure 3 for details*
3. In the window that appears, enter the new Bike's Bike ID in the text field. *See Figure 4 for details.*
4. Click "Add" button.



Bike ID	Dock ID	State	Locked	Actions
6A004A1589BC	202481599488101	in	🔓	Lock | Edit
6A0049EE509D		out	🔒	Unlock | Edit

FIGURE 3. BIKES PAGE IN ADMINISTRATOR USER INTERFACE

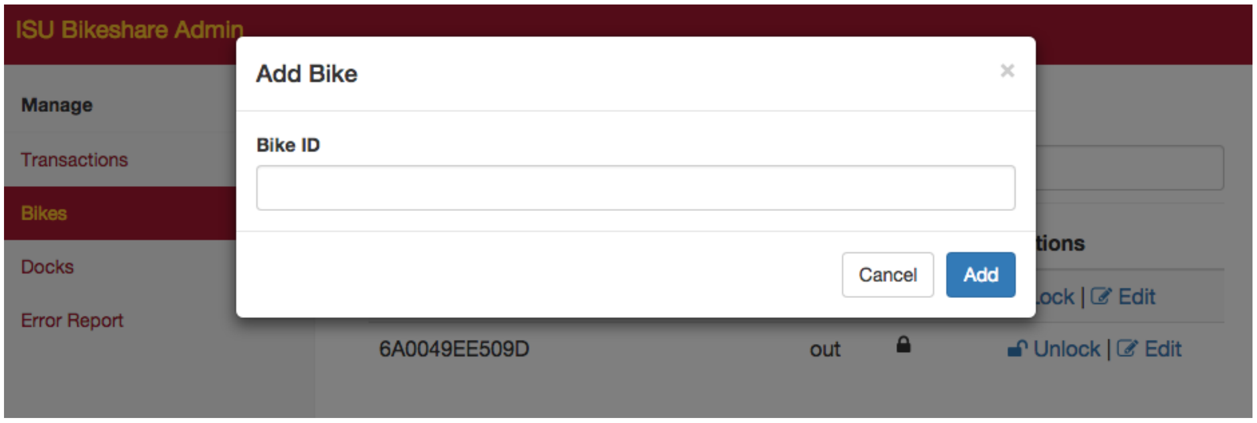


FIGURE 4. ADD BIKE WINDOW

LOCK BIKE

1. Navigate to Bikes page via the sidebar.
2. Find the desired bike to lock, then click "Lock" which appears under the "Actions" column.
3. In the window that appears, click "Yes". *See Figure 5 for details.*

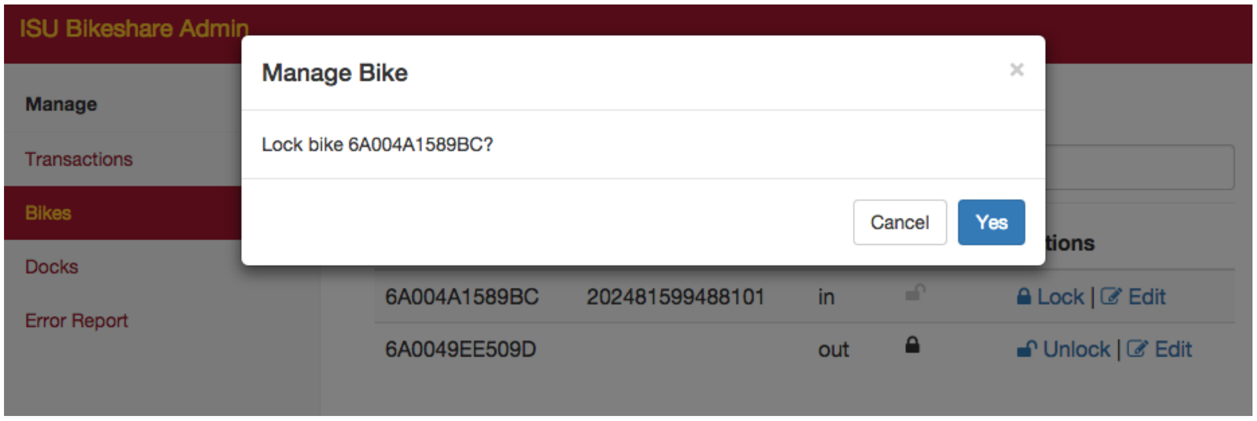
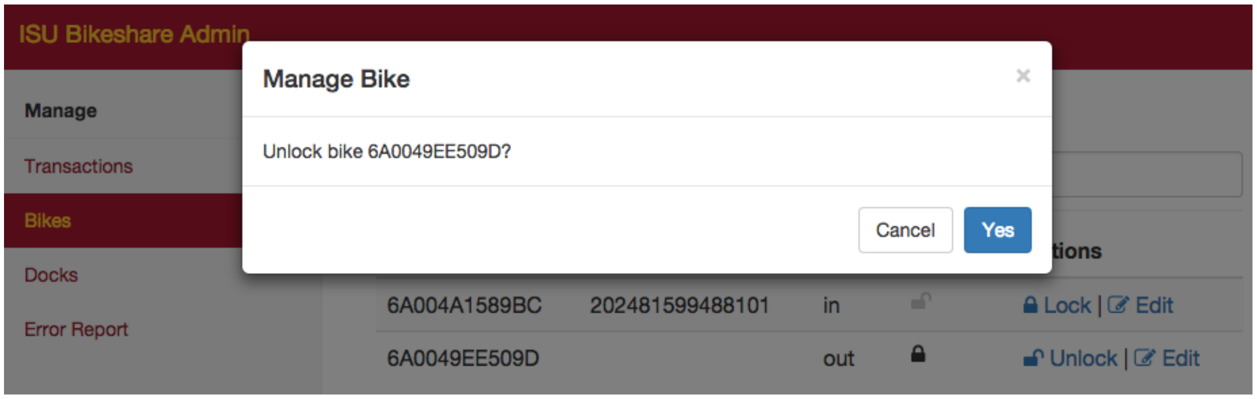


FIGURE 5. LOCK BIKE WINDOW

UNLOCK BIKE

1. Navigate to Bikes page via the sidebar.
2. Find the desired bike to unlock, then click "Unlock" which appears under the "Actions" column.
3. In the window that appears, click "Yes". *See Figure 6 for details.*


ISU Bikeshare Admin

Manage Bike

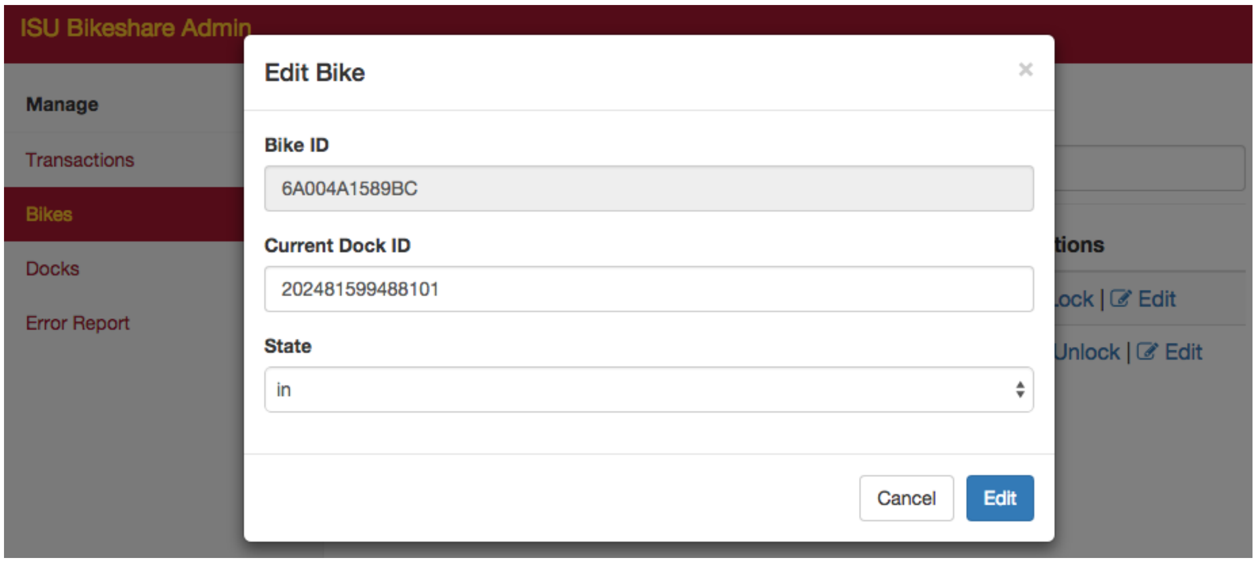
Unlock bike 6A0049EE509D?

6A004A1589BC	202481599488101	in		Lock | Edit
6A0049EE509D		out		Unlock | Edit

FIGURE 6. UNLOCK BIKE WINDOW

EDIT BIKE

1. Navigate to Bikes page via the sidebar.
2. Find the desired bike to edit, then click “Edit” which appears under the “Actions” column.
3. In the window that appears, edit Current Dock ID or state. Once completed, click “Edit” in the bottom-right corner of the window. *See Figure 7 for details.*


ISU Bikeshare Admin

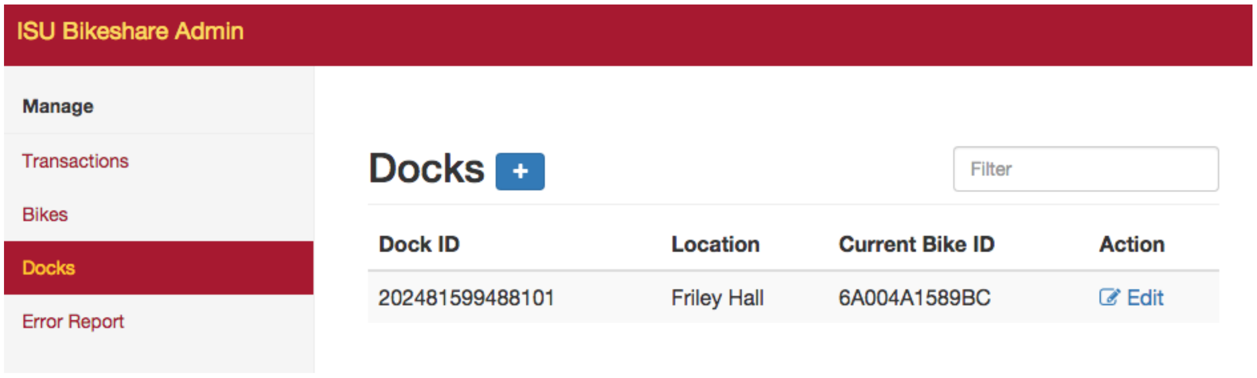
Edit Bike

Bike ID: 6A004A1589BC

Current Dock ID: 202481599488101

State: in

FIGURE 7. EDIT BIKE WINDOW


ISU Bikeshare Admin

Manage
Transactions
Bikes
Docks
Error Report

Docks +

Filter

Dock ID	Location	Current Bike ID	Action
202481599488101	Friley Hall	6A004A1589BC	Edit

FIGURE 8. DOCKS PAGE IN ADMINISTRATOR USER INTERFACE

ADD DOCK

1. Navigate to Docks page via the sidebar.
2. Select blue button with “+” next to Docks header. *See Figure 8 for details.*
3. In the window that appears, enter the new Dock’s ID in the text field. Optionally, you may supply a location. *See Figure 9 for details.*
4. Click “Add” button.

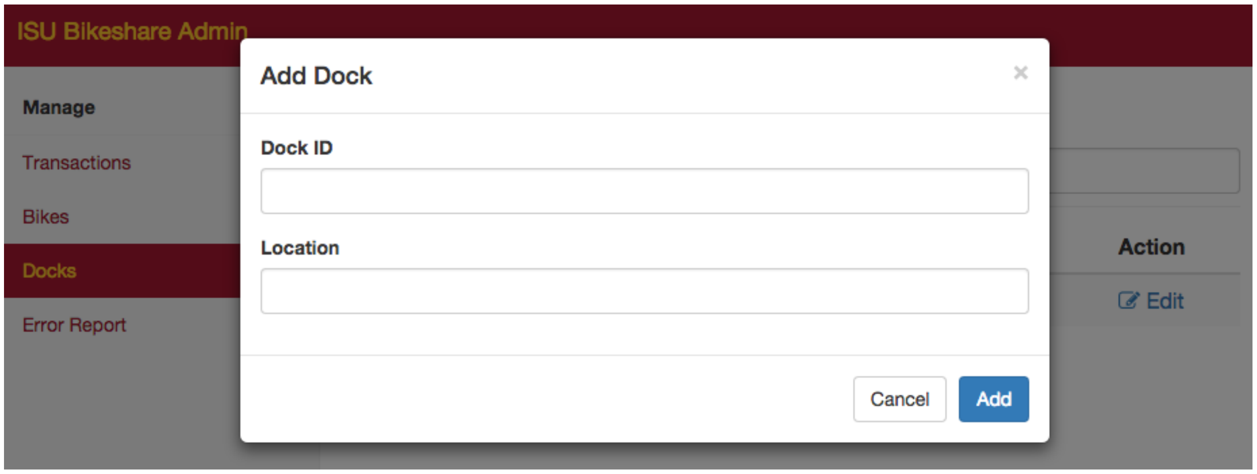
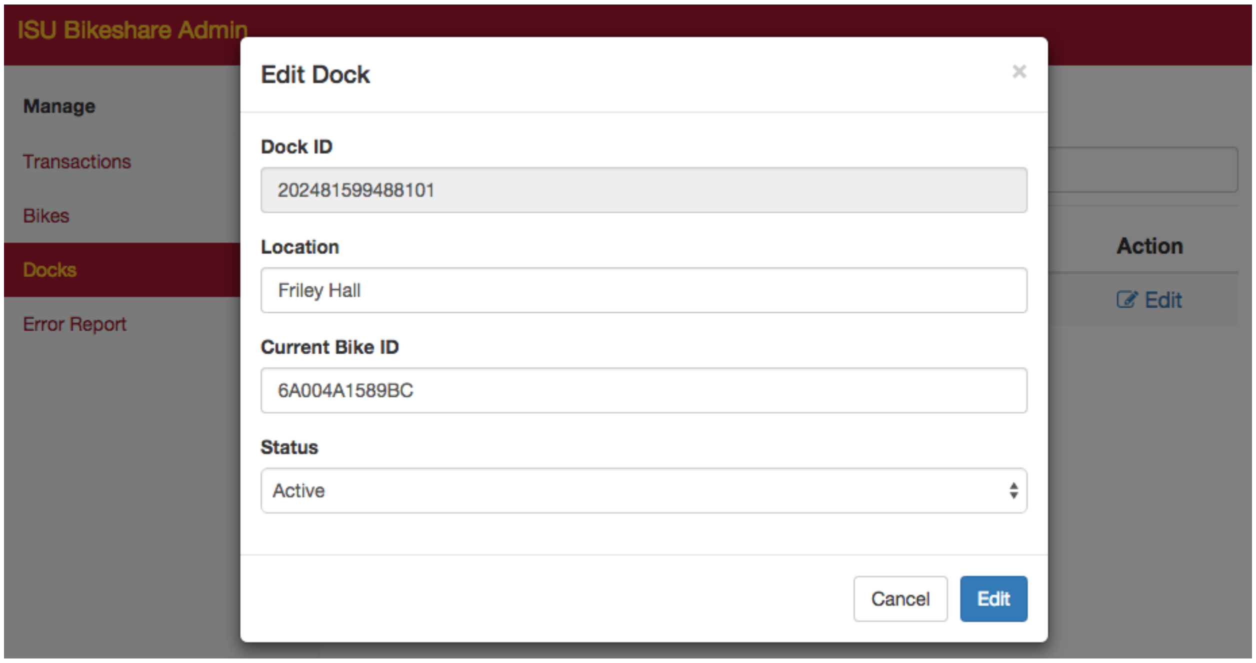


FIGURE 9. ADD DOCK WINDOW

EDIT DOCK

1. Navigate to Docks page via the sidebar.
2. Find the desired dock to edit, then click “Edit” which appears under the “Action” column.
3. In the window that appears, edit location, current bike ID or status. Once completed, click “Edit” in the bottom-right corner of the window. *See Figure 10 for details.*



Edit Dock

Dock ID
202481599488101

Location
Friley Hall

Current Bike ID
6A004A1589BC

Status
Active

FIGURE 10. EDIT DOCK WINDOW

APPENDIX 2: OTHER CONSIDERATIONS

When it came to implementation, transitioning the circuits from breadboard to perf board proved to be difficult. This can be avoided by using a PCB and getting the circuit fabricated before inclusion.

Looking further into how to better the dynamo/light interaction within the bike will be useful. The swivel of the handlebars has caused problems. Along with that, efficient charging components that are relatively small in size could be included for future versions.

APPENDIX 4: CODE

Both the Web Interface and Raspberry Pi code is located on GitHub. The code can be seen at this link: https://github.com/ISUBikeShare