Smart Water Leak Shut-Off Valve

FINAL REPORT

SDMAY21-11
Cheng Huang
Matthew Brandt - Backend Software Developer
Cody Juracek - Hardware Researcher Engineer
Curt Kissel - Frontend Software Developer
Wolfgang Morton - Hardware Test Engineer
Grace Wilkins - Report Manager
sdmay21-11@iastate.edu
sdmay21-11.sd.ece.iastate.edu

Executive Summary

Development Standards & Practices Used

- Consumer electrons = IEEE/IEC 82079-1-2019 - IEEE/IEC International Standard for Preparation of information for use (instructions for use) of products - Part 1: Principles and general requirements.
    - This standard is about designing and integrating new technologies into a consumer's home - what safeties need to be considered, what legal regulations need to be followed, etc.
- IEEE C62.36-2016 - IEEE Standard Test Methods for Surge Protectors and Protective Circuits Used in Information and Communications Technology (ICT) Circuits, and Smart Grid Data Circuits
    - This standard is about surge protectors for application on multiconductor balanced or unbalanced information and communications technology (ICT) circuits and smart grid data circuits are addressed in this standard.
- 1008 - IEEE Standard for Software Unit Testing
- 1063 - IEEE Standard for Software User Documentation

Table of Contents

List of figures/tables/symbols/definitions

1 Figures

2 Tables

1 Introduction

1.1 Acknowledgment

We would like to thank Dr. Cheng Huang for being our advisor for this project. His proposal provides an applicable, hands-on opportunity to combine hardware and software components into an end product that can be utilized by many people today. We appreciate his insights and dedication to keep us on track to produce an efficient and effective product.

1.2 Problem and Project Statement

For many homeowners and property managers, water damage caused by leaking or bursting pipes is a significant concern. Current available market options for monitoring and controlling water flow are too expensive or lack additional functionality for most homeowners to consider purchasing. This project aims to develop a low-cost water shutoff valve with the ability to monitor water flow to prevent and protect against water damage for homeowners and property managers. The product can: measure the flow of water through the waterline; connect to the internet for real-time water control and monitoring; and provide smart features such as text alerts and an automatic shutoff mode.

This product prevents water damage while keeping the component cost around $250, making it an excellent preventative investment for property owners. It delivers a complete internet-connected hardware system that can monitor water flow; shut off the water valve; and be controlled via a user-friendly smartphone app. The app monitors both water flow rate and the state of the control valve.

1.3 Operational Environment

The project's hardware components will be installed into the property's main waterline. These areas can be damp and prone to dust; however, the device's components are enclosed inside a water resistant container. The only exposed components are rated to have a high moisture tolerance, such as: the solenoid control valve, flow sensor, and several wires wrapped in water repellent material. The app serves as the device's main user interface.

1.4 Functional Requirements

- Hardware can be installed in a home or business setting.
- Hardware can monitor the water flow rate through the waterline, then send and store that data on a server.
- Hardware can shut off water flow within the waterline.
- The solenoid control valve can operate under three settings: 1) manual override close; 2) manual override open; 3) automatic via schedule and user interface.
- Hardware can function autonomously with its last updated schedule without WiFi connected.
- Hardware can function autonomously without power for up to 6-hours via rechargeable battery.
- The app can communicate with the hardware to open and close the water valve.
- The app allows the user to view the history of water flow.
- The app allows the user to monitor current water flow.
- The app is responsive and easy to use.
- The app allows the user to set a weekly schedule when water should not be flowing, and close the solenoid automatically when water is detected.

1.5 Non-Functional Requirements

- Total hardware components should stay within a $250 goal.
- The app receives updates from the device every five seconds.
- The user can only connect to the device if they are connected to the same WiFi network.
- The water usage data should be accurate within 5% error of real values
- Updating the state of the valve should be responsive within a 2 second delay

1.6 Intended Users and Uses

The intended users for this product are homeowners and other property owners. Water damage can cause structural damage to buildings and be expensive to repair. This product will supply property owners with a cost-efficient way to prevent water damage and monitor water usage via the smartphone app.

Figure 1.6 - Use Case Diagram

2 Design Process

2.1 Previous Work And Literature

The Moen 900-002 Flo by Moen is an available market option that provides an app and a device to manually shut off the flow of water in a home. It also can record the amount of water used. While our product functions similarly to the Moen device, it costs much less and includes additional features.

2.2 Technology Considerations

As far as software is concerned, we chose to use Android Studio because it supports the Java programming language. While this may limit the amount of users for the application, it is simply easier to integrate all of our ideas in Android Studio. Our software developers are more familiar with Android Studio than other app-development environments.

For the backend, we chose to use the Spring Framework since it is Java-based. It also provides many built-in packages which will simplify many common implementations that we will need. It is also widely used in industry, so it is good to become familiar with it for our project. It is also modular and supports dependency injection, allowing for easier testing of entities in isolation.

During our initial designs we were considering what microprocessor we should use - what would be cost effective, reasonable in size, and powerful. We decided on the Arduino Nano as it has the same capabilities as a regular Arduino. It is easy to program and is significantly smaller in size, allowing the design to be more compact. We did consider a Raspberry Pi, however that device is bigger in size and we are not quite that familiar with programming those devices.

2.3 Initial Design from CPRE 491

During the early design stages back in Fall 2020, the design team immediately focused on who would be the targeted consumer. Most property owners jump on the idea of a product that will potentially save them thousands of dollars, but are usually stopped by the initial costs of the device and installation fees. It is also assumed these same property owners lack the necessary plumbing knowledge to implement such a device. Finally, we wanted a friendly user interface for smooth communication between the user and the solenoid and flow sensor.

For the app, the backend created an application that runs on local machines. It has a successful connection to a MySQL database and can properly store and retrieve data. The controller and service classes for the User entity had been created. The methods to create a user, get a user, get a user's username/email/phone number/password, get a list of all users, and update a user had all been tested. All of these would be used so that a user can create an account for the application. This would allow users to connect their water control device(s) to their account.

Our hardware design is for flow rate where a flow sensor can be integrated into the water piping system along with a control valve. The flow sensor is a Hall Effect sensor, where the water flow will push a hanging magnet up-horizontally creating a magnetic field that will inform the user water is flowing. When the magnet is down, the sensor will indicate zero water flow. The control valve is a water ball valve, a device that opens and closes water flow, usually operated manually by the user, however our design requires something more automatic. We would use a rotating motor - connected to the Arduino Nano - that will be placed above the valve, and enable the user to open and close the valve with the app or automatically due to a tripped sensor.

The design incorporates a WiFi module that allows the user to interact with the device via the mobile app. LEDs would also be used to visually indicate the device's current mode and if water is flowing or not. To view a visual overview of our project, refer to figure 2.3.

Below is the initial ER Diagram for the various entities of our project. A user is created before they can login. Each user can control multiple devices, which can be added by the user. Each device has a list of schedule items, which state the day of the week and the times that the device should be off when in automatic mode. Finally, the water data is the amount of gallons and the gallons per hour that the flow sensor detects.

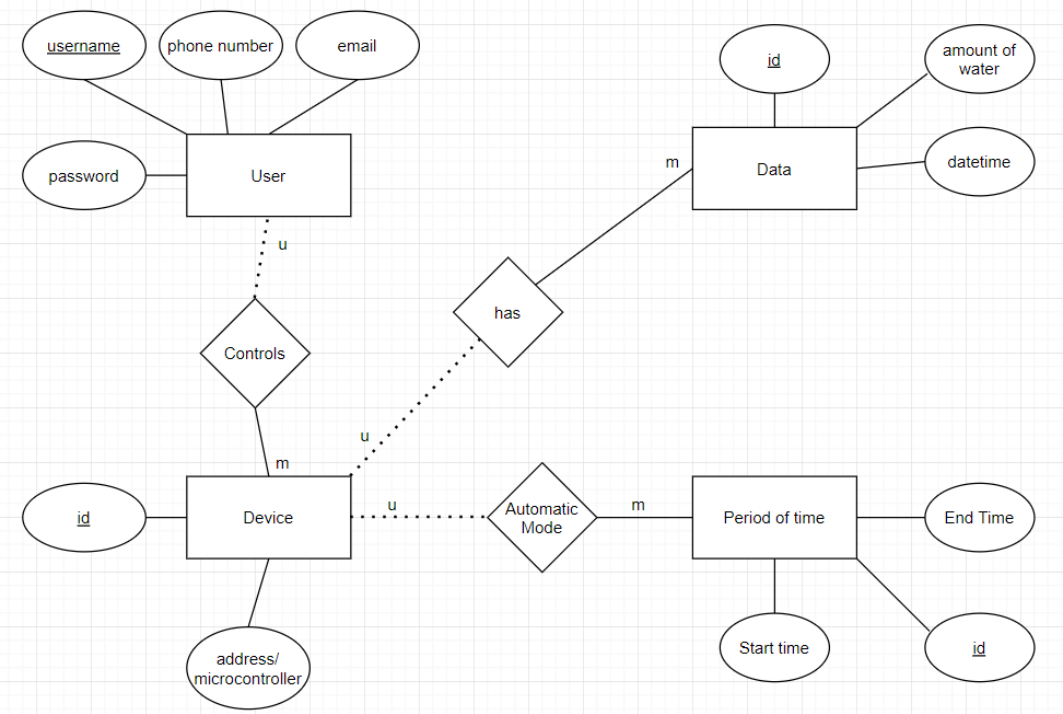


Figure 2.3 - Design Options

In our initial Gantt chart for this project was created prior to losing a member on the software team. A machine learning algorithm was to be used in order to differentiate between wanted and unwanted water usage. In the actual schedule, we ended up spending different amounts of time on tasks than expected. Below is our initial second semester Gantt chart.

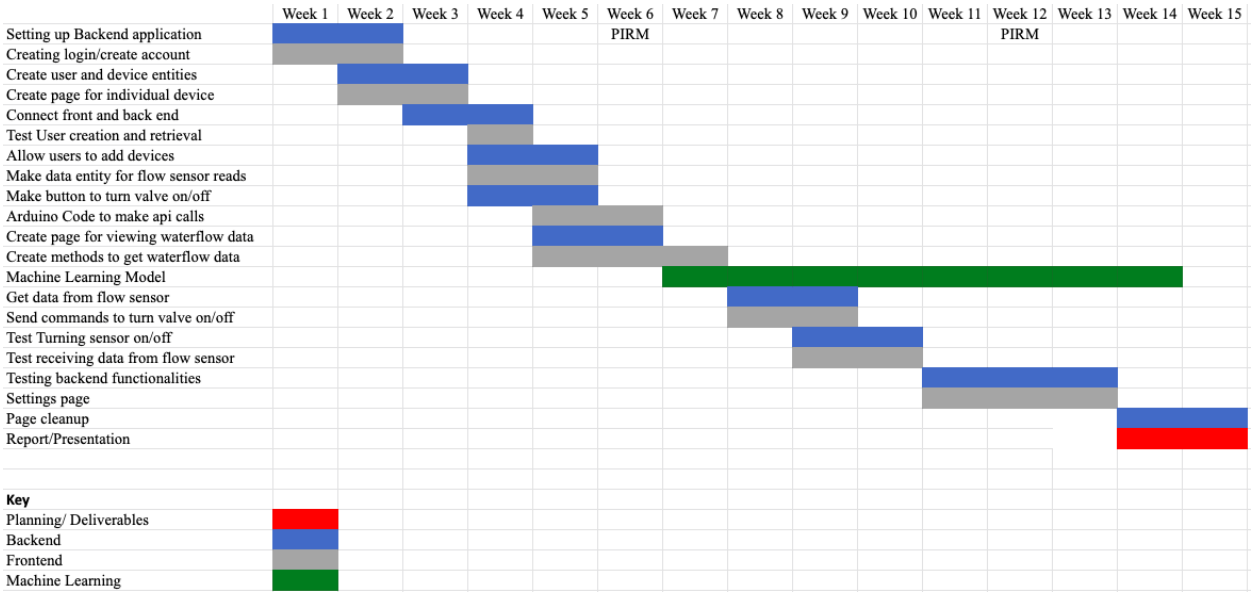


Table 2.3- Initial Schedule

2.4 Final Design for CPRE492

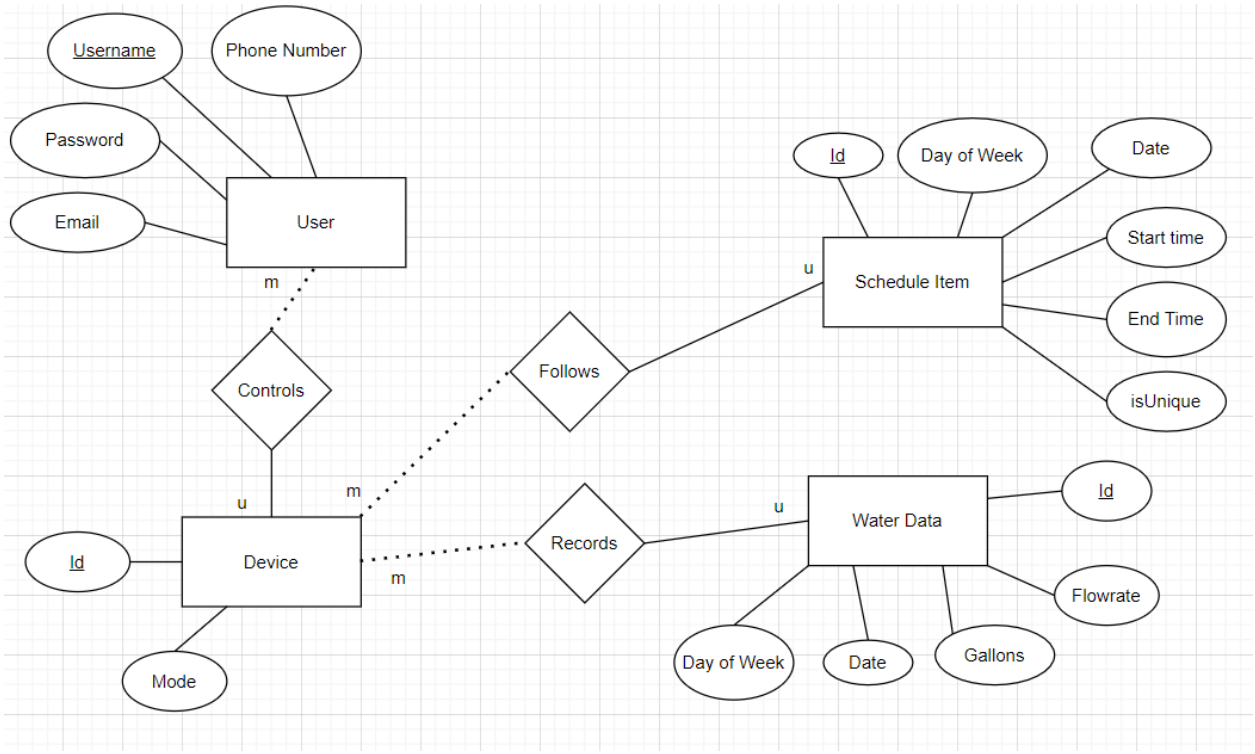


Figure 2.41 - Design Options

Above is the final ER Diagram for the Spring application. It shows the different entities that make up the application, their traits, and their relationships. The design was improved from the original to add some quality of life improvements for the user. For example, the Schedule Item has an isUnique field so that the system knows to automatically delete a block of time when it has expired. Extra fields were also added to the Water Data entity to make it easier to view recent water usage.

The final design requires plumbing as the system is connected to the main water line. A normally open solenoid is used to allow or restrict water from flowing. Water is restricted automatically if no water is scheduled, or manual by the user either through the application or the override switch. Water will be detected through the use of a hall effect sensor and the information is uploaded to the application by the WiFi module. The WiFi module is a NodeMCU ESP8266-12E. This module has more data pins than our originally planned ESP8266-01. These pins are utilized to update the user not only of the state of the solenoid, and the water flow, but also if the battery is running low on power, and to communicate with the Arduino Nano. The Arduino Nano controls the indicator lights, activates the relay coil for the solenoid, and is used for storing information about battery usage. The Arduino Nano also reads in flow sensor data for troubleshooting. All these components can be seen in Figure 2.42.

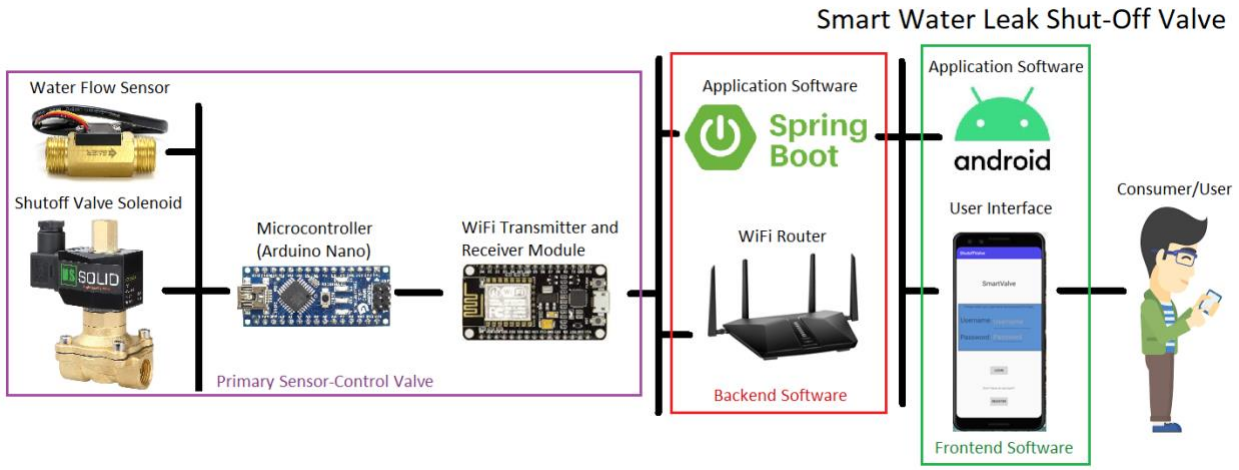


Figure 2.42 – Conceptual Sketch

In the event of a power outage, a rechargeable battery has been added for the device to remain self-sustaining up to 6 hours. If the battery is being used to power the system, the indicator light on the box will turn green. After five hours of battery usage, the indicator light will turn red and the user will be notified. The battery can be removed from the systems dry housing and charged. The system can run as it normally would off the wall adapter while the battery charges.

In addition to these components, we utilize an override switch that allows the user to set the system to open, closed, or normal operation. When the system is open, no power is being sent to the solenoid and water is able to flow. When the system is closed, no water can flow through. Normal operation allows for the system to be controlled through the application.

The final circuit diagram can be seen in Figure 2.43. The PCB schematic can be seen in Figure 2.44. The final physical circuit layout can be seen in Figure 2.45.

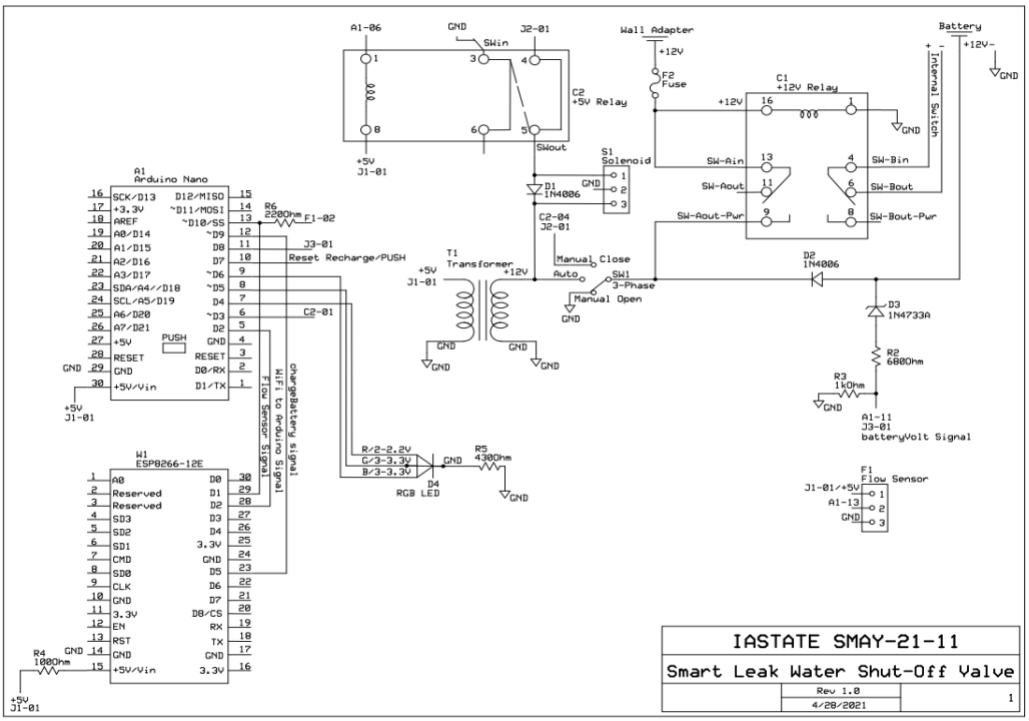


Figure 2.43 – Circuit Schematic

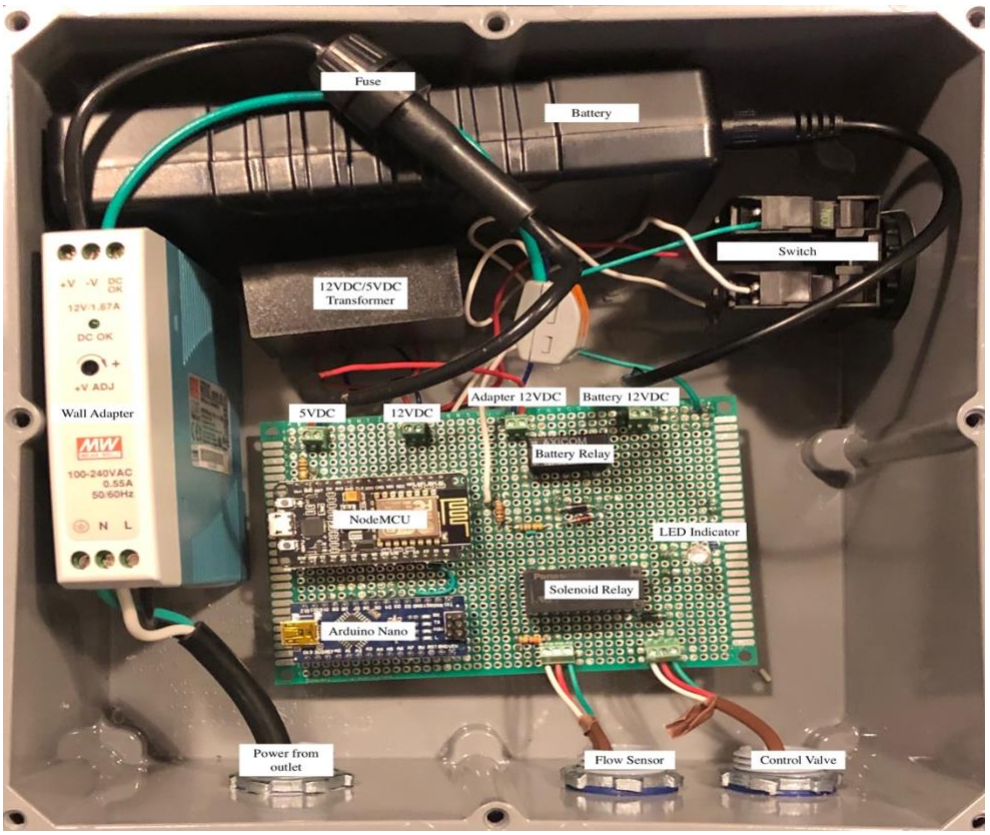


Figure 2.44 – Design Options

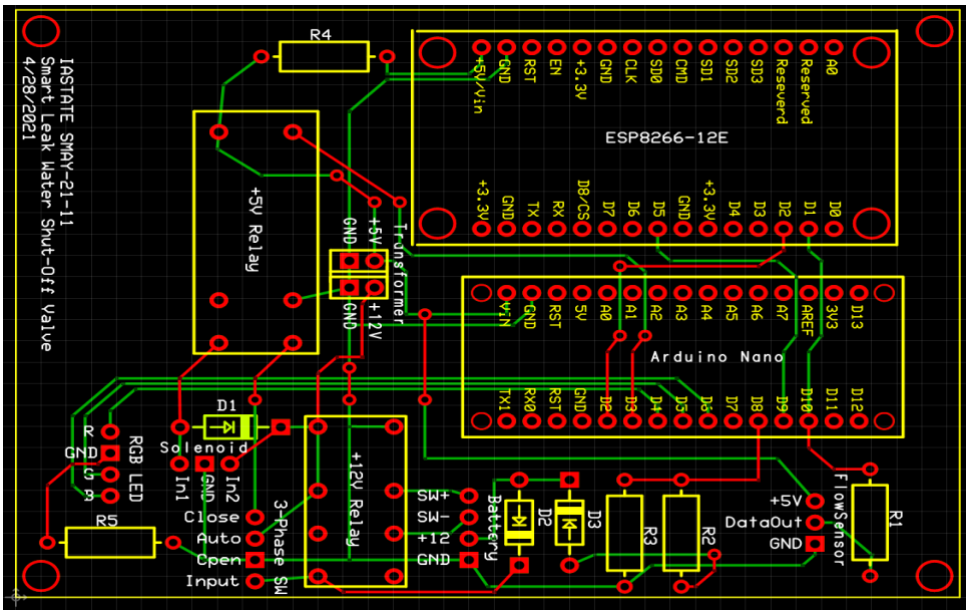


Figure 2.45 – PCB Layout

The total cost for all these components can be viewed in table 2.41.

Parts	Cost
Solenoid	$42.80
Power Supply	$13.50
Arduino Nano	$4.66
Flow Sensor	$10.49
Wifi Module	$4.66
Housing	$28.45
PCB	$51.00
Transformer	$8.98
Fuse	$2.59
220 ohm Resistor	$0.50
1n4006 Diode	$0.14
1N4733A	$0.12
5V Relay Switch	$2.76
12V Relay Switch	$3.66
Rechargeable Battery	$37.99
RGB LED	$0.09
Total Cost:	$212.39

Table 2.41 – Parts and Cost

The final Gantt chart reorganized a lot of the tasks. In this chart, testing and development are counted as one task rather than two separate tasks. Another notable difference is that we took some time at the beginning of the semester to redo some of the planning from the previous semester to account for the lost group member. The machine learning component of this project was scrapped in the actual implementation

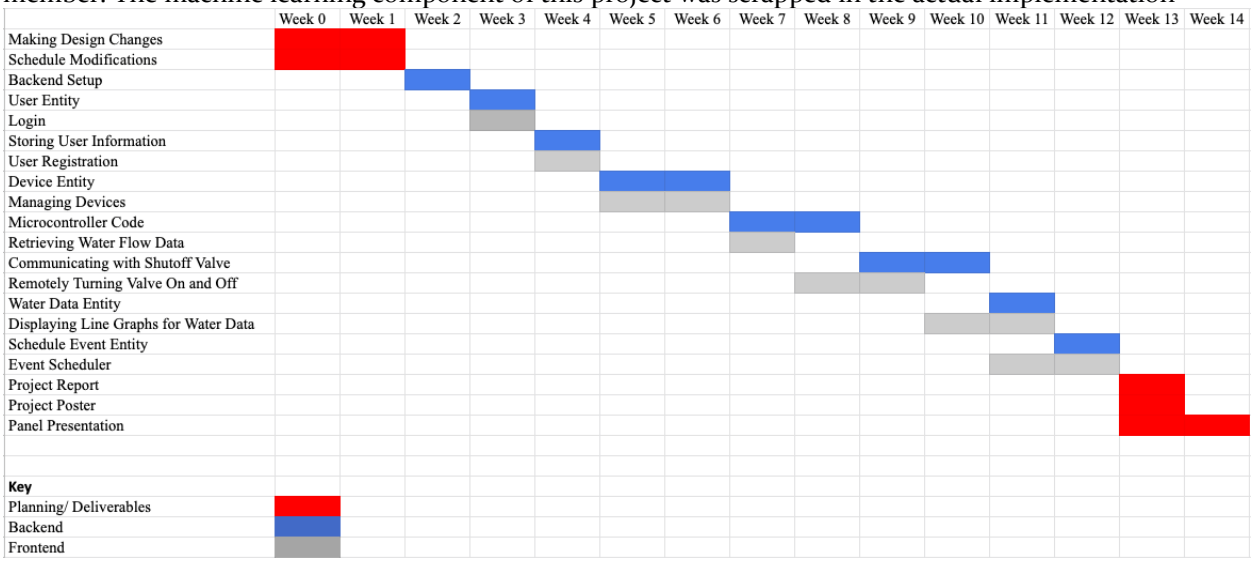


Table 2.42 – Final Schedule

3 Security: Concerns and Countermeasures

One security concern is that learning a user's username and password would allow a malicious party to control the water flowing into a person's home. In order to protect the user's information, the application was configured to work with HTTPS. This will encrypt the data when communicating between the user's phone, the server application, and the water valve switch.

One physical countermeasure that was incorporated to the circuit was a manual off switch that the user can flip to turn off the device. This is in case something goes wrong or a malicious party obtains access to a user's account. This will allow the user to turn off the device manually.

A final concern was what would happen in the result of a power outage. We incorporated a rechargeable battery into the circuit that would allow the circuit to run for up to 6 hours without power. The WiFi module will then also alert the user through text when the battery needs to be recharged.

4 Testing

4.1 Unit Testing

On the software side, we have used the following testing frameworks:

Testing Framework	Purpose of Tests
JUnit	JUnit allowed us to test individual methods for correctness. This will ensure that the logic within each method is sound.
Mockito	Mockito tests were used to "mock" a response from the hardware component while it is still in development.

Table 4.11 - Software Testing

On the hardware side, we conducted the following tests:

Testing	Description
Controlled environment	We will be testing our design on a homemade water piping system, to see if it works before we integrate into the real thing.
Control Valve Operation	Utilize multiple MCUs to signal a change of state. Determine response time as best as possible and what may cause the valve to fail.
Sensor Accuracy	Experiments will be run with both known times and known water volumes to find an accurate time. Utilize calibration information provided within the data sheets and modify as needed.

Table 4.12 - Hardware Testing

Our application relies on properly storing the user's water usage data, as well as properly determining whether or not the water is running. Because of this, there will be a large focus on accurately receiving information from the shutoff valve, and properly storing and fetching user information.

We tested all entities within our database with both JUnit and Mockito tests. The mockito tests mocked responses from the database to ensure that the methods are properly handling the received data. The JUnit tests ensured that the proper information was received from the backend.

The response received from the valve itself will be tested in isolation so that the app and the valve itself are properly linked together. When the valve is on, the app must receive a signal from the valve so that it can properly record water usage.

4.2 Results

The control valve response time is incredibly quick when the signal is sent from either the ESP8266-12E NodeMCU or the Arduino. We are aware based on the data sheet that the valve should not be closed continuously for more than eight hours or it may overheat. However, due to testing and environmental constraints, this could not be verified. We will warn the user to not have the control valve closed continuously for longer than seven hours, and the back-up battery will only keep the valve closed for six hours before needing to be recharged.

Accuracy testing for the flow sensor allowed us to do much needed calibration of data the ESP8266-12E NodeMCU was outputting. We found that the gallons per hour needed to be multiplied by 0.13 to achieve accurate results. When this modification was made, the accuracy of the readings was consistently within 1% of the expected. The figure below shows the expected values in red and the values read by the NodeMCU in blue.

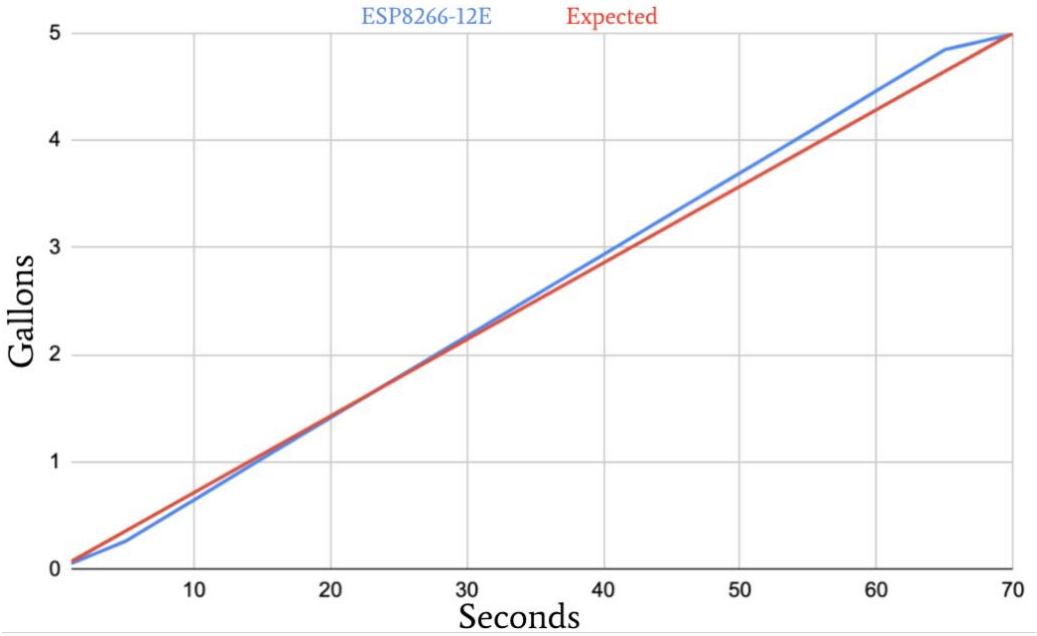


Figure 4.2 – Flow Sensor Accuracy Graph

5 Implementation

The backend server application was made using Spring Boot for Java, a language very familiar to the team, as it provides many libraries to easily build an application from scratch. It also provides dependency injection to keep the different classes modular and easy to maintain. The Spring application uses a MySQL database to store information about different users, their devices, and their water usage.

The front end application is a Java Android Application made by using Android Studio. This was chosen due to our familiarity with Java and the Android IDE. Several libraries, such as Volley, allow the Android application to easily send requests to the Spring application.

The water valve is made up of an ESP8266-12E NodeMCU; an Arduino Nano; a normally open solenoid; a Hall Effect Sensor for water flow; and a rechargeable battery. The NodeMCU's WiFi connectivity allows the user to control the water valve from the Android app. The NodeMCU constantly polls the Spring application for the state of the valve. It then sends a signal to the solenoid via Arduino to open or close the valve, depending on the state set by the user. The NodeMCU also reads the water flow rate directly from the flow sensor, and sends that information to the Spring application.

The Arduino acts as a buffer for the NodeMCU, solenoid, and water flow sensor. When the valve needs to change state, the NodeMCU sends this information to the Arduino. The Arduino will send a digital high signal to the solenoid, thus closing the valve, and similarly a digital low signal opens the valve.

The water flow sensor, attached to the main waterline of the property, reads the amount of water passing through it, then writes this information to the NodeMCU and Arduino.

The rechargeable battery is for when power loss is detected allowing the solenoid to continue running for up to 6 hours before it must be recharged. The NodeMCU will also alert the user through text messages when the battery needs to be recharged whenever WiFi communication is reestablished.

6 Closing Material

6.1 References

Moen Flo:
Press Release. (2020, October 27). Retrieved from Moen: https://www.moen.com/press-room/press-releases/lennar-moen-partnership

6.2 Appendices

Appendix A: Alternative Implementations

Initially, there was going to be a machine learning component that could differentiate between wanted and unwanted water usage. This was scrapped because one of our team members on the software side did not return this semester.

Our team also researched non-plumbing solutions to the combat water leaks. This would include a motor that would attach to the main water inlet control valve and vibration sensors that would attach to the pipe. However, due to inconsistency amongst homes for valve types the motorized control would be difficult to replicate on a large scale. Additionally, vibration sensors would be susceptible to inaccurate readings if the pipes are banged on, located near washer and dryer machines, or many other ways that could cause the pipes to move even slightly. Due to these concerns, our team focused on the plumbing required option and came up with our final product.

While creating the circuit to control the water valve solenoid, we attempted a number of solutions. As the solenoid operates at 12VDC and the Arduino Nano outputs at most 5VDC our solution had to incorporate both voltages. Our initial design focused on the use of a n-type BJT. The base connected to the Arduino Nano, the collector to the solenoid, and the emitter to ground. While the solution did work, the BJT would get incredibly hot quickly. With the circuit being in an enclosure without any air flow, there was concern about the heat emitted by the BJT causing other components like the power supply to run less efficiently. It was because of this problem, a relay was implemented into the system instead.

The shut off valve and application allow users to manually open and close the valve, view recent water usage data, set schedules for the device to follow when in automatic mode, and manage multiple devices on one account. The project could be improved upon in the future by implementing a machine learning algorithm to learn the user's water usage habits. This would allow the application to automatically shut the valve when abnormal water is detected without requiring the user to input a schedule to follow.

Other improvements that could be made include small quality of life changes to improve the user's experience, as well as user interface design changes. A timer could also be introduced to the circuit to get more reliable and responsive updates.

Appendix B: Other Considerations

Overall, the project taught a great deal about time management and expectations. If everything in our project had gone perfectly the first time, we would have consistently been ahead of schedule. This was frequently not the case and certain aspects took longer to even be able to test. Additionally, communication was a very crucial part of the project. Not only did the hardware need to communicate with each other but amongst the groups, ideas and concepts needed to be discussed. An example of this can be seen within the hardware side. There was a problem with the indicator light still indicating that the battery source was supplying power, even if the circuit was being powered by the wall adapter. Was there an error in the code?

Was the signal for battery power coming from the wrong spot? Was there a problem with the circuit overall? The team had to come together to determine it was actually a coding error due to a misunderstanding on behalf of the Arduino programmer.

Another frustrating thing we learned is that we have to be certain that the parts we are using are capable of performing in a way we expect them to. As we were progressing through our project, we decided that the Arduino Nano was no longer necessary and planned on only using the NodeMCU. After hours of debugging, trying to figure out why we were not able to control the valve, we discovered that the GPIO pins on the NodeMCU could only output voltages up to 3.3V, which was not enough to trigger the relay to stop the water. Then we had to incorporate the Arduino back into our circuit in order to control the water valve. We wasted a lot of time trying to figure out something that was never capable of working. This problem could have been avoided if we had a better understanding of the parts we were using.

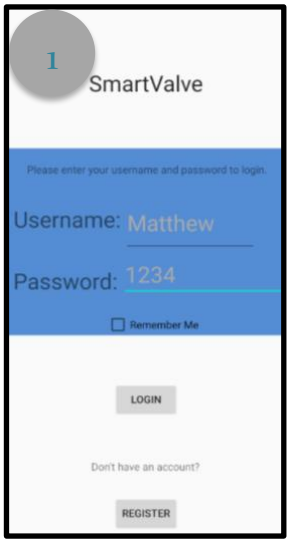
Appendix C: Operation Manual

Step 1: Installing the Water Valve System

The first step to installing your new Water Valve System is to determine where on the main water line it will go. It is recommended that you install this valve before the point where the lines branch off. Once you have determined the best position, you will need to install the valve into the pipeline. If you do not have much experience with plumbing, or are uncomfortable with removing and installing pipes yourself, then it is recommended that you hire a plumber to install this part of the device. Once you have attached the valve to the pipeline, the next step is to install the control box. There are a few ways to install the box, so it is up to you the user to pick the best fit for you.

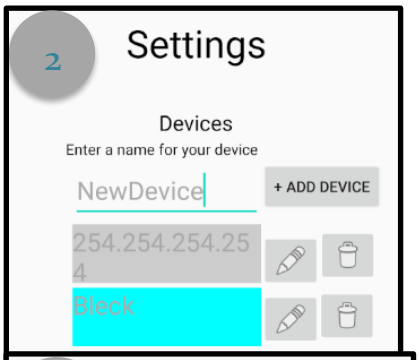
1. First, you will need to attach the provided mounting wings to the box.
2. Next you will want to find the best spot to hang the box. You will want the position to be in a place where you can still access the box and the switch as well as be near a power outlet. Some suggestions are:
    a. A nearby wall
    b. A cross beam
    c. The pipe itself.
3. Once you have determined the best place to install your box, use the appropriate screws or pipe fittings, not included. It is important the box be kept upright. To know the box is in the upright position, the wires will be coming out of the bottom.
4. Once you have installed your box, be sure the switch is in the first position, then plug the box into the nearest outlet. By having the switch in the first position, it starts the device in "Off-Mode". This is done to protect the device from an accidental power surge while installing.
5. Finally, once you have plugged the control box into the wall outlet, you can now flip the switch to the second position "Auto-Mode" and you can now begin your WiFi setup.

Step 2: Connecting to your Water Valve Switch



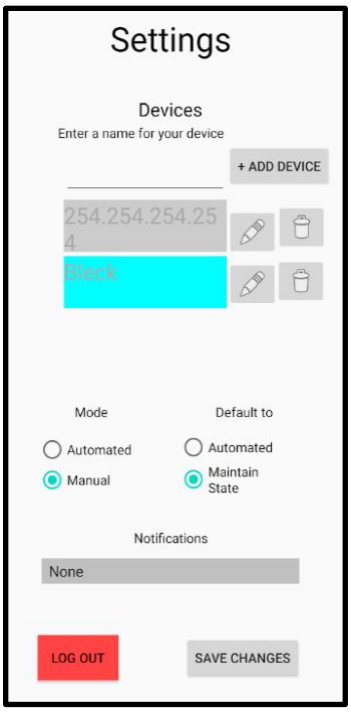
After downloading the Water Valve application and successfully installing the device, the next step is to connect your account to your device.

1. First, either create an account on the application or log into an existing account.
2. This will take you to the main page. From here, click the settings button in the top right corner. Here, you can add a device to your account by simply entering the name of the new device and clicking "Add Device".
3. Next we must allow your Water Valve Switch to connect to your WiFi network. After installation, use any smart device to connect to the WiFi network called 'MyESP8266' with password 'testpassword'. Then, navigate to your browser and input the IP address: 192.168.4.1. Here you will be asked to input the name of your WiFi network, the password, and the name of the device you entered in the application. It is important to enter the name of the device exactly as you entered it in the application.



Step 3: Using the Application

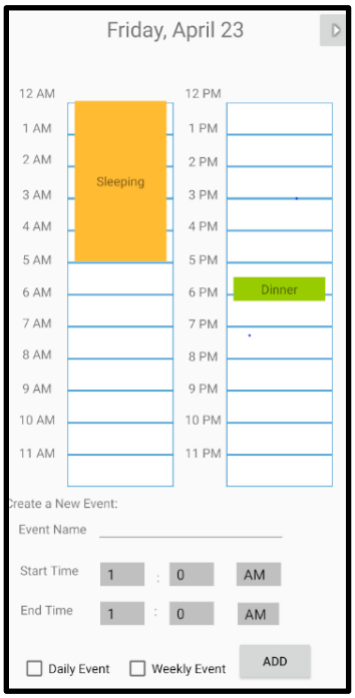
1. After logging in to your account, you will be taken to the application's home page. This page gives the name of the user, the name of the device currently in use, and the current state of the device. The user can also see some of the most recent water usage data and manually control the water valve.
2. From this screen, you can click on the settings page to add and alter devices, the line graph to view water history, or the scheduler to add items to the schedule.



Step 3a: The Settings Page

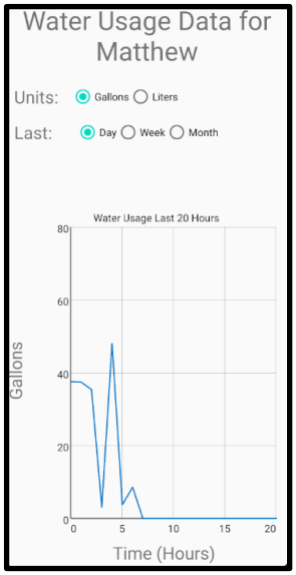
1. From the home screen, you have access to the settings page. To add a new device, enter the name of the device and click 'Add Device'.
2. You can also view the devices linked to your account. To change the currently selected device, simply click on the device you would like to control. The selected device will turn blue.
3. You can also control the mode of the water valve. In automated mode, the device will follow the schedule based on created events. In manual mode, you can open and close the valve yourself.
4. You can also control the default state of the water valve. This tells the water valve how to operate if WiFi connection is lost. In automated mode, the valve will follow the schedule. If 'Maintain State' is selected, the valve will remain open if it was open and remain closed if it was closed.
5. To save the changes of the current device, press the 'Save Changes' button in the lower right corner.
6. The last feature of this page is the 'Log Out' button. If you would like to log out of your account, pressing this button will return you to the login screen

Step 3b: The Schedule Page



1.From the home screen, you have access to the schedule page. The top of the page shows the current date as well as any scheduled events where the water valve should be turned off automatically, as long as the device is set to automatic mode.

2. To add a new event to the schedule, enter a name for the event and the start and end time of the event. Upon pressing the "add" button the event will appear in the grid as long as the necessary fields are filled with valid data.
3. There are three separate types of events that can be added: one time events, daily events, and weekly events. If the "Daily Event" checkbox is filled, then the event will be recurring every single day. If the "Weekly Event" checkbox is filled, then the event will be recurring every single week on that same day of the week. For example, if we add a weekly event on Thursday, then every Thursday after that will also contain that event. If neither of the checkboxes are filled, then the event will be a one time event.
4. A user can view their upcoming scheduled events up to one week in the future. They can also schedule a unique event to occur up to one week in advance.



Step 3c: The Graph Page

1. Also available to access from the application's home screen is the Graph page. On this page, the user can view their water usage from the past 20 hours, the past week, or the past 20 days.
2. This screen allows the user to visualize their water consumption, and see how their water usage behavior varies throughout the day, week, or month.
3. The user has the option to view the data in either amount of gallons used or liters used.