UNIVERSITY OF LA VERNE

LA VERNE, CALIFORNIA

Spring 2019

SENIOR PROJECT REPORT

J2 INNOVATIONS' AUTOMATIC RFID EMPLOYEE CLOCKING SYSTEM

A SENIOR PROJECT SUBMITTED TO:

THE FACULTY OF

COMPUTER SCIENCE AND COMPUTER ENGINEERING

IN CANDIDACY FOR THE DEGREE OF

BACHELORS OF SCIENCE

SOFTWARE / ENGINEERING

BY

KOPPANY HORVATH

Abstract

The purpose of this project is to design and implement an RFID based employee time tracking system for J2 Innovations, a building automation software company. The following hardware technologies were used: IoT enabled ESP 32 microcontroller, RFID scanner module, 3D printing, WiFi AP device configuration. In addition, a variety of software was used: Arduino IDE, NodeJS framework, Ractive library, AM Charts library, MySQL database. The user experience was greatly improved and user complaints were reduced to nearly none. Possible future expansions include handling different time zones for international employees, generating reports and charts for management, and scheduling days off and vacations through the UI.

Table of Contents

Chapter I: Introduction

Introduction

When tasked with overseeing employee work times and productivity, how that data is obtained and managed can be the difference between a good or bad user experience. The point of this study is to design and implement a system that can improve how employees log their work hours and tasks, along with making it easy for management to manage that information. The proposed design directly relates to the problem in that it attempts to solve the problems present in the previous system for tracking employee work times. This study relates to previous work in that it uses RFID technology to acquire parts of the data, but it is applied to tracking the time of employees rather than tracking the locations of materials.

Problem Statement

The employees and management working for J2 Innovations, a building automation software company, are frequently annoyed by the current time tracking system being used. One of the biggest problems that management complains about is that the system is a third party service that requires monthly payment per registered employee, and is not flexible in terms of adding new features or correcting problems with the service. An issue that bothers the employees is that with the current system employees have to manually type in their starting time and ending time which sometimes results in inaccurate information if an employee can't remember their starting time accurately. The project manager is sometimes bothered by not being able to see what the general schedule is for part time employees who have very dynamic schedules through the web interface of the current system. Employees also don't like having to submit a daily email containing their work for the day along with a weekly email containing not only the information from the daily emails, but what their times were for each day that they worked because the current system has no easy way for management to view this information.

Purpose Statement and Rationale

The purpose of this project is to design and build an RFID based employee time tracking system to address the issues caused by J2 Innovations' current time tracking system. This new system will solve the problem of monthly payment by running on a local server rather than as a remote paid service and will solve the problem of being inflexible by being made open source and being designed to be easily modifiable. This system will also solve the problem of employees having to manually type their times by using RFID technology to automatically record that information by swiping an RFID card on an RFID node. The web interface for this system will also allow management to view the schedules of employees and view their times along with the tasks they did throughout the day.

Method

The method of this study consisted of doing initial research to see what previous approaches to the problem had been attempted. Then more research followed to investigate a good way to implement the server software which would work to manage data and route data where it needs to be. Then research was done to see what kind of IoT configurations are generally easy to use.

After research, software implementation testing followed to determine the effectiveness of the chosen implementation. Shortly after software implementation began, the focus changed to hardware testing to see how well the components would fit together and to learn the concepts of IoT programming which would be used in the final hardware product.

Following hardware and software testing, production level implementations began. Hardware had the highest initial priority, with software coming in second in order to provide any web API support that the IoT part of the hardware required. After hardware had been finished, it was tested in 2 physical locations to make sure it worked. The software implementation continued on after that to implement all basic functionality needed by the end users.

Need of Such a Project

This project was necessary in order to address the problems described in the Problem Statement section. Most important of those problems was to make a dynamic system that did not require payment to a third party service which increases in proportion to the amount of employees using the system. This problem alone justifies the need for such a project as far as management is concerned. A quick poll of the employees also reveals that they collectively do not like the current system and even though they are skeptical about the effectiveness of this new system, they would still be willing to try the new system.

Contributions to Knowledge and Society

Unfortunately, due to the limited nature of the Senior Project, this project does nothing particularly special to contribute to humanity's collective knowledge. However, this project has been open-source since the start and is freely available on GitHub. It is hoped that this project might be used at other companies where a free employee time tracking system would be beneficial along with all the convenience features that it provides.

Limitations

The major limitations of this project in its current state are as follows. End users from different time zones other than the one that the server is being hosted at will notice that some of the timestamps will be incorrect. Management still doesn't have fine control over individual reports as the current implementation focuses on mass reports and data. From an IT standpoint, someone well versed in Linux based server systems will be required to install and maintain the server software. These issues are expected to be resolved in future updates of this project.

Chapter II: Literature Review

Historical Implementations

In conducting research, three notable implementations of digital employee time tracking systems were found. The first, Office Timesheet 2000, has the capability to clock workers in and out, along with being able to record time spent working on separate projects or tasks. The interface comes as a desktop application or as a primitive "web app". Timesheet 2000 uses a client-server architecture with the ability to store data in a variety of database applications. This system also supports dynamic reports and the capability to send automatic email notifications (Feibus).

The second time tracking system, TimeSheet Pro, is very similar to Office Timesheet 2000 in its main features, but is designed to be more configurable. This system can either use database applications to store data, or it can store data locally. Like Timesheet 2000, TimeSheet Pro can track projects and tasks and be accessed through a web browser. Unlike Timesheet 2000, TimeSheet Pro is more configurable in terms of holidays, task rules, and pay rules. TimeSheet Pro is also considered more difficult to use due to poor design of the user interface (Feibus).

The third time tracking system is called Employee Proj Clock. This system is designed as a small desktop application that communicates with a server to retrieve and transfer information. Each user can only edit their own information, or an administrator can edit anyone's information. The interface is simple in that the basic inputs are the project to be worked on, who the user is, when they started, when they ended, and any comments the user might have. This information is sent to the server and recorded into a database. A custom reports tool allows reports to be built based on what information is required. It is reported that Employee Proj Clock improves productivity and accuracy of time records when switching over from traditional time cards or manual reporting (Gruber).

Modern Implementations

During research, three modern examples of employee time tracking systems were discovered. The first, Syspro ShopClock, is a real-time data collection system for employee time and attendance. ShopClock can also track the time spent on jobs and materials used in performing a task. This system gives management a GANTT chart that updates in real time to display the status of employees. ShopClock allows for shifts, labor rates, and overtime pay to be defined in advance. ShopClock can also integrate with barcode readers to update the materials used and current inventory. When employees are clocking in or out, ShopClock will alert the user if there is a problem with the information that they are inputting (Syspro).

The second system is Replicon's Cloud Clock, which is a cloud based time clock. This implementation is unique in that it uses an internet connected hardware clock that the users interact with while giving management the power to monitor who clocks in and when. Along with the hardware clock, Replicon offers its Cloud Clock software as a service. This allows management to validate attendance records and ensure compliance of work policy. Replicon reported that compliance for employees submitting completed timesheets increased by 85% while reducing payroll errors from 8% to just 1.9% in less than a year (Replicon).

The last example is the TSheets employee time tracking system. What is unique about this system is how much effort the developers put into making the system as fun and easy to use as possible. TSheets has moved away from relying on dedicated hardware for tracking time and instead uses a cloud based solution with phone apps and web apps so that employees can use any device they want to clock in and out. TSheets also logs employees locations as they use the system so that employees can't cheat on attendance reports because management will know that they weren't really where they said they were when they clocked in. TSheets even has a rating system so that they can know how satisfied employees are with the overall system and its features (Computers).

RFID Systems

Due to this project using RFID technology, research on examples of using RFID technology on humans was necessary. Unfortunately, using RFID for the purpose of tracking employee times does not seem to be something that has been reported in any journals, and thus the following examples are used for purposes other than employee time tracking. The first example is a study trying to determine the best way of utilizing RFID technology in a hospital setting to track tools and equipment throughout the building as well as patients. Of greatest interest in this study were the results and conclusions that the research group came up with while reviewing different RFID implementations. In the comparison between active and passive RFID technology, it was found that while passive RFID has a smaller detection range, it is still a better option due to the RFID tags being smaller and cheaper when compared to active RFID tags. During testing with actual passive RFID hardware, the researchers found out that the RFID tags could only be read when the face of the tag was parallel to that of the RFID reader, and it had trouble being read when in direct contact with a human body or when it was attached to some objects with a high liquid content. Eventually it was discovered that passive RFID technology has many limitations, but is acceptable because of the cost and size factors (Najera).

The second example is a study attempting to determine the effectiveness of RFID technology in collecting data from a construction site in order to be analyzed for better decision making to improve productivity. Like the previous study, the researchers in this study came to the same conclusion that passive RFID technology was better suited than active RFID technology because the tags are smaller and cheaper, even despite certain problems caused by RFID signals being attenuated by metal beams within the walls of the construction site. Also like the previous study, these researchers found that passive RFID technology has a relatively small detection range, but they were able to solve this problem by strategically positioning the RFID readers in places that would come as close as possible to where materials and workers with tags would be passing by such as doorways (Costin).

Chapter III: Analysis

Feasibility Study

In studying other works to determine the feasibility of this project, along with what requirements are needed for this project, a conclusion was reached in that this project is quite feasible. This conclusion was reached because all of the necessary hardware and software technologies could be found as commercial off-the-shelf modules and libraries that would make implementation of this project possible within the deadline of the Senior Project.

Analysis of Findings From Previous Chapter

Based on the research from the previous chapter, there are common patterns that seem to be standard of employee time tracking systems. The biggest commonality that stuck out was the use of client-server architecture when implementing such a system. One of the systems, TimeSheet Pro, had the ability to store data locally, but could also use a remote database if necessary. Otherwise, the other systems exclusively used a client-server setup to keep the stored data centralized on a server and give the end users either a web interface to work with, or API calls for desktop applications that interface with the server.

The next most common implementation feature was the use of a web page interface to interact with the employee time tracking system. With the exception of Employee Proj Clock, the other platforms at least offered some form of web interface, even if a desktop application was the main interface for that system. In comparing the more modern systems with the older systems, it appears that a web app style interface is preferred over a desktop application style interface.

One smaller common feature that was found to be common during research is adding features other than employee time tracking. This seems to give the system a greater market advantage because everything can be managed with one system as opposed to multiple systems. Examples of this are tying it into the inventory system as in SysPro's ShopClock. Another example is tying in physical time tracking hardware as in Replicon's Cloud Clock.

In terms of RFID technology implementation, a preference between the two RFID types was detected as well. With RFID technology, there are two kinds, active and passive. Active RFID transmits data from the RFID tag to the RFID scanner, passive RFID is powered by the RFID scanner and communicates through backscattering. Because active RFID transmits by itself, it requires a power source such as a battery, which when compared to passive RFID, makes it a very bulky option since passive RFID is capable of data transfer without having any on-board power. Unfortunately, since passive RFID is wirelessly powered through the RFID scanner and does not broadcast by itself, the data transfer distance is very short, usually a few centimeters for low cost off-the-shelf modules.

Hardware and Software Choices

After understanding the basics of an employee time tracking system along with the basics of RFID implementations, choices can now be made about the kind of hardware and software that would be needed to accomplish a project that combines both together. Because the software is going to be supporting the hardware in this project, it would be helpful to know about the hardware first. For RFID communication, a passive approach is preferred due to its low cost. The MFRC522 RFID module satisfies this requirement and is commercially available, even at the hobbyist level. Since the RFID node will also need some data processing capabilities along with network communication to send data to a main server, the decision to go with the ESP32 microcontroller was made. This microcontroller is relatively cheap, has WiFi networking capabilities, is easy to program, and is also commercially available.

With the main hardware components having been chosen, choosing what software to use comes next. Because of the need of a main server that the hardware will communicate to and that users will interface with, the Node.js framework was chosen. This allows for the customization of every aspect of the web server and the API that will be needed to manage everything. Since the server will also be used for serving up the web interface for the users, the standard languages of the web will be used, HTML, CSS, and JavaScript. In order to facilitate faster development of the interface, a dynamic template framework known as Ractive will be used on the client side as well as the AM Charts library to aid in rendering charts of employee work and progress. For data storage, a MySQL database will be used due to its proven reliability and ubiquity.

Chapter IV: Design and Implementation

Overview

In order to put the pieces together and build this project, everything had to be designed first. Since there is an idea of how the users will use the system, that provides an entry point into the basic system design. The user will scan their RFID card with the RFID node. The RFID node will report to the server that the user has just scanned their card. The server will clock the user in for the day. From there the user can use the web interface from the server to interact with the data. A diagram of this basic architecture can be found in Appendix C as Figure 1.

Knowing what the architecture of this project is helps to generate a workflow for the users, which in turn will help to better design the software to meet the workflow. Figure 2 in Appendix C shows the basic workflow of the user. When users walk into work in the morning, they scan their RFID card with the RFID node. This clocks the user in and allows them to start working. As the user works on tasks or projects that they have been assigned, the user can use the web interface to log them individually. This gives management a better idea of what was done between clocking in and clocking out that day. Once the user has finished working for the day, the user scans his card as he walks out to indicate that he is clocking out. The next day the cycle is repeated and the system keeps track of it all.

RFID Hardware

Having defined the functionality of the hardware of this project, it can now be designed fully. The hardware, referred to as “the RFID node”, must be able to scan the user's RFID card when presented with it, and report this to the server for recording purposes. As is generally considered good design, the hardware should also provide some sort of feedback to the user if scanning and reporting to the server was successful or if it had failed.

In the Hardware and Software Choices section of Chapter III, two of the main hardware components were already described. The first was the MFRC522 RFID module, this is the part that will be able to read the unique serial number from an RFID card so that it knows which user has been scanned in. The second was the ESP32 microcontroller, this part interfaces with the RFID module, does whatever processing is required of the data, then packages it up and sends it to the server to be recorded.

However, more components are needed since these two alone do not provide feedback to the user, nor do they provide any form of reconfiguration of the hardware. To address the issue of feedback, it would be best to have both auditory feedback and visual feedback in the event that the user either can't hear the feedback (due to environmental noise) or cannot see the feedback (due to not having a line-of-sight) to the device. The simplest and most common choice for visual feedback is an LED, this would allow it to blink with certain patterns based on what type of feedback the RFID node is trying to convey. For auditory feedback a small speaker would work to provide different tones that hint at states like success or failure. This small speaker is chosen to be a piezoelectric disk speaker due to its extremely low profile and low power use.

To address the problem of hardware reconfiguration, buttons and switches are the simplest and most intuitive interfaces. In the case of this RFID node design, a 4-bit DIP switch would be sufficient for providing 16 total user-defined values to give the hardware a unique address. A momentary push button would also provide the mechanism with which to launch the RFID node into access point mode so that it can be configured over WiFi.

In Appendix C, Figure 3 is a block diagram that shows how all the hardware components will be connected together. Since this RFID node will be powered by electricity, a power jack and voltage regulator have been added as well to supply the energy needed to run all parts of the RFID node. The diagram shows the LED and speaker as outputs of the system. The RFID module, DIP switch, button, and voltage regulator are shown as inputs to the system. The system itself is run by the ESP32 microcontroller which is shown in the center of the diagram. This is the piece which will be accepting the inputs and delivering the outputs. Figure 4 of Appendix C shows the physical hardware prototype on a breadboard, but without the voltage regulator. A bill of materials can be found in Appendix A detailing the cost of the parts needed to make one RFID node, but including enough material to build two; the parts were sourced from Ebay.com and as such there may be more parts bought than necessary.

RFID Firmware

Of course, the hardware cannot just automatically know how to work together when there is a microcontroller in the middle, therefore firmware is needed to tell the microcontroller how to coordinate everything. The firmware must be able to get the data from the RFID module, prepare it for transmission to the server, then send it to the server. While this is happening, the firmware must also be able to control the LED and speaker to alert the user as to the current state of the RFID node. Since the RFID node is designed to be configured over WiFi, an access point mode must also exist within the firmware that can be triggered from the normal operation mode.

Figure 5 in Appendix C is a flow chart that shows the basic operation of the RFID firmware. This flow chart does not include details about certain implementation details, or details about error handling modes. Initially, the mode of the RFID node doesn't matter until after it boots up. On boot, the firmware gets the state of the DIP switch to get the user-defined 4 bit number in hexadecimal. Then the on-chip unique identifier gets read and has its least significant nibble replaced with the 4 bit DIP switch number. Then the firmware causes the speaker to make a generic beep to indicate that it successfully booted up.

After this point, the firmware looks into persistent memory to see which mode it should be in, either normal RFID scanning mode or access point configuration mode. If the mode is scanning mode, then the firmware reads the WiFi SSID, WiFi password, address of the time tracking server, and server's port number from the persistent memory and stores them into string variables. Then the firmware connects to the specified WiFi network. If connection is unsuccessful, then the firmware will give a failure beep and go into a loop where the LED blinks SOS in Morse code and tries to connect to the WiFi over and over (this an error mode not in Figure 5).

Once a WiFi connection is established, the firmware will attempt to contact the time tracking server to get the current time. If the server cannot be reached, the firmware will make a failure beep and use a Unix timestamp of 0, it will periodically attempt to contact the server to get the current time (this is an error mode not in Figure 5). Then the firmware will create a queue to store up to 256 RFID scan records in the event that they could not be communicated to the server immediately (not in Figure 5). At this point there will be another general beep to alert the user that the RFID node is ready to start scanning RFID cards.

The firmware then enters a loop where it first checks to see if the access point configuration button has been pressed. If it has been pressed, then the firmware will set the mode record in persistent memory to be access point mode, then it will give a general beep 3 times to indicate that it is preparing to go into access point mode. Then it reboots the node so that it may go through the boot process and get to the point where it checks which mode to be in. More on the access point mode will be discussed after the general operation mode is discussed.

After checking the access point button, the firmware asks the RFID module if there is a card waiting to be scanned. If so, then the LED is turned off to show that the RFID card is being read, a scan beep is made, a RFID scan record is made containing the time the card was scanned along with the serial number of the card, the record is sent to the server, then the LED is turned back on. If the server replies that the scan was invalid, then an error beep is made. If the RFID scan record could not be sent to the server, it will be stored in the queue and an error beep will be made (this is an error mode not in Figure 5).

The firmware then continues the loop by checking to see if it has been an hour since the time was last updated or if the time was not updated during boot. If so, then the time is requested from the server and updated (not in Figure 5). Following this, the firmware checks the queue to see if there are any unsent RFID records and attempts to send them if possible (this is an error mode not in Figure 5). The loop then starts from the beginning after this point.

As stated above, the access point mode can be discussed after the general operation mode. After the first general boot beep, when the firmware is checking the mode to see which mode to start, if the mode is access point configuration mode, then the firmware runs the other half of the code. The first thing it does in access point mode is to generate an SSID with the form of "Arecs_Node_X" where X is the uppercase hexadecimal value of the DIP switch nibble (not in Figure 5). Then it generates a password with the form "passwordX" where X is the same uppercase hexadecimal value (not in Figure 5). With this SSID the node becomes a WiFi access point that broadcasts the SSID which can be connected to with the password.

After starting the access point, the firmware then sets up the web interface and all the API calls that are needed to support the web interface and apply configuration options. The web interface, which can be seen as Figure 6 in Appendix C, shows the modified unique identifier, inputs for the WiFi network to connect to, and inputs for the RFID server to connect to. The API calls that the web interface has access to can be seen in Figure 7 of Appendix C. The web interface uses a JavaScript AJAX call to "/getssids" to get a list of WiFi networks that the RFID node can connect to, this populates the WiFi network selector in Figure 6.

Once the configuration web interface has been filled out, clicking "Save Configuration" (Figure 6) will send the data as a POST request to the "/configure" API call. This will save the WiFi options if both password fields have been filled out. If the passwords do not match, then it will redirect to the web interface for correction (not shown in Figure 7). The "/configure" API will also save the RFID server parameters if the address of the server is filled in. Once finished, it will set the mode to be scanning mode and then return a web page to prompt a node restart in order to apply the changes. Clicking restart will call the "/restart" API call and restart the server so it may start up in scanning mode.

After defining the web interface and API calls, the firmware then starts up a DNS server. This is so that navigating to any website while connected to the access point will direct you to the configuration web interface page, this technique is known as a captive portal. The firmware will then cause a specific access point beep and blink the LED slowly to let the user know that the access point is up and ready to be connected to. The firmware continues this until the user submits the configuration and restarts the node.

RFID Enclosure

After the hardware and firmware had been designed, it was obvious that it would need to be put into an enclosure to protect it from the users and to give it a professional appearance. A tool-less approach to the enclosure was preferred as this would allow for ease of maintenance and configuration of the DIP switch without requiring tools or force to open the enclosure. With this tool-less requirement, a snap-fit mechanism seemed obvious for holding the enclosure together and making it easy to open without tools. It works by having a base with flexible beams with something like a hook at the end that gets caught in indentations in the lid (refer to Figure 8 of Appendix C). They can then be opened by pushing the beams inward to release the hooks from the indentations.

After a bit of design and testing of snap-fit mechanisms, a final snap-fit test piece was designed and made. The outline of the design of this test piece can be seen as Figure 8 of Appendix C, and the piece made from the design can be found as Figure 9. Unfortunately, this piece broke during a demonstration and was no longer considered a valid design, especially when combined with the failure of previous snap-fit test pieces. This was due to the fact that the plastic parts were printed from PLA plastic, which is a strong but brittle plastic that could not flex enough for snap-fit joints without breaking. The breakage would normally occur at the base of the snap joint's arm (Figure 10 is an example of this).

A new design was needed to replace the snap-fit concept, but that would still provide the tool-less features of the snap-fit mechanism. This new design was inspired by the combination of the snap-fit mechanism and a latch mechanism. The final version of this design can be found as Figure 11 of Appendix C. This new mechanism to hold the enclosure together is referred to as a pin-lock or pin-latch mechanism, but the true name of this mechanism is unknown since mechanisms similar to it could not be found. This mechanism also went through a few design and test stages before leading to this final version. Figure 12 highlights some of the changes in a later design to eliminate the breakage seen in the older version of the pin-latch, those structures were added to support and disperse the high stresses experienced in those areas. Other changes were made as well, but were not as significant.

The pin-latch mechanism works by having one side of the lid (light gray on the left of Figure 11) held down by something like a reinforced snap-fit joint that doesn't actually flex (dark gray on the left of Figure 11). The lid is able to hinge on and off of this reinforced snap when the pin (black on the right of Figure 11) is not present. When the pin is present, it prevents the right side of the lid from lifting off of the base (dark gray of Figure 11), and therefore it cannot hinge open and disengage from the snap on the left. The pin uses a snap-fit mechanism to hold itself in place, but this is not a load-bearing snap-fit joint with a high degree of flex required. An image of a physical test piece using this design can be found as Figure 13 of Appendix C.

With the mechanism to hold the enclosure together figured out, the actual enclosure of the RFID hardware could finally be designed. The layout was designed by arranging the physical hardware components on a piece of paper until a compact and efficient layout was found (Figure 14 of Appendix C). The final design for this enclosure can be found as Figure 15 of Appendix C. The basic design consisted of a base where the electronics would be mounted on (green on Figure 15), a lid to cover everything up (gray on Figure 15), a riser to hold the RFID module close to the surface of the lid (blue on Figure 15), and the pins of the pin-latch mechanism (yellow on Figure 15).

The area of the base (green) was estimated based on how much space the hardware components took up from the layout in Figure 14 of Appendix C. The base also has holes on it for mounting to a wall, these were copied from measurements taken of the mounting holes of a household smoke detector. There is a hole roughly in the middle for the access point configuration button. There are also the reinforced snaps and the second set of holes for the pins that came from the test design. Finally, the base has a large protrusion between the two pin holes for mounting the power jack.

The lid (gray) has the two holes for the pins, along with a bigger hole in the middle for the power jack to stick out of. There is a hole to the side of the pin holes used for connecting a debug cable to the ESP32 microcontroller in the event that long term debugging or re-flashing of the firmware is required. The back of the lid has two slots for each of the snaps to hook into.

The pins (yellow) of the enclosure are the exact same design as used by the test piece. The riser (blue) for the RFID module is just a slanted flat surface (to give the crystal oscillator some space under the lid) with legs that have a little extra material at the bottom for gluing to the base.

Figure 16 of Appendix C shows the RFID node with the assembled electronics within. Everything was secured using hot-melt glue and all connections were soldered. As can be seen, there was extra space in the enclosure due to the fact that the components were able to be packed tighter than anticipated because of the three-dimensional arrangement. Figure 17 shows the completed RFID node powered on with the blue status light in the bottom right side and the two power indicators showing through the somewhat translucent lid of the enclosure.

Database

Since the hardware is not designed to be a standalone product, a server is needed to process and handle the data generated by the RFID nodes. Since that data is also useless unless it can be accessed and manipulated, a front end to the server is required as well. Even before the server, the data generated by the users needs to be stored somewhere, so a database is needed. As mentioned in a previous chapter, the MySQL database was chosen due to its reliability and ubiquity.

Figure 18 of Appendix C shows the different tables that were used to store and organize the data generated by the users. The table *userdb* is used to store information about the users such as their name, email, password for the system, RFID serial number, and other minor information about them. The table *projdb* is used to store the available projects that can be worked on, specifically the title of the project, a description of it, and whether or not it is currently active (can actually be worked on or is there for historical purposes). The table *scannerdb* holds the

RFID nodes that have been registered and are allowed to interact with the server; this information is the node's unique identifier, the name of the node, and the location that the node is installed in.

As for tables that store and relate data, there is the *rfiddb* table which holds the events generated by the RFID nodes; this table holds the id of the user who clocked in, the id of the scanner that created this event, along with what day and time the event occurred. The *daydb* table holds the "clock in, clock out" records for the days for each user; it stores the id of the user who clocked in/out, the day that the record describes, when the shift started, when the shift ended, and how many hours the user worked that day. The *workdb* table stores each thing that the user worked on for each day; specifically it stores the id of the user, the id of the record of the day when the work happened, the id of the project that this work relates to, a description of the work done, when the work started, and when the work ended.

To access the data in the database, standard SQL queries were used such as SELECT, INSERT, and UPDATE. All database access is done by the server, along with the generation of any SQL queries. A complete listing of the SQL queries used in the server can be found in Table 1 of Appendix B. If more than one query is used in an API call, then the query will be numbered. Options wrapped in curly braces ({}) and separated by slashes (/) indicate that the query may be generated using any one of those options. Parts of the queries wrapped in square brackets ([]) indicate that the part may or may not be used in the generation of the query depending on what flags were given to generate the query. Question marks (?) indicate where data is inserted into the query before sending it to the database for evaluation.

Server Software

Server software is needed to connect the database to the user interface. As mentioned in a previous section, the Node.js framework is used to run the logic needed to provide that connection. The role of the server is quite simple; the server has to be able to serve web pages to the users, accept API calls and return the relevant data, and interact with the database to fulfill the API calls. Of course, this is easier said than done. Because of this, the use of third party libraries was warranted to ease the development effort. A total of eight libraries were used: express for serving web pages and handling GET/POST requests, body-parser for decoding POST query parameters, client-sessions for maintaining session cookies to store login information, fs for accessing files on the server's file system, mysql for providing a software interface to the MySQL database, crypto for generating random numbers for use in password salts, cron for executing jobs at scheduled times, and multer for handling files uploaded through POST requests.

The server begins to set itself up by configuring the multer library to store files in the *public/userpics/* directory, but only if the files are less than 10MB and end with one of the following extensions: png, jpg, jpeg, gif. Then it specifies the *public* folder as the directory to look for a file to serve up when an unknown GET request is made. Then it uses the *db.key* file to configure a connection to the MySQL database with the mysql library. The *db.key* file is a JSON encoded file that contains the following properties: hostsite (the URL of the MySQL server), database (the name of the database), username (the username to the database account), password (the password of the database account). Afterwards, the server uses the cron library to create a cron job that runs every midnight to clock out any accounts that weren't clocked out for the day. This cron job uses the SQL queries in Table 1 of Appendix B to read all the clock-in records for the day (*cron_job(1)*), read all RFID events (*cron_job(2)*), then see if any clock-out events were generated and update the clock-out records accordingly (*cron_job(3)*). Then the server creates an interrupt handler that waits for the SIGINT signal and uses that to stop all cron jobs, close the database connection, and stop the server. The server then applies the configuration for session cookies that have a 7 day duration whose 7 day limit resets on any activity by a user. Then the server configures a request to the root URL to go to the clock-in page if a user is logged in, or to the login page otherwise.

Then some helper functions are defined for use in the API sections of the code. The first function returns a boolean value if the user making a request is currently logged in or not. The second function is a simple wrapper for the functions in the mysql library needed to send an SQL query to the database and receive a response; this function takes care of error states as well. The third function is for the purpose of logging incidents on the server, such as an unauthorized user trying to call an API function which requires administrator privileges; the incidents are viewed in the user interface.

Then a special lookup table is created for the purpose of letting the account API know when a user's data cache has become outdated and must be refreshed from their entry in the database.

The account API is then set up to be accessed by sending a POST request to the */account* URL with the required parameters embedded in the POST request. It is not possible to call the API or send over parameters using a GET request because it was thought to be less secure to send user information over a GET request because less work is required to intercept the data in a GET request. With the exception of the call to log in, all account API calls require the user to be logged in.

The following are the API calls available in the account API. Table 2 of Appendix B shows the parameters for the API call; all parameters must be send through a POST request. The *login* call uses the *login* query from Table 1 of Appendix B to see if a user with the specified email exists, if so then it hashes the provided password and sees if it matches the hash in the database, if so then it creates a persistent session for the user. The *logout* call redirects the user to the */account/logout.html* page where the persistent session is terminated. The *getuser* call uses the *getuser* query of Table 1 of Appendix B to retrieve data about a user, admins can retrieve data for users other than themselves. The data that is returned is stripped of the hash and salt, along with having the estimated times parsed from JSON. This call also updates the user cache if the user is marked in the special lookup table because the cache is outdated.

The following account API calls serve for modifying user records, refer to Table 2 of Appendix B for more details. The *newuser* call uses the *newuser* SQL query in Table 1 to insert a minimal user record into the users table, afterwards returning the id of the new record so that the fields may be filled out. The *edituser* call updates properties on a user's record (admin users can edit the records of users other than themselves) and marks the user cache to be updated for this user. This call uses the *edituser(1)* SQL query of Table 1 to update a password using the crypto library if that is the field to edit, the *edituser(2)* query is used to update the picture of the user, and the *edituser(3)* query is used to modify any of the other user properties. The *deleteuser* call uses the *deleteuser* SQL query to delete the record of a user, it also updates the user cache for this user so the system will be rendered unusable for a deleted user. The *updateestim* API call edits the daily estimations of a user (admins can edit those of other users) which are stored as the JSON encoded times a user typically starts work on a certain weekday and how many hours the shift usually lasts. The *updateestim(1)* SQL query is used to write the updated JSON data back to the user record, or if an admin is updating this data for another user, then the *updateestim(2)* query is used to read the JSON string before making modifications and sending it back to the database. This function also marks the user's cache to be updated.

After the accounts API is set up, some extra GET request handlers for the accounts are set up. These GET request handlers are specifically to redirect to other accounts pages or actually serve up the files for the requested pages. A GET request to the */account* URL either redirects the user to the user profile page, or to the login page. A request to the /*account/logout* URL terminates the user's session and redirects the user to the login page. A request to the */account/login.html* URL will either serve up the *login.html* file or redirect the user to the */account/user.html* if they're already logged in. A request to */account/user.html* either serves up the *user.html* file or redirects the user back to */account/login.html* if they're not logged in.

After all the setup for the account portion of the server is finished, the server proceeds by setting up a cache for unknown RFID events that arrive to the server. This cache stores the day and time that the event was received along with the id of the RFID card that was scanned and the id of the RFID node that submitted the event.

Then the server proceeds with setting up the general API calls for use by the UI. The server listens for GET or POST requests to the */api* URL and merges any POST parameters with the GET parameters; it is still possible to access the original POST data for API calls that require POST-only parameters, but the original GET data is not preserved and can be overwritten by POST parameters with the same key name. The parameters are then sent on to a function whose job it is to process the general API calls and handle permissions. If a non-admin user tries to access an API call that is admin-only or the admin-specific features of an API call, then that user is reported to the incident log mentioned above. Table 3 of Appendix B shows the available API calls, their parameters, and whether the call or any parameters are for admin usage only.

Only two of the API calls are available without having to be logged into the server, the *rfidevent* call and the *gettime* call. The *rfidevent* call uses the *rfidevent(1)* SQL query of Table 1 in Appendix B to see if the scanner sending the request is registered with the system, if not then it adds a record to the unknown RFID cache. If it is a registered node, then it uses the *rfidevent(2)* query to see if a user is associated with the id of the RFID card, if not then it adds a record to the unknown RFID cache. If the card is registered to a user, then the *rfidevent(3)* SQL query is used to log the RFID event to the database, the server uses *rfidevent(4)* to check if the user has been clocked in for the day, and if not, then the server uses *rfidevent(5)* to clock the user in for the day. The *gettime* API call just returns the UNIX time in seconds for the purpose of synchronizing the clocks on the RFID nodes.

The rest of the API calls require the user to be logged in to use them. The *getclocks* call uses the *getclocks* SQL query to return the clock in/out records for a user with optional filter to select which records to return. This call also has the admin-only feature that allows for returning the clock in/out records of other users. The *addclocks* call uses the *addclocks* SQL query to allow a user to clock in for the day using the web interface instead of scanning their RFID card. The *editclocks* call uses the *editclocks(2)* query to allow a user to change the hours that they worked for a day, or it uses *editclocks(1)* to allow other properties to change while automatically updating the hours that they worked if they change the time they started or ended their shift; this call allows an admin to edit the clock in/out records of other users.

The *getworks* API call uses the *getworks* SQL query to return the work descriptions of a user with optional filters to control which work descriptions are returned; this call allows an admin to view the work descriptions of other users. The *addworks* call uses the *addworks* query to allow a user to enter a work description for the project they're working on. The *editworks* call uses the *editworks* query to allow a user to edit various properties of a specified work description; this call also allows admins to modify the properties of another user's work descriptions.

The *getprojs* API call uses the *getprojs(2)* SQL query to return all the projects that have been registered with the system, or it uses the *getprojs(1)* query to return all the projects without their descriptions; there is an optional filter to only show projects that are still active. The *addprojs* call is an admin-only call that uses the *addprojs* SQL query to register a project with the system. The *editprojs* call is an admin-only call that uses the *editprojs* SQL query to update a property of a project record.

The *getscanners* API call is an admin-only call that gets a list of registered RFID nodes using the *getscanners(2)* SQL query, or *getscanners(1)* query if location text isn't needed. The *addscanners* call is an admin-only call that registers an RFID scanner with the system using the *addscanners* SQL query. The *editscanners* call is an admin-only API call that uses the *editscanners* query to edit a property of a specified RFID node registration record.

The *gethours* API call returns the times that a user worked for a given span of days using the *gethours(2)* SQL query. An admin has the ability to return the hours worked by another user, or by all users for a given span of days using the *gethours(1)* query. The *getemployees* call is an admin-only command that returns the records for all employees with password and RFID data stripped out; this call uses the *getemployees* SQL query.

The *getrfidevents* API call is an admin-only call that returns the RFID event records using optional filters for day spans and users, this call uses the *getrfidevents* query. The *geturegrfidevents* call is an admin-only call that returns the cache of unknown RFID events. The *getincidents* call is an admin-only call that return the log of attempted unauthorized API access showing which API call was called and which user attempted it.

Finally, once the server is set up and ready to begin accepting HTTP GET/POST requests, then a function is called to start the server and have it listen on port 8080.

Web UI Software

In order for the system to be used by the intended users, a user interface is required. As mentioned in the section about the conclusions of the research from the literary review, a client-server architecture seems the best implementation with the client being a web site/page. Also as mentioned in the section about hardware and software choices, the client will be made using the standard HTML/CSS/JS languages with the Ractive library and the AM Charts library to aid in data-driven UI design.

The first part of the UI is the login page, which can be seen in Figure 19 of Appendix C. This is a simple page with the company logo, the email field, the password field, and a login button. If the user enters a wrong password, the text "Password is incorrect" shows up in red above the email field as shown in Figure 20. If the user enters an email that does not correspond to any user, the red text shows up with the words "Email not found" like in Figure 21. If for some reason the database connection fails in the middle of the login process, the red text shows "Login Database Connection Error" as in Figure 22.

When the user accesses the home page, or any other page, an animated loading bar shows up until all required server API calls have returned the requested data, see Figure 23. This "clock in" home page is the default page that the user is redirected to after logging in or accessing the root URL of the server. Along with showing a loading bar while loading data in the background, all

UI pages are designed to report any major errors that they encounter. This is done through the user of the alert function in JavaScript (refer to Figure 24) with more detailed information being printed to the JavaScript console. The company logo, any additional navigation buttons, and the user's icon show up as the top bar for all of the UI pages. Clicking the user's icon will redirect the user to the user's profile page as seen in Figure 36.

Once all data for the home page has loaded, the user is presented with a giant green button to press in order to clock into the system, see Figure 25. If the user presses the "Clock In" button, then they are clocked into the system and the page updates to show the view from Figure 26. If the user clocked in previously that day from an RFID node or by clicking the green button and accesses the home page from another browser tab, then they will be shown the same clocked-in view from Figure 26. The clocked-in view features a giant red button for clocking out with as an alternative to using an RFID node. Below the red button, the interface shows the time that the user was clocked in at, along with estimates for how long the user's shift is planned to last and when their shift might end. Below that are the fields to enter a description of the work they are currently doing with when they started the task, when they ended the task, and what project that task belongs to. The table below that shows all of the work descriptions that they submitted for that shift. Figure 27 shows a work description record having its project property edited through the use of a combo box; all the table fields are editable by clicking on them and entering the new values. Figure 28 shows the initial clock in page as seen by a user without admin privileges; the main difference is that the gray bar at the top doesn't have the additional navigation links.

If an admin user clicks on the additional navigation link called Manage Employees, then they will be taken to the Employee Management page as shown in Figure 29. This page shows all registered users as a gallery of cards containing the user's profile picture, their name (sorted in alphabetical order), their email address, how much money they've made in the specified pay period if they work hourly or what their salary pay is, and how many hours they've worked during the pay period. Clicking on a user's card will take the admin to the View Profile page of that user (see Figure 32). An arbitrary pay period can be entered in the Pay Period field, or it can be left as is to show the previous 14 days. Below they Pay Period field, there is a button to generate a table view (Figure 30) that shows all the employees, all their shifts during that pay period along with the hours they worked for that shift, when it started and ended, and a total of hours for that pay period for that user. The table can be hidden with the Hide View button, or can be downloaded as a CSV file with the Download as CSV button. If the admin clicks the New User button, they will be given a confirmation prompt to continue with the new user creation (Figure 31). Clicking OK will take the admin to the profile page of the new user where all the fields can be updated for the new user, refer to Figure 32.

The View Profile page, as seen in Figure 32, allows an admin to not only see the details of a user, but also their historical data such as previous work days and RFID events generated by that user. This page also allows an admin to edit almost any property of the user, including historical work data as well. This page is used to edit the properties of both new and existing users. Any properties that can be edited will have a dark background and a pencil icon when hovered over, as shown in the top left corner of Figure 32; this editing feature also applies to all other pages with editable fields. If the editable property is clicked, then the appropriate data entry field will pop up so that a new value can be entered and saved using the Save Changes button (Figure 33). Figure 34 shows the table where the estimated times for each weekday for the user can be entered (these values are used in the estimated time section of the clock in page in Figure 26). There is also an RFID scans table that shows all of the RFID events created by the specified user for that day. At the bottom there is a button to set the user's password with the entry for the password coming from a JavaScript prompt popup. There is also a button to delete the user and all their data; this button is followed by a JavaScript confirmation prompt.

From the home page, an admin user can navigate to the Settings page with the navigation link at the top left of Figure 25. The Settings page, as seen in Figure 35, shows a table of all the registered projects and has a button to add another project, all fields except for ID are editable. There is a table displaying all registered RFID nodes with a button to add another node, all fields except for ID are editable. There is a table to show all logged RFID events with a selector for the span of time that the RFID events happened in along with which user caused the event and what node they used, none of these fields are editable. There is a table to show all the restricted API events that were accessed by non-admin users, this table shows who the user was and what API call they tried to access; this table has no editable fields. There is also a table to show all RFID events that either do not belong to a known user, or came from an RFID node that was not registered with the system.

If a user or admin click on their icon in the top right corner of any page, they will be redirected to their profile page as seen in Figure 36. The only difference between the admin's profile page and the user's profile page is the lack of certain navigation links for the non-admin user (see Figure 37). The profile page shows the user's picture, which can be updated by clicking on it and uploading a new image. The user's name, email, current pay, and hours are shown but are not editable with this page. The estimated hours for that day are editable for that user. A chart below shows the shifts for each day for that user along with what work tasks they did. The long light colored bar in the back represents the time that the shift lasted for, and the darker colors represent each individual work description entry. Below the chart there is the view for the user's historical hours, but they cannot edit any of those entries, only view them. Below that is the table showing the estimated times for each day of the week, which is editable by the user (see Figure 37).

Chapter V: Conclusion

Summary and Conclusion

In the end, three smaller projects were completed and joined together in order to make the complete time tracking system possible. The first part was to make the physical hardware and firmware required to scan an employee's RFID card in order to clock them in or out of work. The second part was to make the server software that would receive data from the RFID hardware and store it for further processing, along with providing the necessary two way communication interfaces for the client user interface. The third part was to make the web based client user interface so that both employees and administrators could access the data in a friendly way.

During the implementation of this project, there were very few problems that occurred. One problem that did occur was that the enclosure for the RFID node came out somewhat warped due to a defect in the 3D printer that was used, but this problem was minor because it didn't hinder the construction of the RFID node. Another problem that occurred was one relating to poor communication with the database, this turned out to be caused by the cloud based database service that was used for initial development and was easily fixed by using a database server installed on the development computer.

Aside from those two minor problems, the only really big problem that was encountered was the issue of the design for the locking mechanism of the RFID enclosure. This problem took the longest to figure out how to fix and required the most work to fix.

Overall this project was very successful in that the end product not only worked, but exceeded the expectations for performance and usability.

Discussion

By taking on and completing the task of creating a new employee time tracking system, I have contributed to the environment of the company J2 Innovations by giving the employees a system that is easier and more convenient to use while also giving management a system that allows easier access to employee data. Along with contributing to J2 Innovations, I have also contributed to the open-source community by making all files related to the project open source and publicly accessible on GitHub so that anyone may replicate or improve this system and use it for themselves free of charge.

My study was able to resolve the original problem that the third party time tracking systems had by providing a system which addresses the faults of the other systems without taking away useful functionality. Through this study I was able to conclude that sometimes it is beneficial to use custom software solutions, especially if it means greater convenience.

Implications

The implications of the ideas from this study are quite large, allowing any company to have a convenient employee time tracking system for little more than the cost of the hardware required to run the system. The implications that an open system like this can have have also remained largely the same since the system was proposed, or especially more so since the system was not originally proposed to be open source.

Personal Reflection

In the design and implementation of this project, I have learned many things. As a whole, what I have learned is a valuable lesson in product design. This was the largest project I have ever had to design myself that was actually supposed to be used by people other than myself. As such, I had to take many considerations into account, such as how to design a user interface that is friendly and intuitive for someone who is not particularly tech-savy.

On a more specific level, I learned a lot about web development. This project required that I learned how to effectively use asynchronous web programming to achieve communication with the server and it pushed me to find creative solutions to problems regarding how to work with functions calls that return at random time intervals. I also learned how to create a custom REST API implementation that the UI could use to communicate to the server with.

On the hardware side I learned quite a bit about 3D printing. The basics of 3D printing were already familiar to me, but this project exposed me to designing and 3D printing mechanical components that had to withstand real-world use and abuse. I also learned how to program IoT firmware for a real hardware device with failure modes and feedback to the user.

If I had to redo this project, there are a few changes I would make, namely in the area of product quality. On the hardware side, more time could have been invested into the RFID node enclosure to make the design more compact with real support structures integrated into the enclosure to facilitate a better assembly technique. More time could have also been spent trying to fix the defect of the 3D printer to produce an enclosure without any warping. On the software side, more effort could have been put into designing a stylish web interface. The current interface is acceptable but is not close to what is expected of a modern web interface.

Future Research and Recommendations

If someone in the future would wish to take this project and develop it further, there are some recommendations that I would give to enable them to create a superior version. My largest recommendation would be to design the data model to include multiple time zones, possibly by encoding everything as a UNIX timestamp and using a locale property on the user record to do any time conversions necessary. Another recommendation to create an improved product is to design the interface to allow users to request time off of work, such as sick days or vacation days. A strictly UI change that I would recommend to a future developer would be to include more and varied charts for management to use in assessing employee data.

I would also recommend that any future developers do research on hardware enclosure design to be able to design an enclosure that looks less like a piece of equipment and instead something that looks like it belongs in the room. My primary recommendation for this is to try to design the RFID node to be circular as the rounded edge would stick out less, or at least possibly look nicer, than the current rectangular enclosure design.

I would also recommend that the future developer looks into the best way to create an automated installer program to be able to install and set up the software system with minimal effort from the user. This would allow the system to come into the reach of less technical companies and businesses, such as “mom and pop” shops with a few employees.

References

Computers, software; employees and employers share the love with online time tracking. 2013. *Computer Weekly News,* May 02. http://0-search.proquest.com.leopac.ulv.edu/docview/1341082761?accountid=25355.

Costin, Aaron, et al. "Leveraging Passive RFID Technology for Construction Resource Field Mobility and Status Monitoring in a High-Rise Renovation Project." *Automation in Construction*, vol. 24, July 2012, pp. 1–15., doi:10.1016/j.autcon.2012.02.015.

Feibus, Andy. 1999. Automated time-trackers. *InformationWeek* no. 742: 53-56, http://0-search.proquest.com.leopac.ulv.edu/docview/229141683?accountid=25355.

Gruber, Elizabeth M. 2001. Project time tracking. *Modern Machine Shop* 74, no. 1: 203, http://0-search.proquest.com.leopac.ulv.edu/docview/213704139?accountid=25355.

Najera, Pablo, et al. "Real-Time Location and Inpatient Care Systems Based on Passive RFID." *Journal of Network and Computer Applications*, vol. 34, no. 3, May 2011, pp. 980–989., doi:10.1016/j.jnca.2010.04.011.

Replicon releases cloud clock for tracking employee time. 2011. *Wireless News*, http://0-search.proquest.com.leopac.ulv.edu/docview/904432535?accountid=25355.

Syspro rolls out real-time employee tracking solution. 2007. *Wireless News*: 1, http://0-search.proquest.com.leopac.ulv.edu/docview/210111631?accountid=25355.

Appendices

Appendix A: Bill of Materials

Material	Quantity	Price	Unit Price
5.5/2.1 mm DC Barrel Jack	5	$1.49*	$0.298
3 A DC Step-Down Buck Converter	10	$5.95	$0.595
ESP32 IoT Board	2	$17.99	$8.995
MFRC-522 RFID Reader w/ RFID Card and RFID Fob	2	$9.98	$4.99
27 mm Piezo Speaker	2	$4.59	$2.295
3 mm Diffused LED	10	$1.89	$0.189
Tact Switch	10	$4.99*	$0.499
7 Bit DIP Switch	5	$4.25*	$0.85
5.5/2.1 mm 9 VDC Wall Adapter	5	$30.00*	$6.00
Completed RFID Node Unit Price	1	*N/A*	$24.711**

*These parts were on hand during the development of the RFID node and therefore did not have to be purchased for the project. These prices are estimates.

**This price includes estimated prices along with excluding the prices of plastic for the RFID case, solder, and wire for the electrical assembly.

Appendix B: Tables

API Call	SQL Query
cron_job(1)	SELECT did,uid,tstart FROM daydb WHERE day=? AND tend IS NULL
cron_job(2)	SELECT uid,time FROM rfiddb WHERE day=?
cron_job(3)	UPDATE daydb SET tend=?, hours=TIME_TO_SEC(TIMEDIFF(tend,tstart))/3600 WHERE did=?
login	SELECT * from userdb WHERE email=?
getuser	SELECT * FROM userdb WHERE uid=?
updateestim(1)	UPDATE userdb SET days=? WHERE uid=?
updateestim(2)	SELECT days FROM userdb WHERE uid=?
edituser(1)	UPDATE userdb SET hashpw=?, salt=? WHERE uid=?
edituser(2)	UPDATE userdb SET picture=? WHERE uid=?
edituser(3)	UPDATE userdb SET {email/fname/lname/admin/wage/picture/rfid/passwd}=? WHERE uid=?
newuser	INSERT INTO userdb (email,salt,hashpw,fname,lname) VALUES ('email@email.com','0000','00000000','First','Last')
deleteuser	DELETE FROM userdb WHERE uid=?
rfidevent(1)	SELECT sid from scannerdb WHERE devid=?
rfidevent(2)	SELECT uid from userdb WHERE rfid=?
rfidevent(3)	INSERT INTO rfiddb (uid, sid, day, time) VALUES (?,?,?,?)
rfidevent(4)	SELECT did FROM daydb WHERE uid=? AND day=? AND tend IS NULL
rfidevent(5)	INSERT INTO daydb (uid, day, tstart) VALUES (?,?,?)
getclocks	SELECT * FROM daydb WHERE uid=? [AND day=?] [AND WEEK(day)==WEEK(?)] [AND day>=?] [AND day<=?] [AND tend IS NULL]
addclocks	INSERT INTO daydb (uid, day, tstart) VALUES (?,?,?)
editclocks(1)	UPDATE daydb SET {day/tstart/tend}=?, hours=TIME_TO_SEC(TIMEDIFF(tend,tstart))/3600 WHERE did=? AND uid=?
editclocks(2)	UPDATE daydb SET hours=? WHERE did=? AND uid=?
getworks	SELECT * FROM workdb WHERE uid=? [AND did=?] [AND did>=?] [AND did<=?]
addworks	INSERT INTO workdb (uid, did, pid, description, tstart, tend) VALUES (?,?,?,?,?,?)

editworks	UPDATE workdb SET {tstart/tend/pid/description}=? WHERE wid=? AND uid=?
getprojs(1)	SELECT pid,title,active FROM projdb [WHERE active=1]
getprojs(2)	SELECT * from projdb [WHERE active=1]
addprojs	INSERT INTO projdb (title, description, active) VALUES (?,?,?)
editprojs	UPDATE projdb SET {title/description/active}=? WHERE pid=?
gethours(1)	SELECT uid,did,day,hours,tstart,tend FROM daydb WHERE 1 [AND day>=?] [AND day<=?] [AND uid=?]
gethours(2)	SELECT did,day,hours,tstart,tend FROM daydb WHERE uid=? [AND day>=?] [AND day<=?]
getemployees	SELECT uid,email,fname,lname,wage,days,picture FROM userdb
getrfidevents	SELECT * FROM rfiddb WHERE 1 [AND uid=?] [AND day=?] [AND day>=?] [AND day<=?]
getscanners(1)	SELECT sid,name FROM scannerdb
getscanners(2)	SELECT * FROM scannerdb
addscanners	INSERT INTO scannerdb (devid, name, location) VALUES (?,?,?)
editscanners	UPDATE scannerdb SET {devid/name/location}=? WHERE sid=?

Table 1: List of SQL queries used by the server software, refer to Database section Chapter IV

API Call	Parameters	Description
login		Logs the user in with a persistent session, no special permissions
	email	The email to log in with
	passw	The password to log in with
logout		Logs the user out of a persistent session, no special permissions
getuser		Returns a user's session data, user accessible, some admin only features
	uid	Optional, selects a different user, admin only feature
newuser		Creates a new account and returns the account id, admin only
edituser		Edits a user's account, user accessible, some admin only features
	uid	Optional, selects a different user, admin only feature
	tag	Which property to update (email, fname, lname, admin, wage, picture, rfid, passwd)
	value	The new value to assign to the property
deleteuser		Deletes a user's account, admin only

	uid	The id of the user to delete
updateestim		Updates a user's estimated times table, user accessible, some admin only features
	uid	Optional, selects a different user, admin only feature
	tag	Which property to update (start, hours)
	value	The new value to assign to the property
	day	Which day's estimate to update

Table 2: List of account API calls and parameters, refer to Server Software section of Chapter IV

API Call	Parameters	Description
rfidevent		Receives and processes RFID events, no authentication
	time	POST only, UNIX timestamp (seconds) of the event
	id	POST only, the id of the RFID card that was scanned
	node	POST only, the id of the RFID node that did the scan
gettime		Returns the UNIX timestamp (seconds) of the server, used for maintaining time accuracy on RFID nodes
getclocks		Returns clock in/out records of a user, some admin only features
	uid	Optional, selects a different user, admin only feature
	date	Optional, filter the records by a certain date, overrules week
	week	Optional, filter the records by a week, overrules fromday
	fromday	Optional, return records starting from this date
	today	Optional, return records ending by this date
	notjustnull	Optional, also return records that are clocked out
addclocks		Adds a clock in record for a user
	date	The date of the shift to clock in for
	time	The time that the user clocked in at
editclocks		Edits a clock in/out record for a user, some admin only features
	uid	Optional, selects a different user, admin only feature
	tag	Which property to edit (day, tstart, tend)
	value	The new value to assign to the property
	did	The id of the clock in/out record to edit
getworks		Returns work description records of a user, some admin only

		features
	uid	Optional, selects a different user, admin only
	did	Optional, the id of the clock in/out record that these belong to
	fromdid	Optional, the id of the beginning clock record that these belong to
	todid	Optional, the id of the ending clock record that these belong to
addworks		Adds a work description record
	did	The id of the clock record that this record will belong to
	pid	The id of the project that this description is for
	desc	The description text of the work being done
	tstart	The starting time of the work
	tend	The ending time of the work
editworks		Edits a work description record, some admin only features
	uid	Optional, selects a different user, admin only
	tag	Which property to edit (tstart, tend, pid, description)
	value	The new value to assign to the property
	wid	The id of the work record to edit
getprojs		Return the available project categories
	nodesc	Optional, return the projects without descriptions
	active	Optional, return only projects that are marked as active
addprojs		Adds a project category record, strictly admin only
	title	The title of the project
	desc	A description of the project
	active	Optional, a boolean setting the active status of the project
editprojs		Edits a project category record, strictly admin only
	tag	Which property to edit (title, description, active)
	value	The new value to assign to the property
	pid	The id of the project record to edit
gethours		Returns the amount of hours worked by one or all user for a given time span, some admin only features
	allemployees	Optional, return hours for all users, admin only
	uid	Optional, return hours for a specific user, admin only
	fromday	The start date to get hours for

	today	The end date to get hours for
getemployees		Return all user records, strictly admin only
getrfidevents		Returns RFID event records, strictly admin only
	uid	Optional, return events generated by a specific user
	day	Optional, filter events by a date, overrules fromday and today
	fromday	Optional, starting date to get RFID events of
	today	Optional, ending date to get RFID events of
geturegrfidevents		Returns all RFID events that came from unknown RFID cards or unknown RFID nodes, strictly admin only
getscanners		Return all RFID node records, strictly admin only
	noloc	Optional, do not return records with the location property
addscanners		Register an RFID node with the system, strictly admin only
	devid	The hardware id of the RFID node
	name	The name of the RFID node
	location	The location that the RFID node is installed at
editscanners		Edits the record of an RFID node, strictly admin only
	tag	Which property to edit (devid, name, location)
	value	The new value to assign to the property
	sid	The id of the RFID node record to edit
getincidents		Return all incidents where a non-admin user tried to access admin only API features, strictly admin only

Table 3: List of general API calls and parameters, refer to Server Software section of Chapter IV

Appendix C: Diagrams

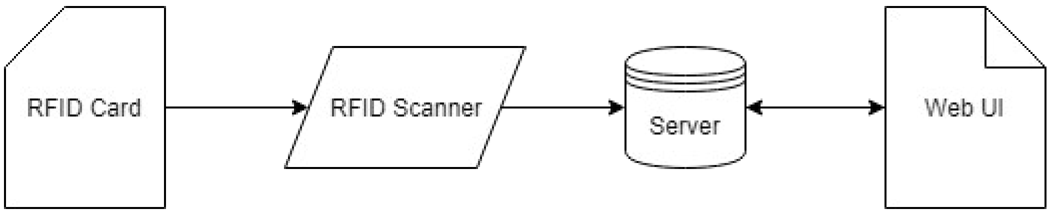


Figure 1: Basic system architecture

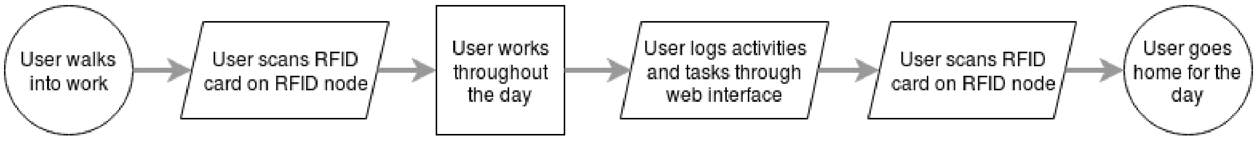

User walks into work
User scans RFID card on RFID node
User works throughout the day
User logs activities and tasks through web interface
User scans RFID card on RFID node
User goes home for the day

Figure 2: Non-management employee workflow

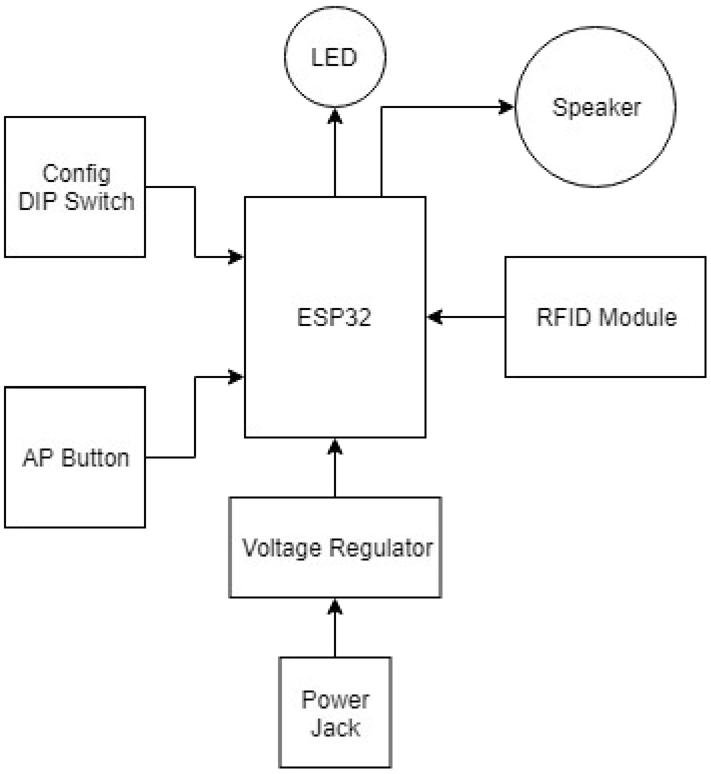


Figure 3: RFID hardware connection block diagram

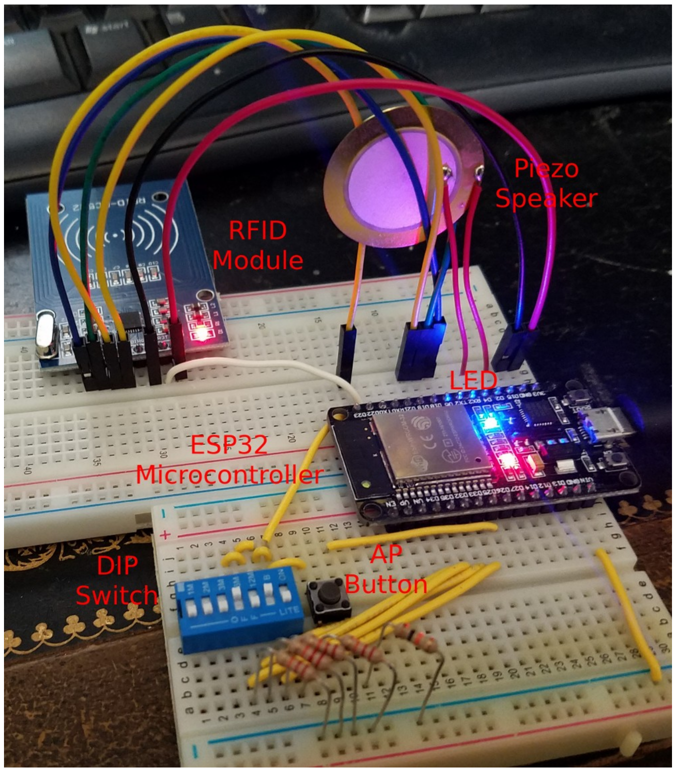


Figure 4: Physical prototype on breadboard

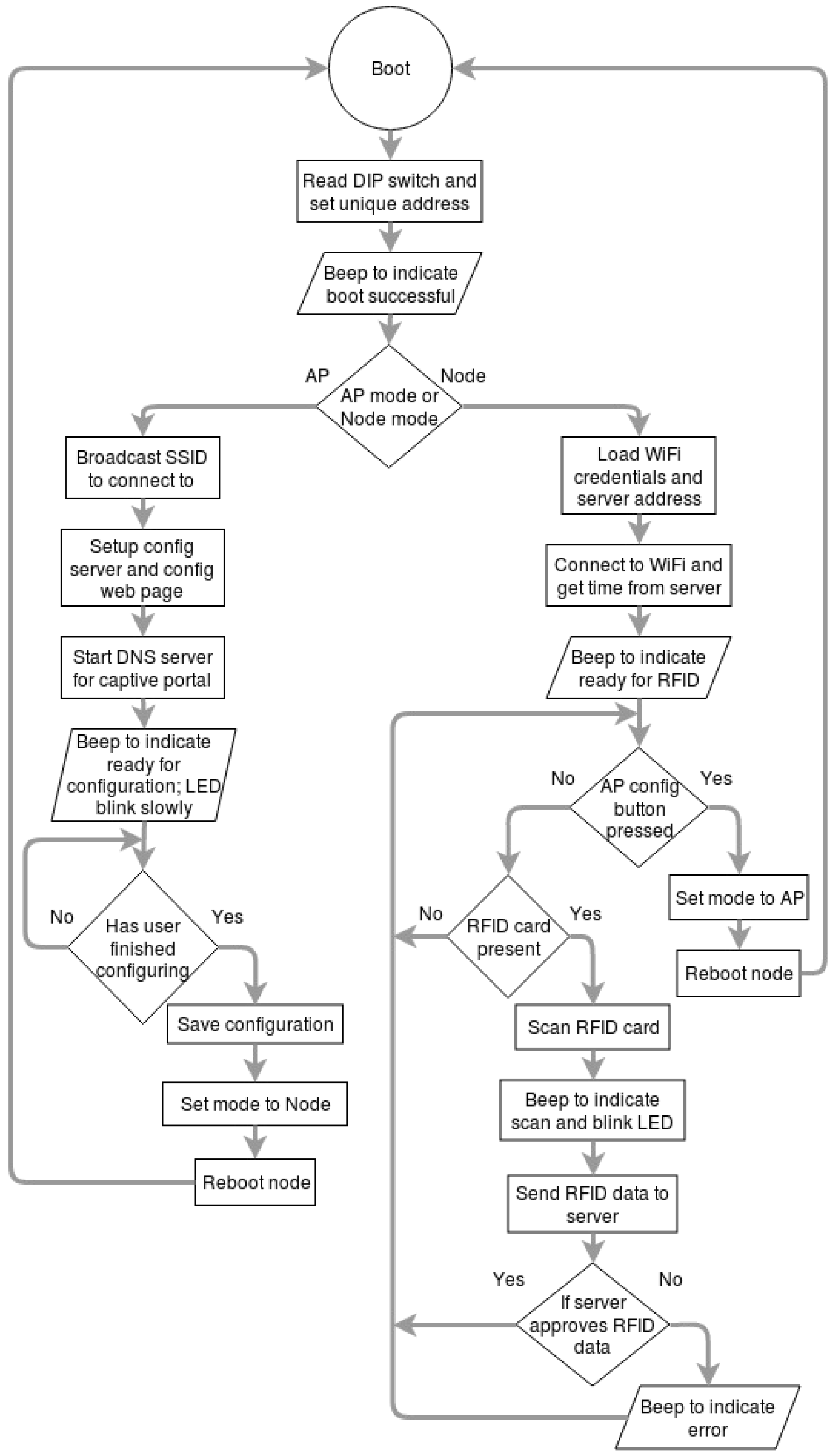
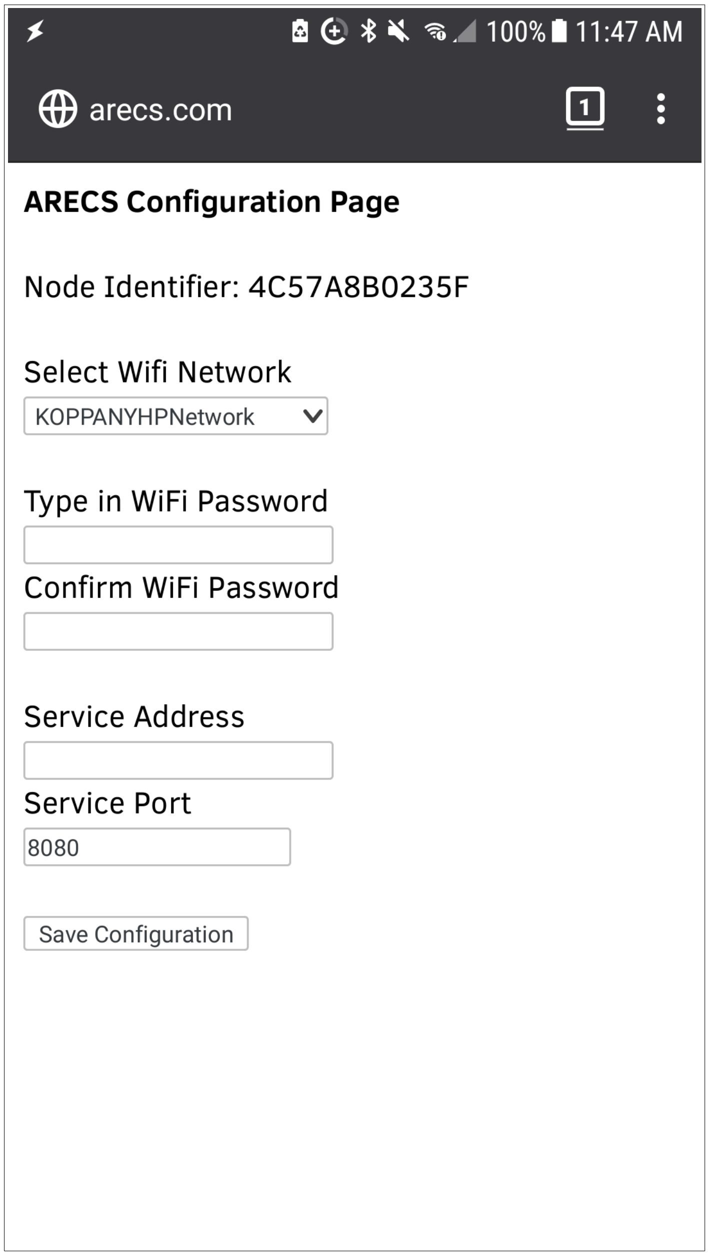

Boot
Read DIP switch and set unique address
Beep to indicate boot successful
AP
AP mode or Node mode
Node
Broadcast SSID to connect to
Setup config server and config web page
Start DNS server for captive portal
Beep to indicate ready for configuration; LED blink slowly
No
Has user finished configuring
Yes
Save configuration
Set mode to Node
Reboot node
Load WiFi credentials and server address
Connect to WiFi and get time from server
Beep to indicate ready for RFID
No
AP config button pressed
Yes
Set mode to AP
Reboot node
No
RFID card present
Yes
Scan RFID card
Beep to indicate scan and blink LED
Send RFID data to server
Yes
If server approves RFID data
No
Beep to indicate error

Figure 5: Firmware flowchart for RFID node


100% 11:47 AM

arecs.com

ARECS Configuration Page

Node Identifier: 4C57A8B0235F

Select Wifi Network

KOPPANYHPNetwork

Type in WiFi Password

Confirm WiFi Password

Service Address

Service Port

8080

Save Configuration

Figure 6: AP configuration web interface

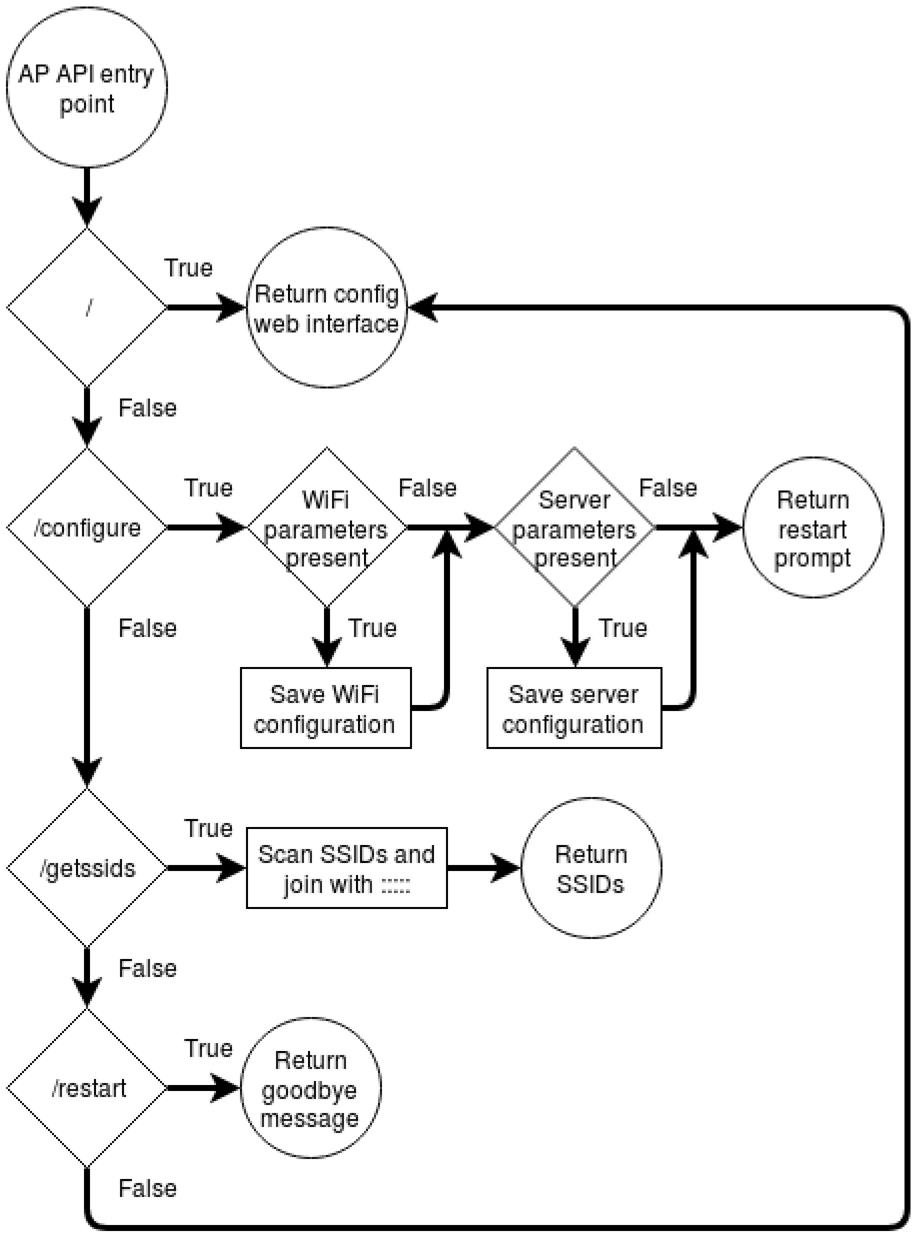


Figure 7: API flowchart for RFID node AP firmware

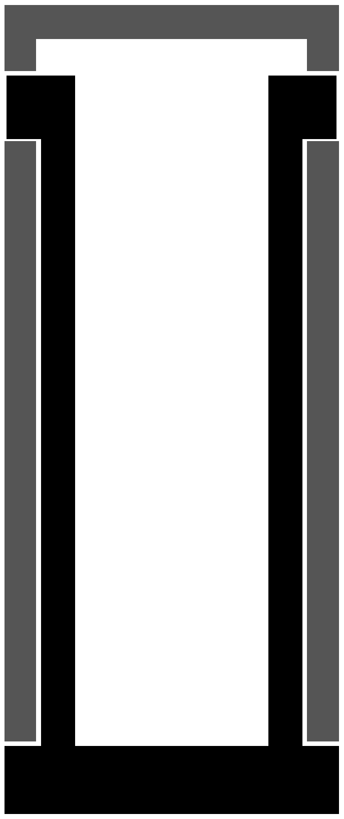


Figure 8: Design of snap-fit test piece

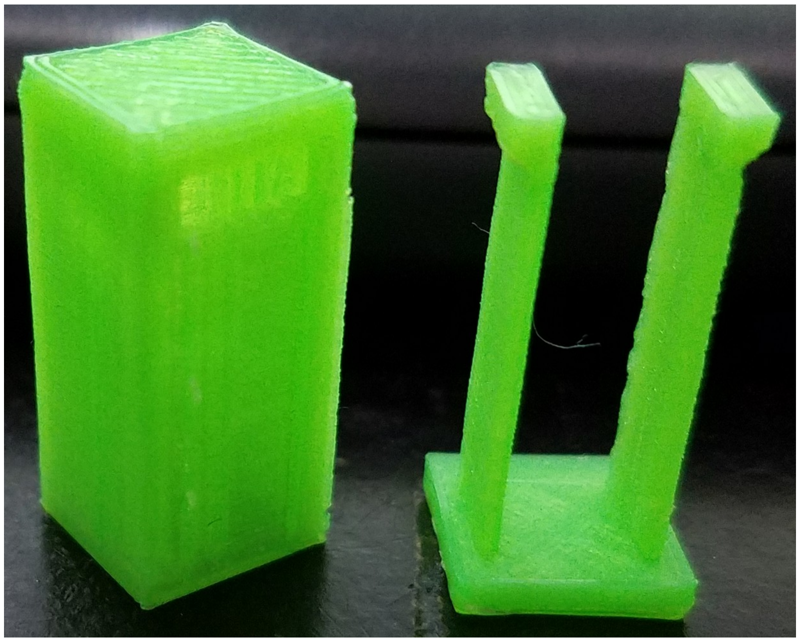


Figure 9: Physical snap-fit test piece

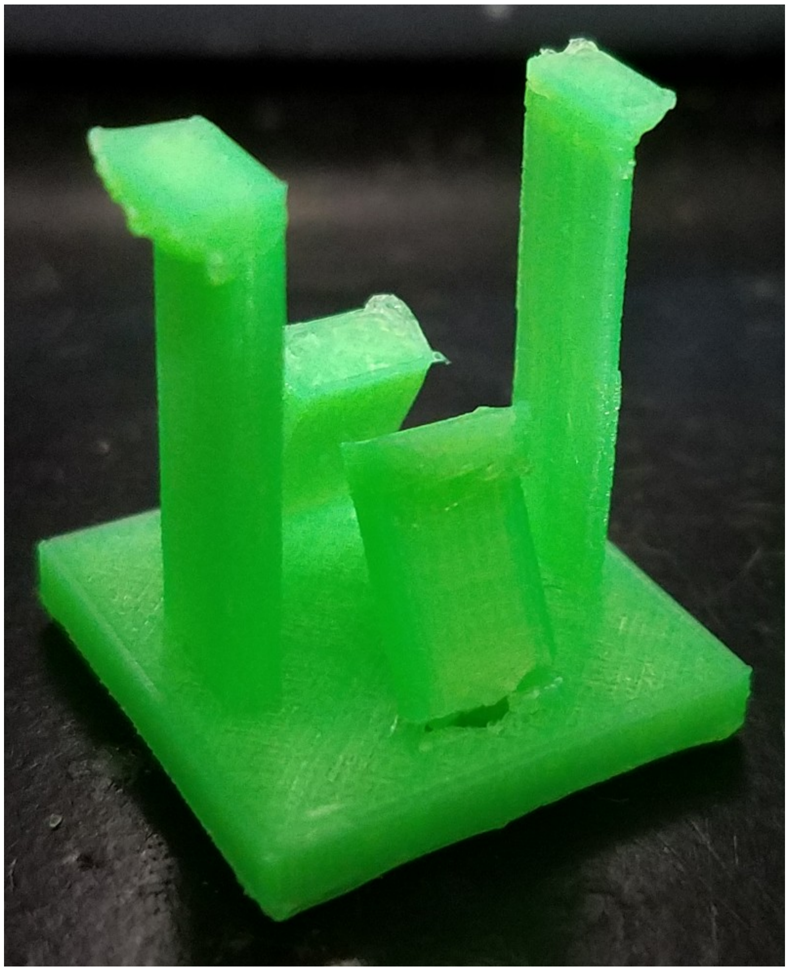


Figure 10: Example of broken snap joints

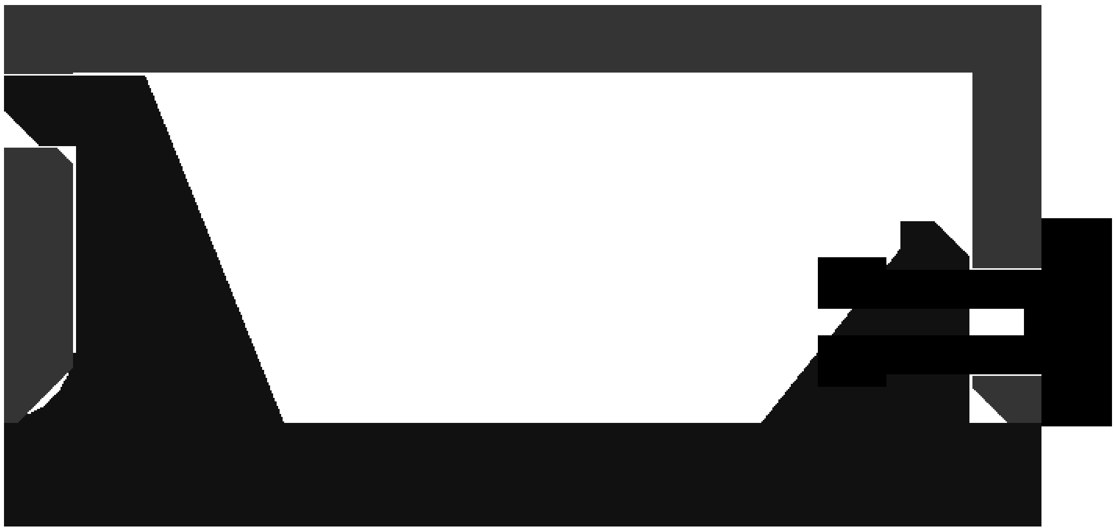


Figure 11: Design of pin-latch test piece

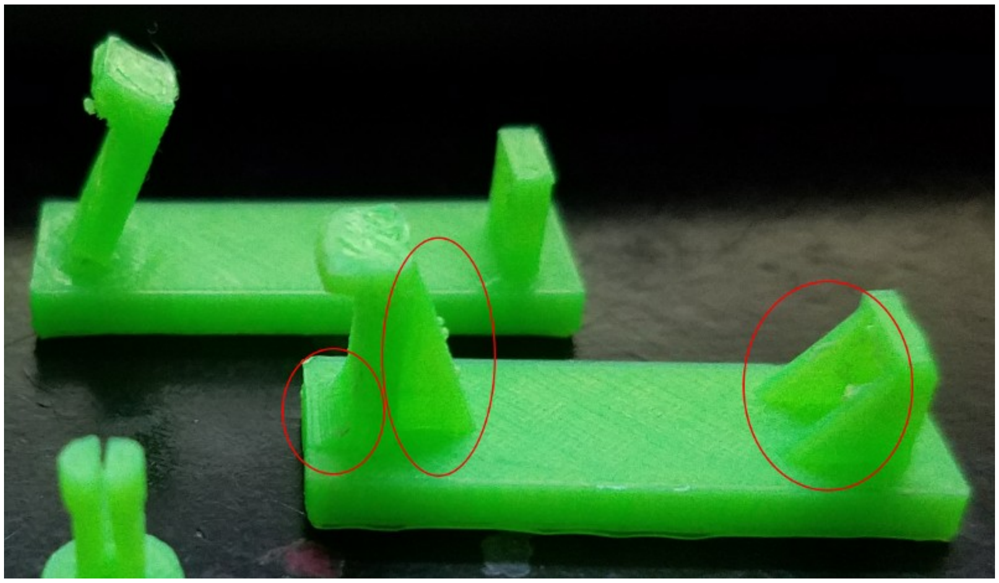


Figure 12: Newer version of pin-latch with supporting structures in red

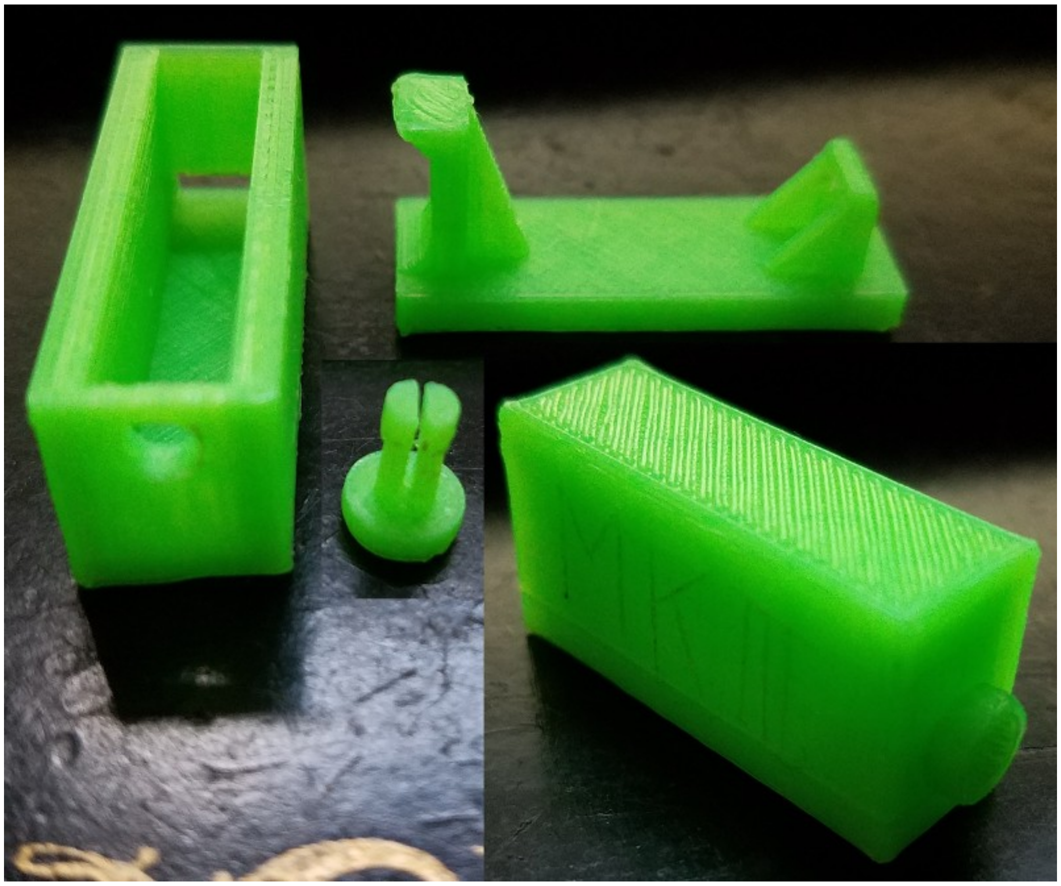


Figure 13: Physical pin-latch test piece

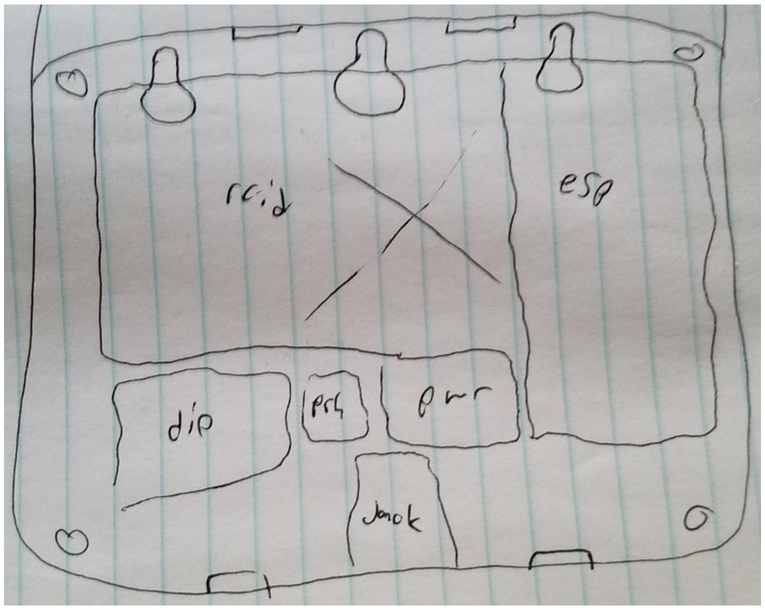


Figure 14: Hardware layout designed on paper

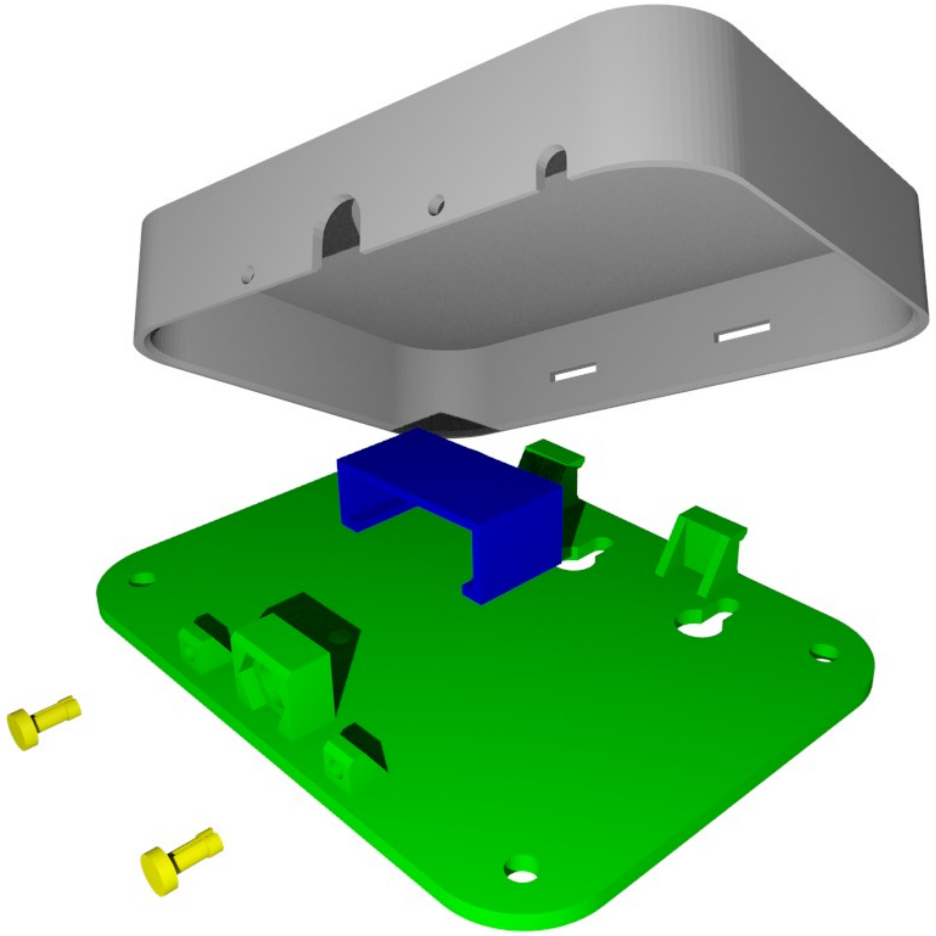


Figure 15: Enclosure design using pin-latch mechanism

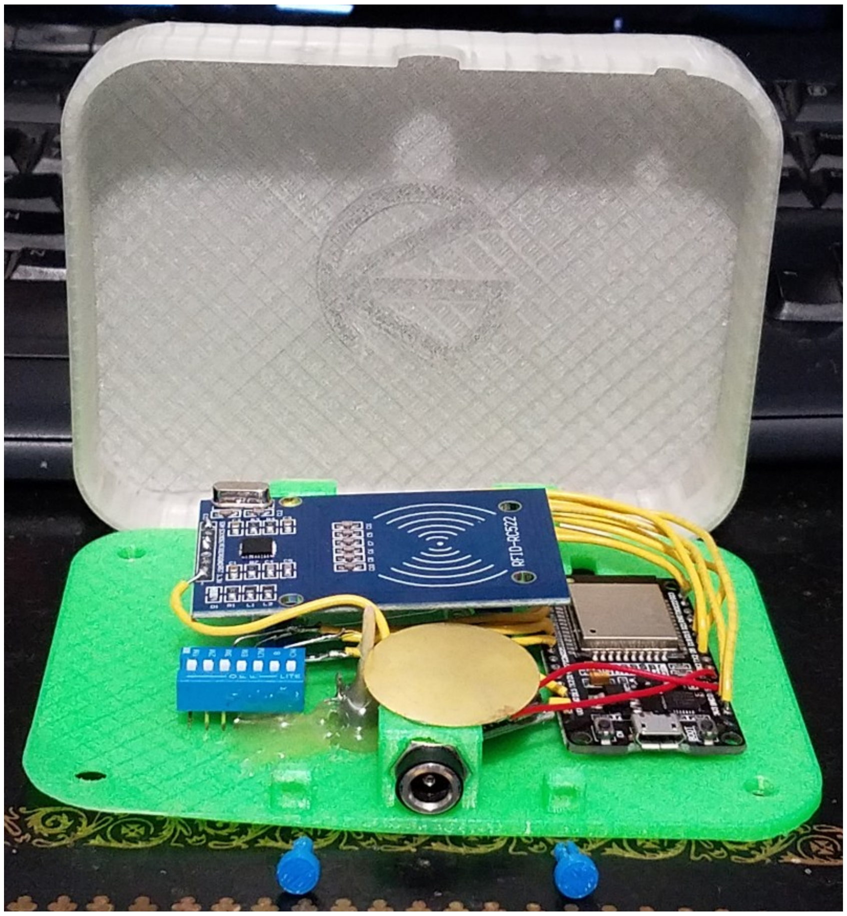


Figure 16: Enclosure with assembled electronics

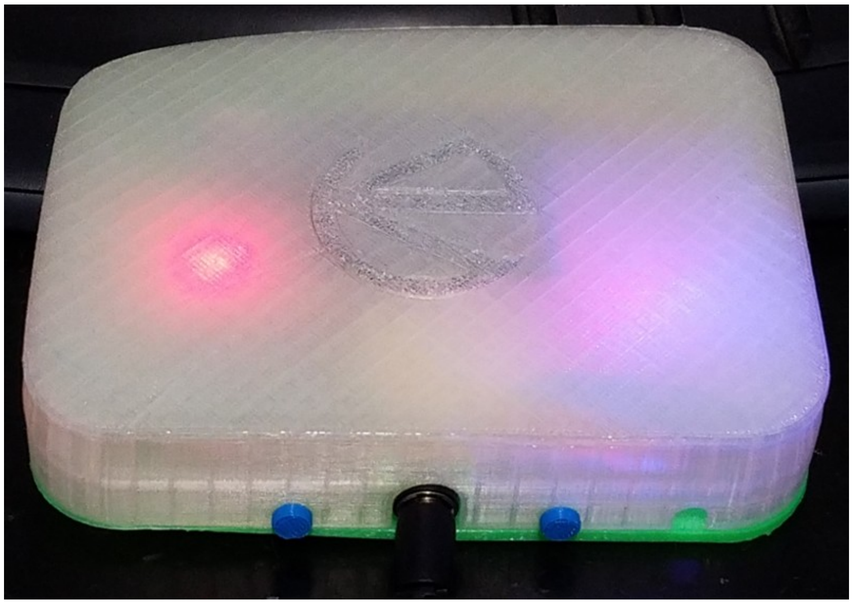


Figure 17: Assembled RFID node powered on with lights

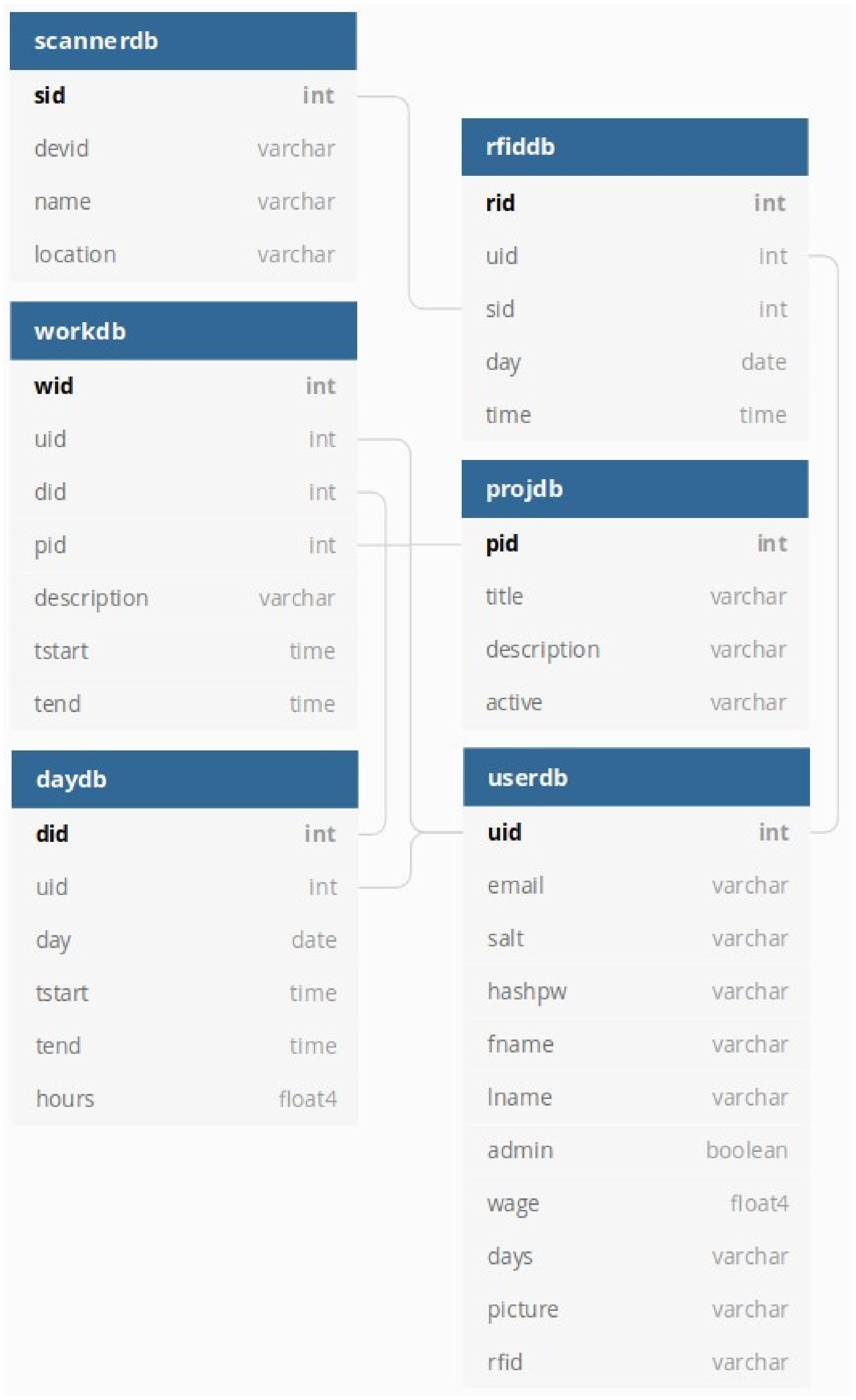


Figure 18: Database tables and relations

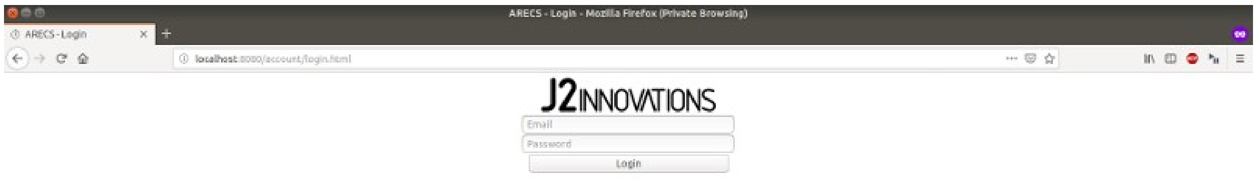
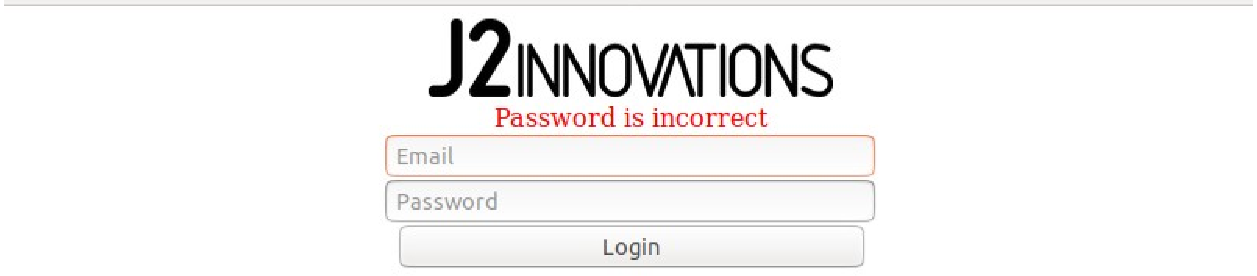
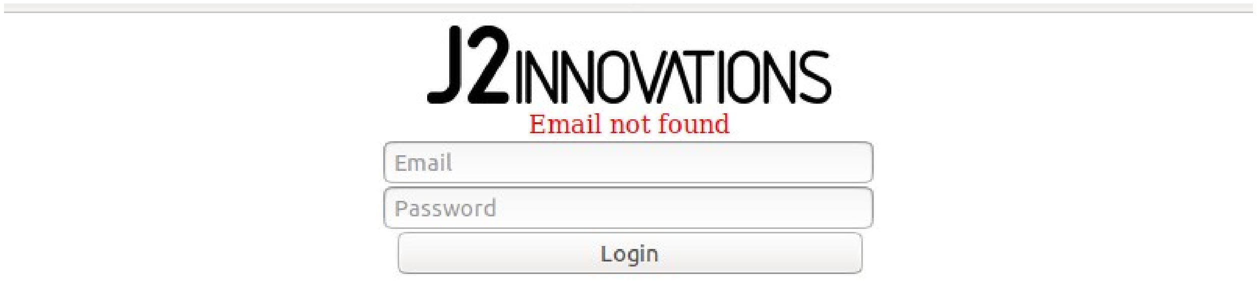

ARECS - Login - Mozilla Firefox (Private Browsing)
ARECS - Login
localhost:8080/account/login.html
J2INNOVATIONS
Email
Password
Login

Figure 19: UI login page



Password is incorrect

Figure 20: Login prompt showing that the password is wrong



Email not found

Figure 21: Login prompt showing that the email does not exist

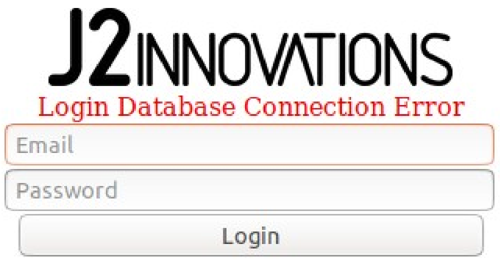


Figure 22: Login prompt showing that the database is inaccessible

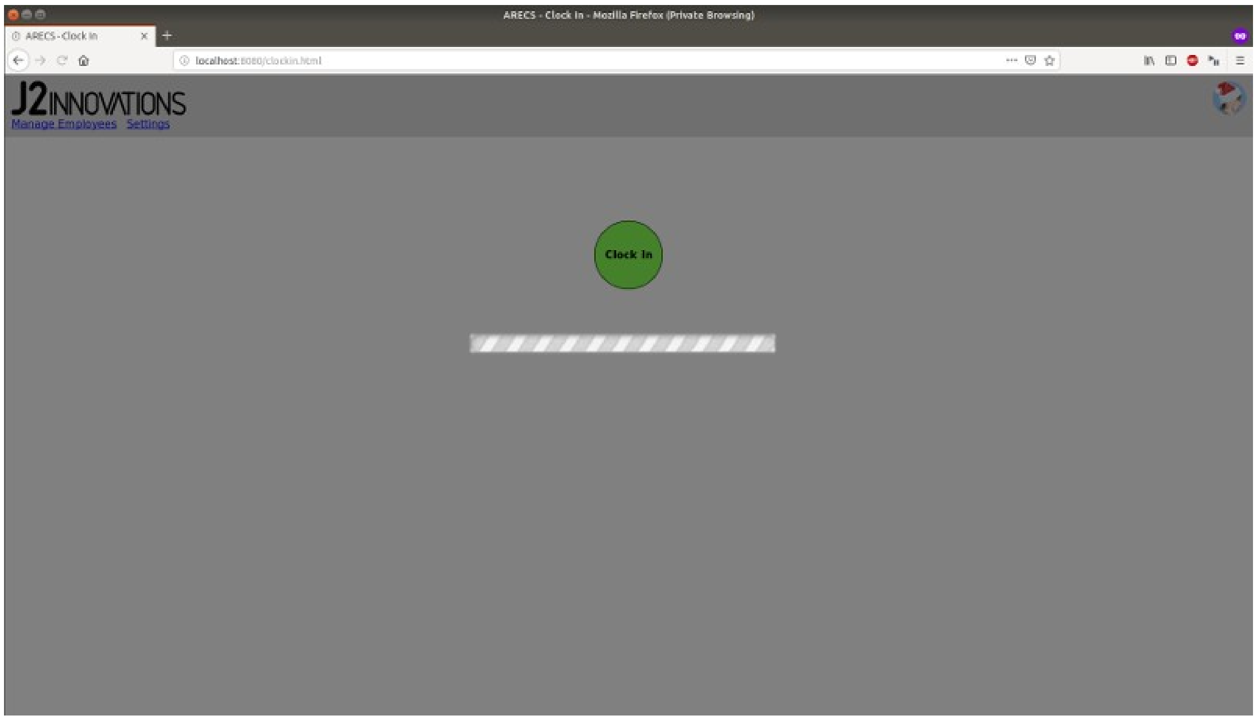

ARECS - Clock In - Mozilla Firefox (Private Browsing)
ARECS - Clock In
localhost:8080/clockin.html
Manage Employees
Settings
Clock In

Figure 23: Loading bar on home page

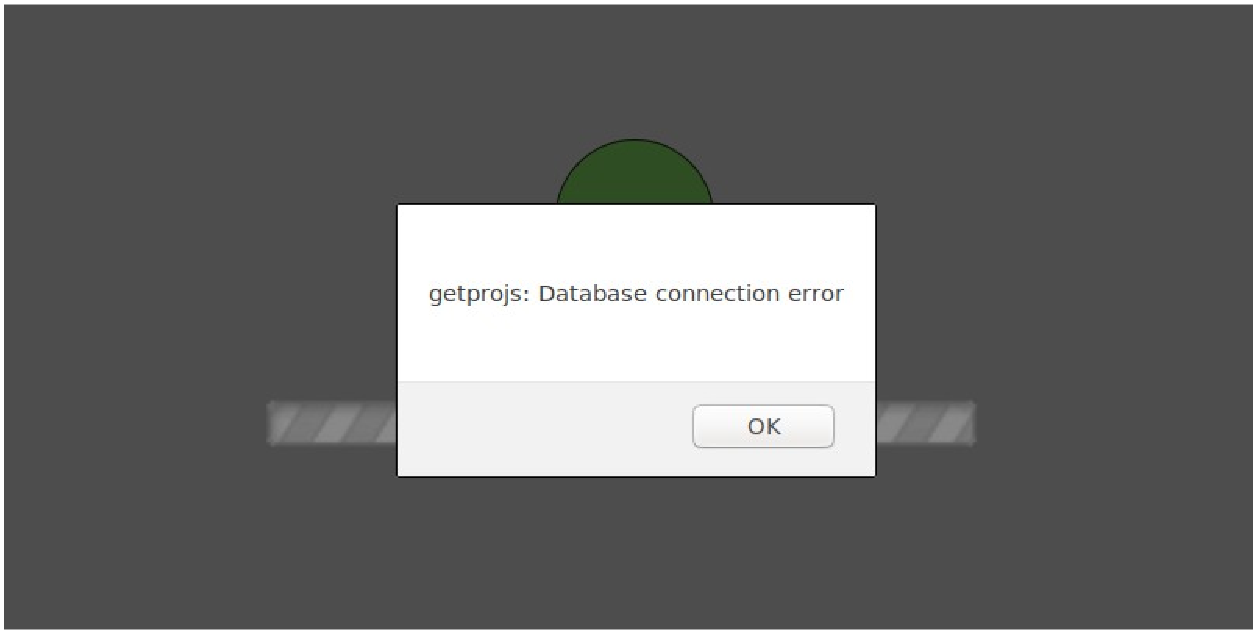

getprojs: Database connection error

Figure 24: UI alerting the user that an API call has failed

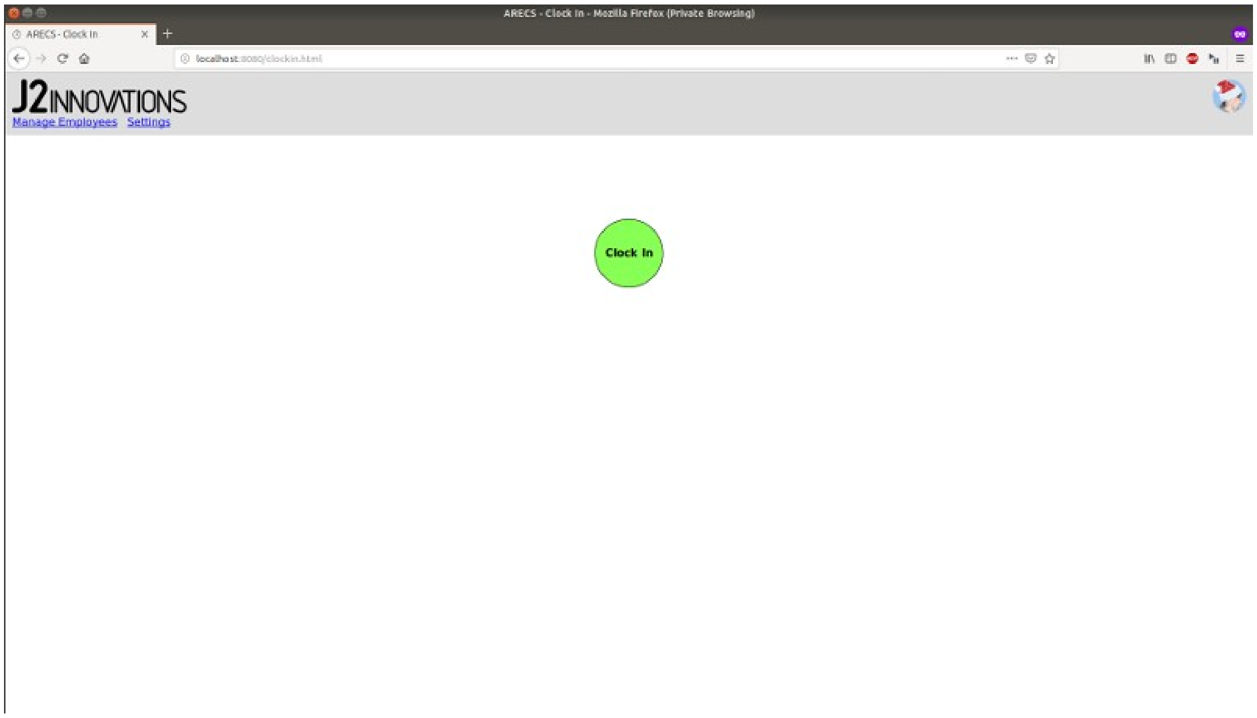
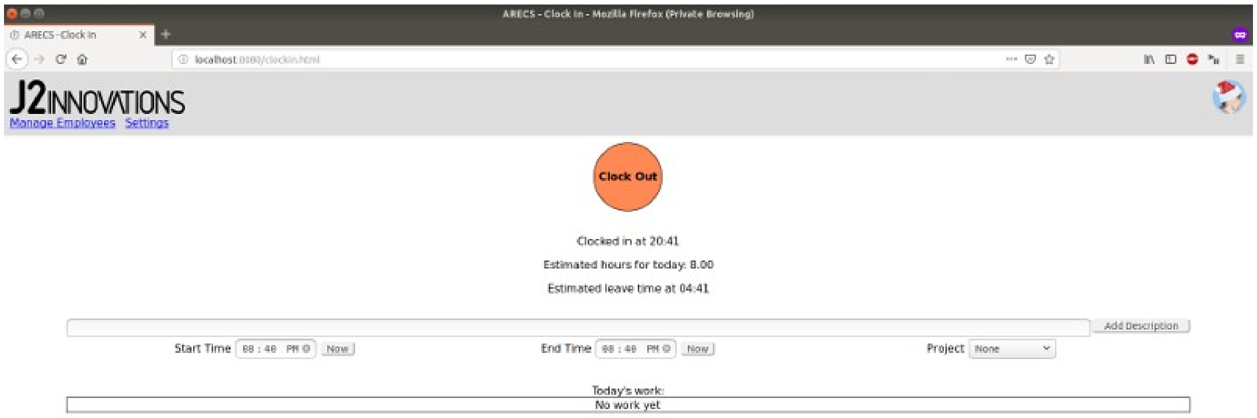

ARECS - Clock In - Mozilla Firefox (Private Browsing)
ARECS - Clock In
localhost:8080/clockin.html
J2INNOVATIONS
Manage Employees
Settings
Clock In

Figure 25: Button to clock into the system


ARECS - Clock In - Mozilla Firefox (Private Browsing)
localhost:8080/clockin.html
J2INNOVATIONS
Manage Employees
Settings
Clock Out
Add Description
Now
Now
None

Clocked in at 20:41

Estimated hours for today: 8.00

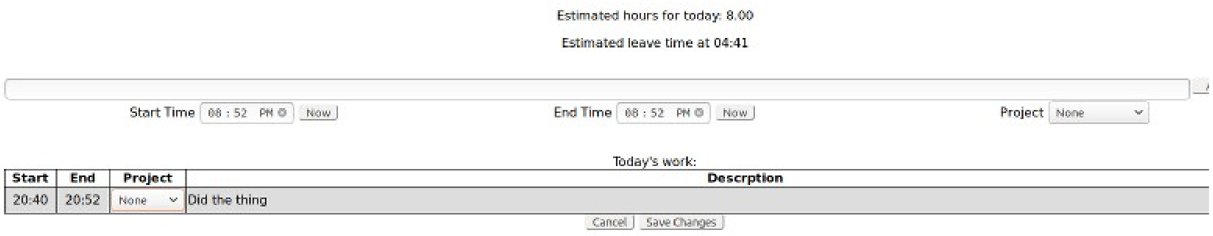
Estimated leave time at 04:41

Start Time 08 : 40 PM | End Time 08 : 40 PM | Project None

Today's work:

No work yet

Figure 26: Home page when clocked in showing estimates and work entry fields



Estimated hours for today: 8.00

Estimated leave time at 04:41

Start Time 08 : 52 PM | End Time 08 : 52 PM | Project None

Today's work:

Start	End	Project	Descrption
20:40	20:52	None	Did the thing

Figure 27: Work description list with project field being edited

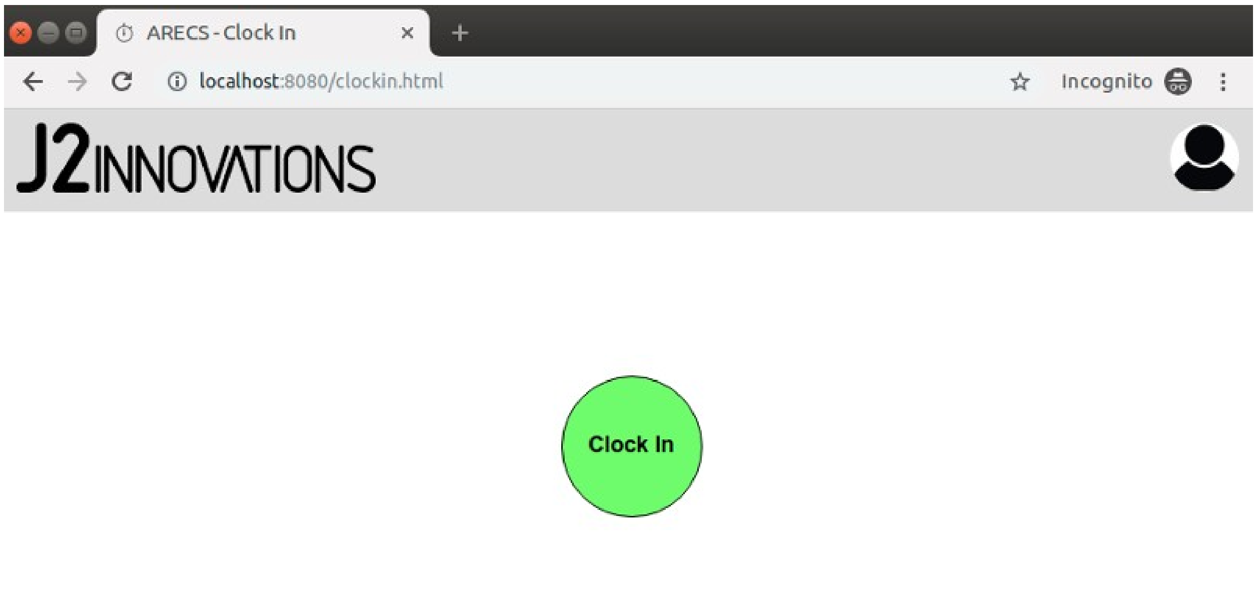


Figure 28: The clock in page from a non-admin account

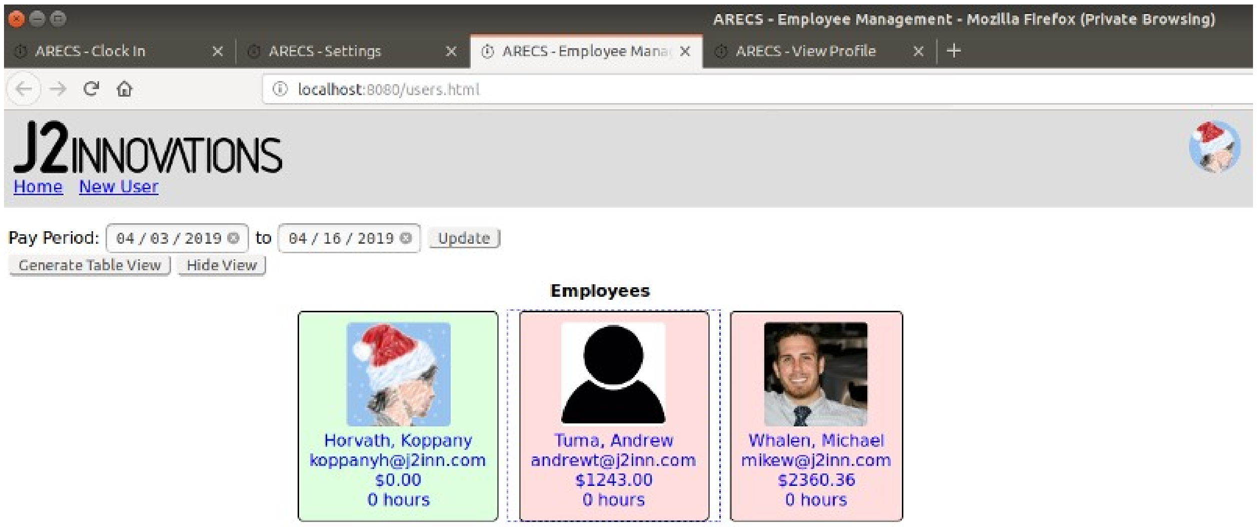
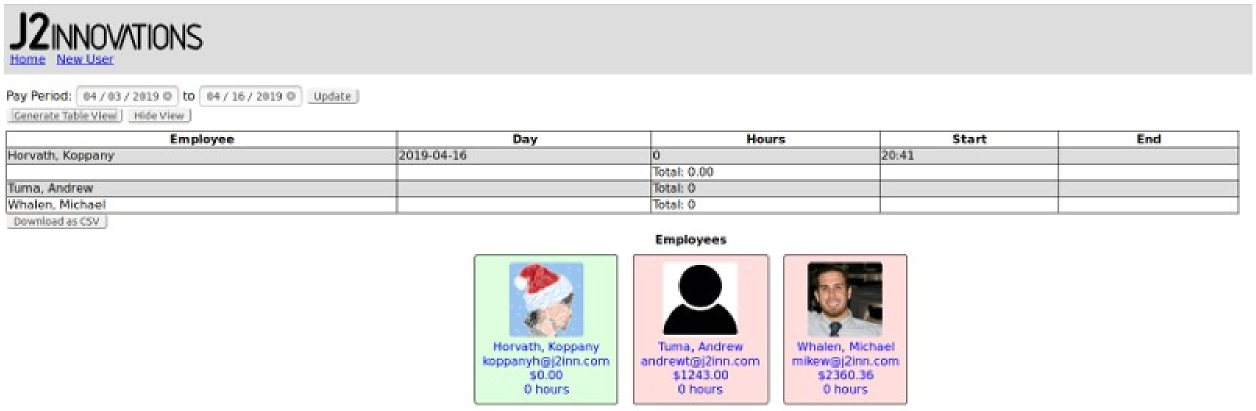


Figure 29: Manage Employee page showing employee statuses



Employee	Day	Hours	Start	End
Horvath, Koppany	2019-04-16	0	20:41	
		Total: 0.00		
Tuma, Andrew		Total: 0		
Whalen, Michael		Total: 0		

Figure 30: Shift report table

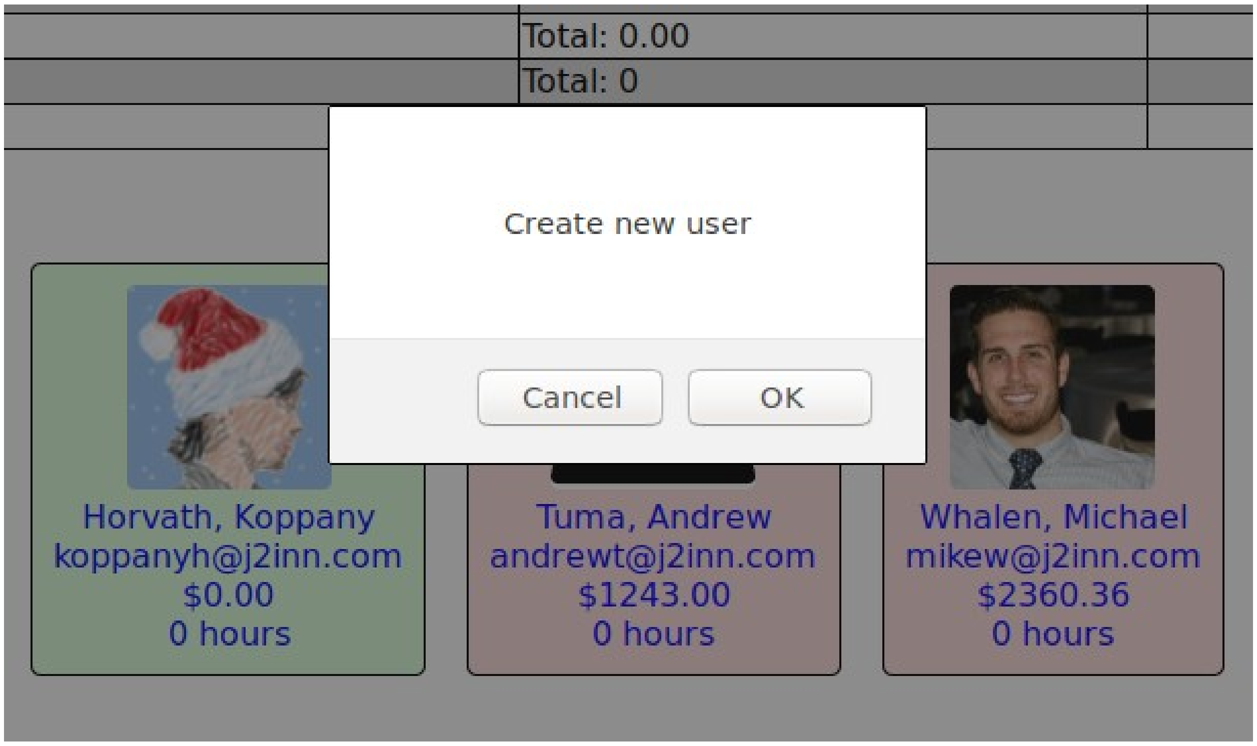
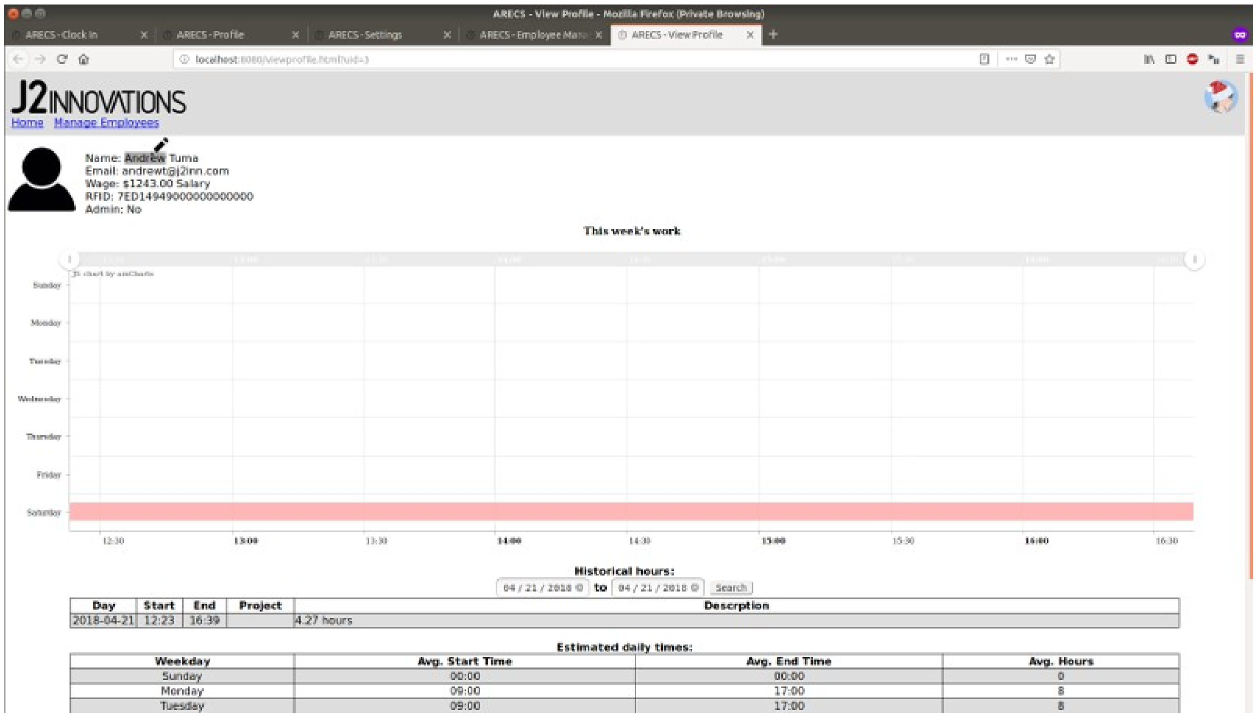


Figure 31: Confirmation to create a new user



Name: Andrew Tuma
Email: andrewt@j2inn.com
Wage: $1243.00 Salary
RFID: 7ED149490000000000000
Admin: No

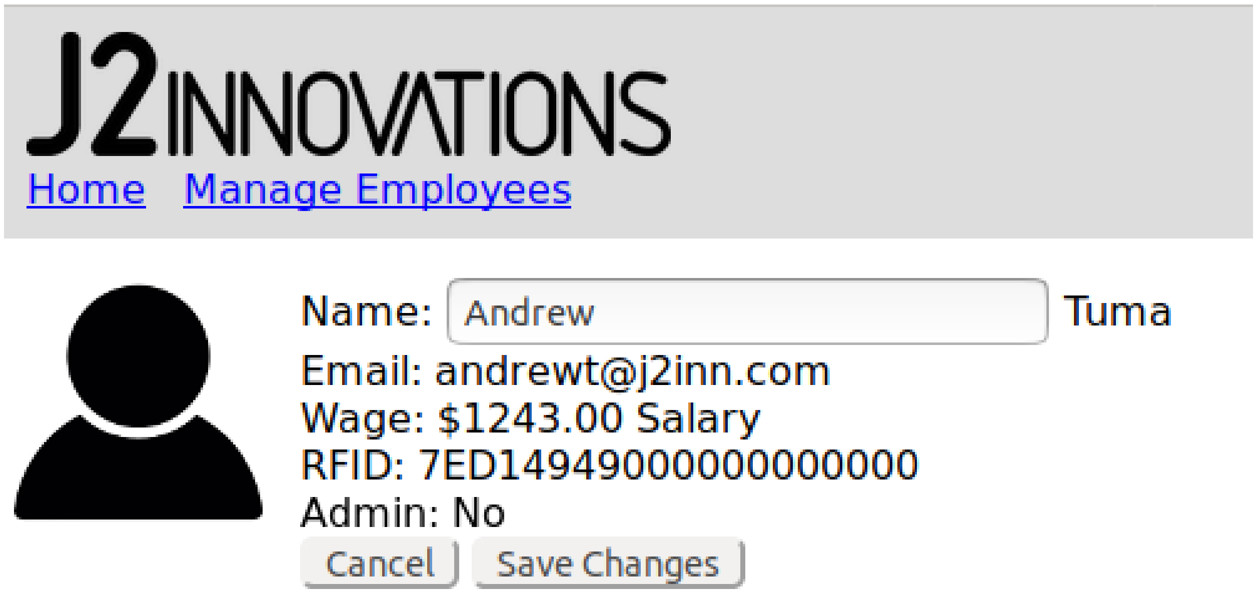
Historical hours:
04 / 21 / 2018 to 04 / 21 / 2018

Day	Start	End	Project	Descrption
2018-04-21	12:23	16:39		4.27 hours

Estimated daily times:

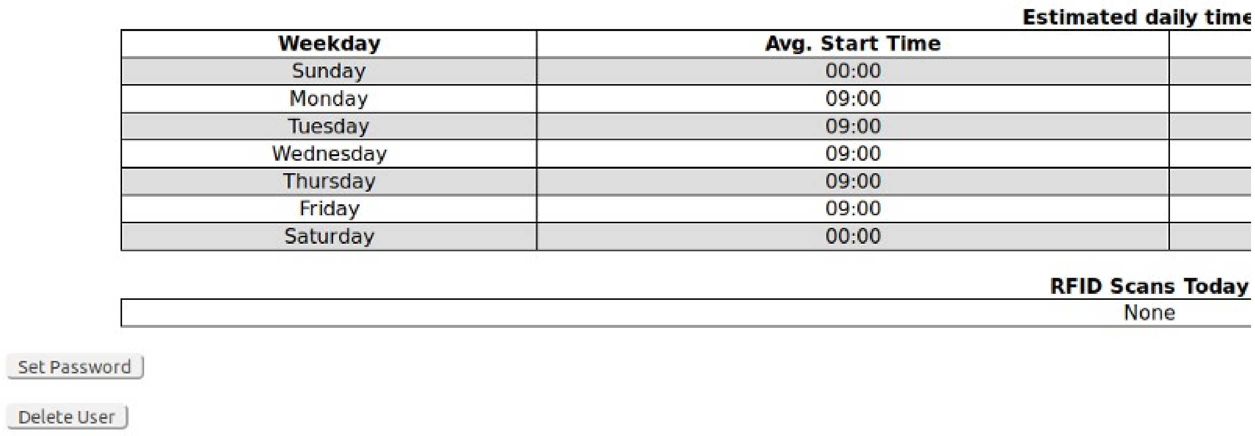
Weekday	Avg. Start Time	Avg. End Time	Avg. Hours
Sunday	00:00	00:00	0
Monday	09:00	17:00	8
Tuesday	09:00	17:00	8

Figure 32: The View Profile page as seen by an admin



Name: Andrew Tuma
Email: andrewt@j2inn.com
Wage: $1243.00 Salary
RFID: 7ED1494900000000000
Admin: No

Figure 33: Editing a user's properties

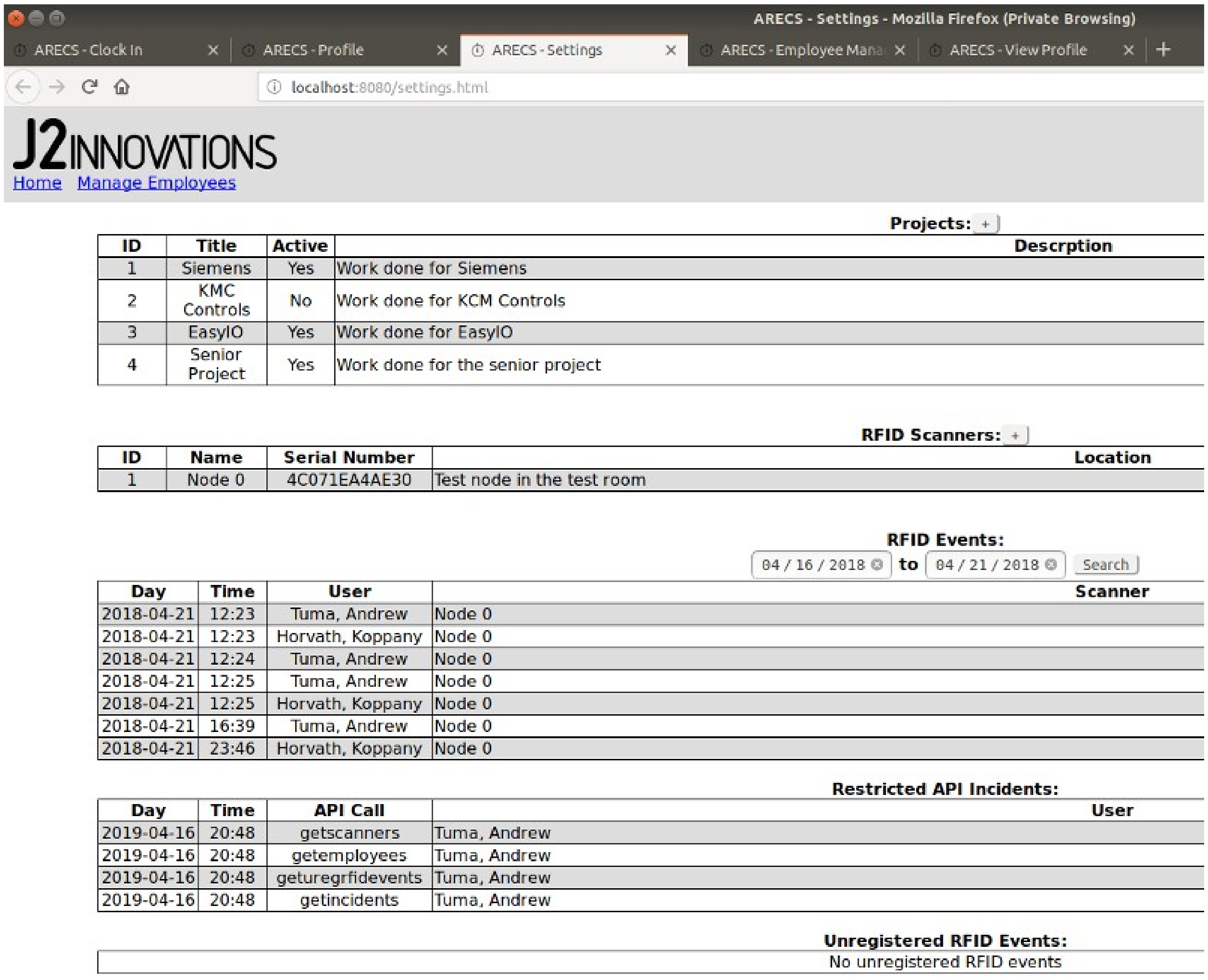


Estimated daily time

Weekday	Avg. Start Time
Sunday	00:00
Monday	09:00
Tuesday	09:00
Wednesday	09:00
Thursday	09:00
Friday	09:00
Saturday	00:00

RFID Scans Today
None

Figure 34: Estimated times table, RFID scans table, password button and delete button of the View Profile page



Projects: +

ID	Title	Active	Descrption
1	Siemens	Yes	Work done for Siemens
2	KMC Controls	No	Work done for KCM Controls
3	EasyIO	Yes	Work done for EasyIO
4	Senior Project	Yes	Work done for the senior project

RFID Scanners: +

ID	Name	Serial Number	Location
1	Node 0	4C071EA4AE30	Test node in the test room

RFID Events:

04 / 16 / 2018 to 04 / 21 / 2018 Search

Day	Time	User	Scanner
2018-04-21	12:23	Tuma, Andrew	Node 0
2018-04-21	12:23	Horvath, Koppany	Node 0
2018-04-21	12:24	Tuma, Andrew	Node 0
2018-04-21	12:25	Tuma, Andrew	Node 0
2018-04-21	12:25	Horvath, Koppany	Node 0
2018-04-21	16:39	Tuma, Andrew	Node 0
2018-04-21	23:46	Horvath, Koppany	Node 0

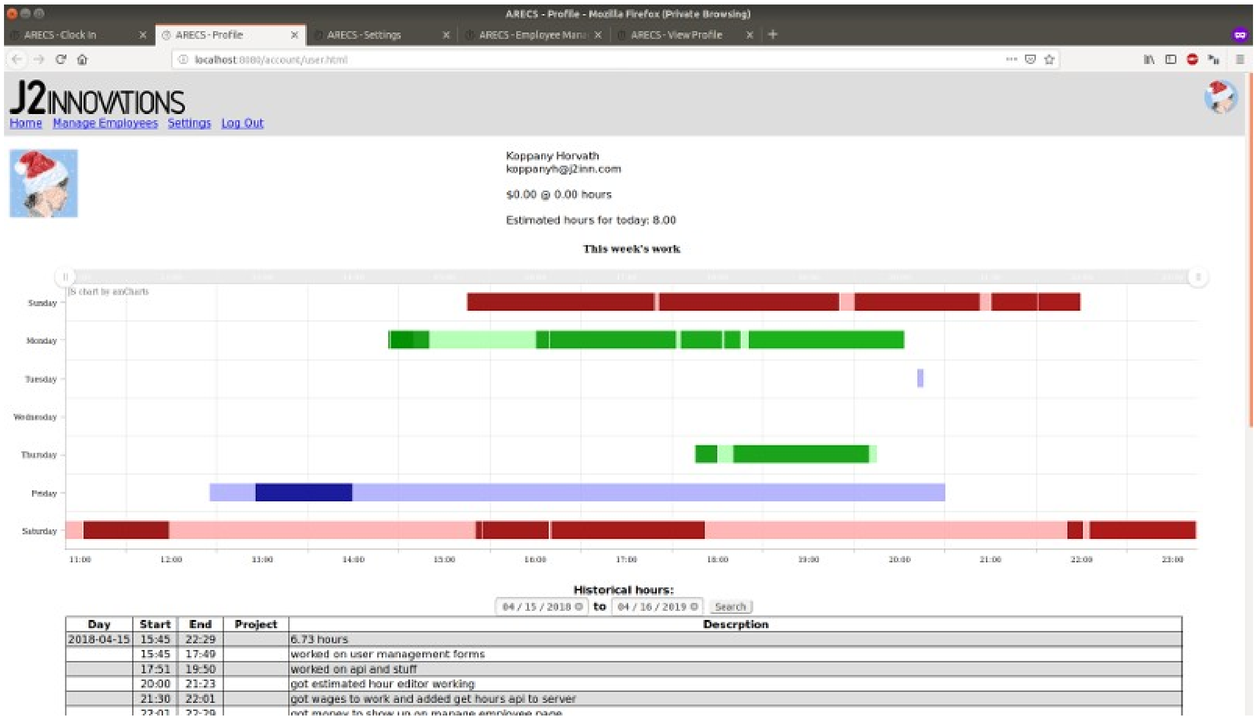
Restricted API Incidents:

Day	Time	API Call	User
2019-04-16	20:48	getscanners	Tuma, Andrew
2019-04-16	20:48	getemployees	Tuma, Andrew
2019-04-16	20:48	geturegrfidevents	Tuma, Andrew
2019-04-16	20:48	getincidents	Tuma, Andrew

Unregistered RFID Events:

No unregistered RFID events

Figure 35: Settings page



Koppany Horvath
koppanyh@j2inn.com

$0.00 @ 0.00 hours

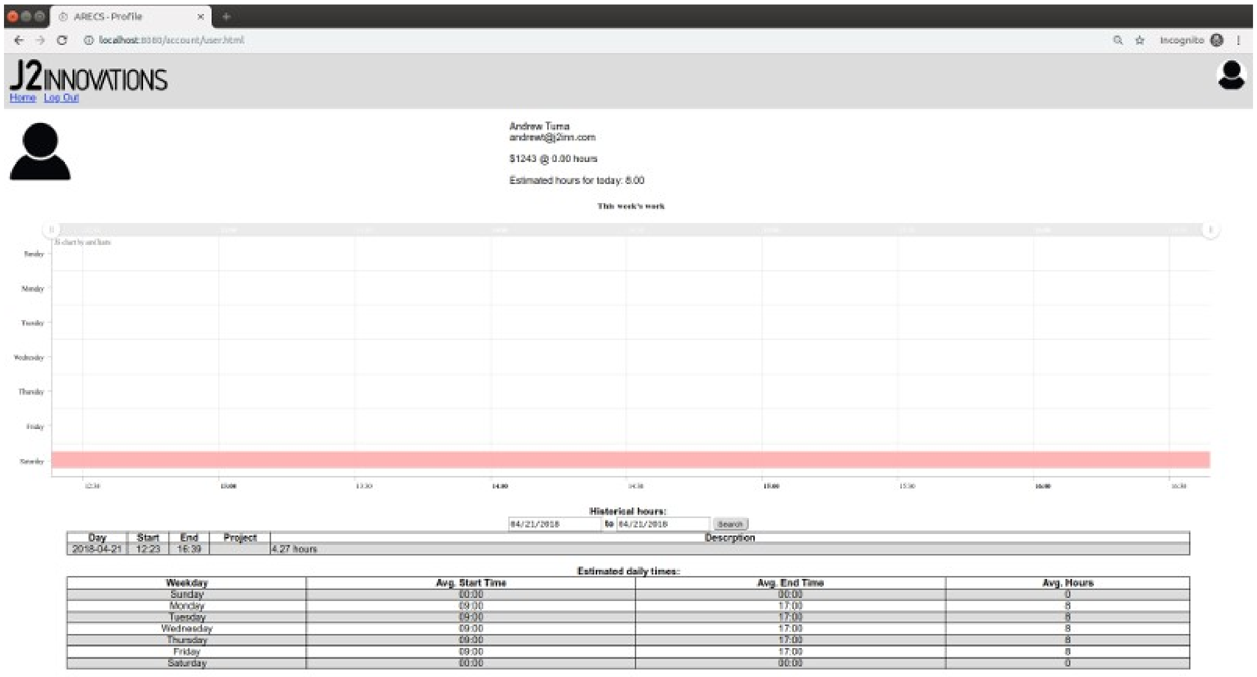
Estimated hours for today: 8.00

This week's work

Historical hours:
04 / 15 / 2018 to 04 / 16 / 2019 Search

Day	Start	End	Project	Descrption
2018-04-15	15:45	22:29		6.73 hours
	15:45	17:49		worked on user management forms
	17:51	19:50		worked on api and stuff
	20:00	21:23		got estimated hour editor working
	21:30	22:01		got wages to work and added get hours api to server
	22:01	22:29		got money to show up on manage employee page

Figure 36: User profile page as seen by an admin user



Andrew Tuma
andrewt@j2inn.com

$1243 @ 0.00 hours

Estimated hours for today: 8.00

This week's work

Historical hours:
04/21/2018 to 04/21/2018 Search

Day	Start	End	Project	Descrption
2018-04-21	12:23	16:39		4.27 hours

Estimated daily times:

Weekday	Avg. Start Time	Avg. End Time	Avg. Hours
Sunday	00:00	00:00	0
Monday	09:00	17:00	8
Tuesday	09:00	17:00	8
Wednesday	09:00	17:00	8
Thursday	09:00	17:00	8
Friday	09:00	17:00	8
Saturday	00:00	00:00	0

Figure 37: User profile page as seen by a non-admin user

Appendix D: Source Code

Due to the combined line-count of the source code almost reaching 4000 lines of code, printouts of the source code are not included as the estimated page count for that much code reaches around 50 pages.

All files related to the academic version of this project can be found at the following link:

https://github.com/koppanyh/ProjectARECS/tree/master/seniorproject

The *arecs* folder contains the server and web UI code. The *rfidfirmware* folder contains the firmware files for the RFID node along with the files for the enclosure design. The *tests* folder contains miscellaneous files used in the research and development of this project, but were not needed for the final version of the project. The *text* folder contains all the document files for this project such as this document, progress reports, various images, and rubric and prompt documents for use in explaining the requirements of the project and report.

Scanning this QR code will bring you to the above URL:

A CD is attached to the physical copy of this report to act as an offline backup of this project. It contains the complete code and R&D files that are also present in the above URL.

Appendix E: Proposal

Insert proposal here

Appendix F: PowerPoint

Insert powerpoint here