University of Arkansas – CSCE Department
Capstone II – Final Report– Spring 2023

GateMate

Jackson Bullard, Nathaniel Fredricks, Jose Martinez, Carissa Patton, Ivris Raymond

Abstract

The use of alternate wetting and drying (AWD) is gaining popularity among rice farmers in Arkansas as a way to reduce water usage without sacrificing crop yield. Doing this effectively requires frequent adjustment of levee gates, and timely accuracy is necessary in order to avoid diminished crop growth. This is difficult to achieve with traditional gates, where workers must manually adjust each gate, a process that is both time consuming and prone to human error. Gates remotely controlled by a stakeholder through software such as a mobile app would provide an effective solution to this problem.

The GateMate software will be developed to utilize and interface with an existing GateMate device prototype, along with the sensor array present on the prototype and any necessary field tracking technologies. The GateMate software application will not only allow rice farmers to raise and lower gates in the field remotely, but it will also factor in weather conditions to automatically adjust the gate levels to conserve the most water as well as assisting with optimal initial placements of the gates.

The architecture of the GateMate application will consist of four separate pieces of software: the mobile application that the users interface with, the central server and database, raspberry pi for disseminating server commands to the gates, and the gate software that will run on each gate's control circuit. The mobile application will be implemented using Flutter, which is Google's open-source framework for multi-platform applications from a single codebase [12]. A database will be used to store the necessary geographical and weather data needed for optimal gate placement as well as locations of fields and gates. The third piece of software will network the gates using the ESP8266 Wi-Fi Mesh API, allowing the user to issue commands to individual gates or groups of gates through the Flutter Application. Finally, in order to enable remote commands to be forwarded to the gate network, it will be necessary to create an intermediary module in the network capable of connecting to the cloud back-end and the gate network.

The significance of an application like this involves resolving many issues faced by rice farmers today, including aiding in water conservation. Being able to remotely control gates will help farmers reduce excess water consumption, allowing farmers to save money and use optimal water levels to maximize crop yield. Additionally, one with less irrigation experience would be able to use the GateMate software to identify the most advantageous gate placements on a given topography as well as automate the process of modifying gate heights to regulate water levels on the field.

1.0 Problem

Rice, wheat, and maize provide over 50% of all calories consumed by the world population [1]. Alone, “rice provides 21% of global human per capita energy and 15% of per capita protein” [1]. Arkansas farmers harvest over 200 million bushels of rice on an average of 1.3 million acres annually, and, in 2015, Arkansas accounted for 49% of the United States’ total rice production [2]. More recently, rice production in the U.S. exceeded $3 billion in 2021 [3].

While many irrigation techniques exist, an increasingly popular technique referred to as alternate wetting and drying (AWD), also known as intermittent flooding, is gaining traction in Arkansas [4]. Already popular in surrounding states, this water management practice is intended to reduce water usage and greenhouse gas emissions without sacrificing crop yield; however, it can be labor intensive due to the need to frequently adjust gate height in order to precisely control water level. The current stage of growth of the crop, soil characteristics, and weather conditions can all affect the desired level of water. Furthermore, with so many factors in play, there are many chances for human error year after year.

In order to regulate traditional gates to maintain desired water levels, workers must travel out into the (usually flooded) field and manually adjust the gates. This is time consuming and error prone. Workers may not always appropriately adjust the gates, and farmers may incorrectly deduce the proper gate height; many gates can feed into the source of others, creating a complicated irrigation chain that complicates the decision-making process. Furthermore, workers may neglect adjusting the gates entirely, and the probability of this increases with the difficulty in reaching a gate and the granularity with which the stakeholder wishes to control the water levels. Precise control can be necessary for maximizing crop yield, and lost yield is a cost intolerable to the modern farmer.

As the world’s population grows, the Earth’s natural resources bear increasing strain. Some estimates suggest that the near-to-mid future may require as much as 40% more water than is currently accessible [5]. Furthermore, water withdrawal by rice can be significantly higher compared to other field crops (depending on method of irrigation) [6]. AWD is designed to minimize wasted water, and automated, weather-informed irrigation systems present an opportunity to utilize rainwater to further reduce the need for water withdrawal. In addition to increased water conservation, every acre-inch of groundwater that is offset by rainwater capture (or otherwise not pumped from a surface/groundwater source) could save as much as a gallon of diesel fuel [7]. The opportunity to conserve natural resources without sacrificing yield necessitates investigation. A solution would benefit both rice farmers’ livelihoods and the Earth’s increasingly strained ecosystems.

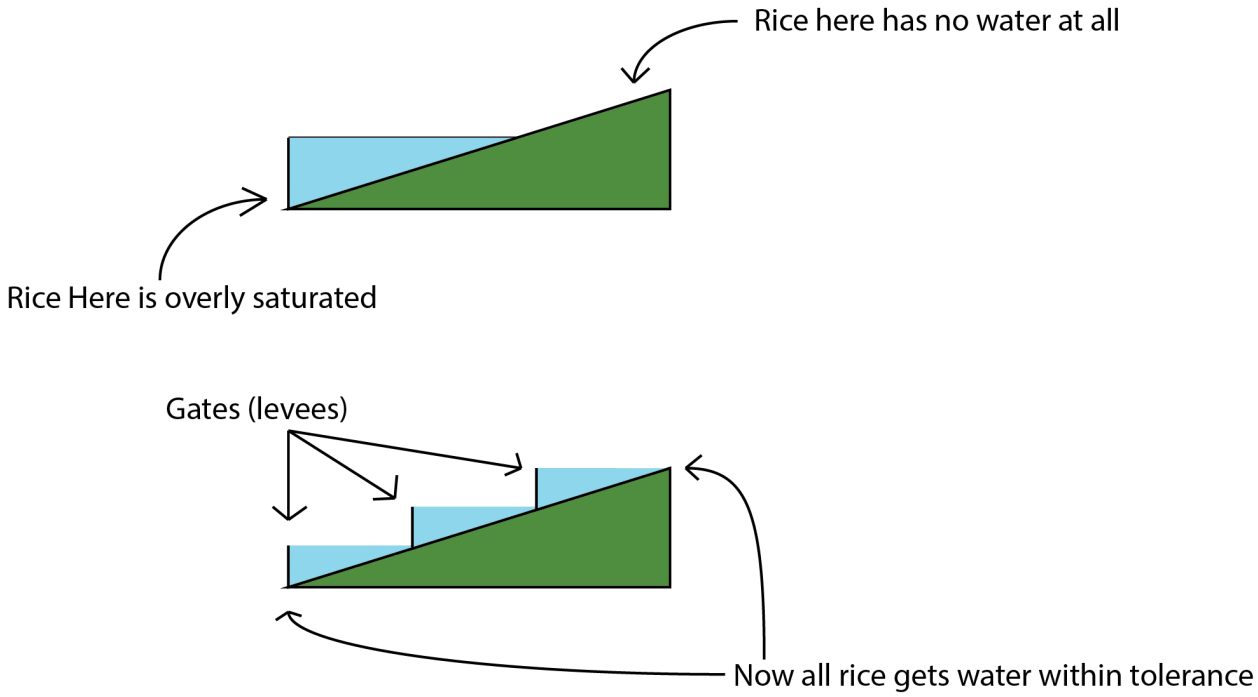

Figure 1:

The figure above (Figure 1) illustrates the purpose of the gates in a field. Without gates, a farmer cannot properly regulate water within the tolerance of the crop throughout the entire range of elevations across their field. With gates, it is possible to create zones of certain water levels to mitigate this issue. Proper management of these gates is essential to crop yield, and our project aims to make this management easier for our users.

2.0 Objective

- Required Objectives
  - Interface allowing user to manually raise and lower gates
  - Application to assist with the initial placement of gates
  - Solution automatically raises and lowers gates according to current water levels as measured by sensors on the gates
  - Solution notifies user of forecasted weather to allow for changes in paddy water input
- Optional Objectives (Stretch Goals)
  - Allow user to see the path of least resistance for the water
  - Solution automatically raises and lowers gates according to…
    - Crop height
    - Crop growth rates
    - Forecasted extreme weather
  - Application alerts user of suspected runoff using data from water-well flow meter

A feature complete implementation of a solution should provide farmers with an interface for GateMate that is available at least on their mobile phones and ideally across several platforms. The GateMate application should provide most of this information through a map interface that displays the user's field and gates.

The minimum viable product version of the GateMate software should allow farmers to raise and lower the gates manually. The software should also assist farmers in optimally placing their gates given their field's topographical information. This is important as the gates will differ in elevation, so allowing an amount of water into one section of a field may mean raising a gate more or less than another gate controlling the water level in another section of the field. This degree of implementation will at least accomplish the objective of making it possible to move the gate positions without having to be physically present to do so. Finally, at a minimum, the solution should notify the user regarding relevant weather developments (using, for example, the Google Earth Engine API [8]). There exists an opportunity to conserve water by leveraging rainfall to maintain field water levels. Since GateMate does not control the well from which water is supplied to the paddy, the farmer will need to adjust the amount of water coming from the well using methods outside of the scope of the GateMate project. While most weather events will be handled by other aspects of the project (i.e., the water level sensors present at each gate), it may be the case that extreme weather could require advanced adjustments of gate levels. In this case, the user can make the necessary changes after being notified by the mobile application. Even still, an ideal implementation would be able to prescribe gate changes according to extreme forecasted weather; the fickle nature of forecasted weather, however, would demand a finely tuned threshold for the required certainty of said weather before making automatic adjustments. In addition to the manual adjustment of the input water from the field's well, the real-time, software-controlled regulation of gate heights according to measured water levels should be more than enough to capitalize on rainfall to conserve water usage.

As a final stretch goal, our project should allow the farmer to instead input just the current height of the crop, then use that information to automatically calculate the amount of water the crop needs. The primary variable concerning a rice crop's water level requirement is the current height of the crop. The GateMate software should receive periodic crop height measurements (input by the user), likely in various locations throughout the field, then automatically calculate the amount of water required and make adjustments to the gate elevations based on that calculation and the amount of water available in the system. Furthermore, it would be possible to have the user input the date the crop was planted. The app could then prescribe water flow based on that and a model for the expected growth of the plant based on other factors such as the amount of sun the crop is getting. Since the application would already take weather into account, constructing this model would likely not be beyond the GateMate software, but the accuracy of such a model is yet unknown, so periodic height measurements done by the user would likely still be necessary.

An application that implements the above outlined features would certainly meet the objectives for this project, as the primary goal of the GateMate project is to make it easier for farmers to manage the water usage of their crops throughout the growing season. Implementing even just the minimum viable product version of the GateMate software would mean that a farmer does not need to physically raise and lower every gate to make adjustments and could do so from their phone. To also account for weather and to further automate the water adjustments based on just the height of the crops would further reduce the workload for farmers and

potentially result in greater yields due to fewer errors in water management. This technology will enable fewer farmers to manage larger fields, thus making it easier to keep up with the ever-growing demand for food as the world's population continues to grow.

3.0 Background

3.1 Key Concepts and Technologies

There are many key concepts and technologies related to this problem, including the concept of alternate wetting and drying as well as technologies such as the MSP430 microcontroller, ESP8266 Wi-Fi module, and more. While it is important to understand what each key concept and technology entails and is capable of, it is even more important to consider how each of these falls into place in the larger picture of this project.

As the proposed software system will be used in agriculture to regulate water levels on fields, it is critical to understand the currently utilized method of alternate wetting and drying (AWD), which is a form of watering fields. The "drying" part of AWD is when the flood is allowed to recede, typically until the soil surface is visible [5]. Once the water level reaches its intended low point, the fields are flooded again (the "wetting" part). This alternation is performed instead of maintaining a continuous flood in an effort to reduce the amount of water used. One way of implementing AWD is by controlling water flow by raising and lowering the rice paddy's levee gates, and the software proposed here will help make controlling an entire field of gates far less time consuming and labor intensive.

In past semesters, students have worked on other aspects of the GateMate device, with the current gate design containing a box with computer hardware to control the gate. The circuit consists of two custom printed circuit board (PCB) designs. One PCB contains the main processor in the circuit: an MSP430 microcontroller, which is the "brain" that makes calculations and controls peripheral devices. An ESP8266 Wi-Fi module is attached to this PCB, which connects the module to the microcontroller. The direct current (DC) motor is also connected to this PCB. The second PCB is responsible for recharging the battery using the attached solar panel. The PCB with the MSP430 can be programmed to run software that is designed for it. This software can be loaded onto the board from a computer using Energia IDE, which is a development environment for microprocessors and microcontrollers.

A mesh Wi-Fi network is a group of devices that act as a single network. Instead of a centralized source of Wi-Fi, such as a router, each device in the group essentially acts as a node, with each node offering Wi-Fi access to the network [9]. For example, the ESP8266 Wi-Fi module attached to the gate's controller could be set up to connect with other nearby gates and create a network that could then be accessed by connecting to any one of the gates.

Typically, mesh Wi-Fi networks are communicated with from the outside by connecting directly to the network and sending http requests to the individual nodes. In order to forward commands from our cloud back-end to the gate network, we are using a Raspberry Pi embedded computer to act as an intermediary controller. A Raspberry Pi is a small computer with a fully featured SoC built upon the ARM architecture. This offers a relatively inexpensive and easy to deploy solution for connecting an offline mesh network with our cloud back-end while maintaining separation between those two components for modularity.

An application programming interface (API) is an interface that allows developers to write software that interacts with another piece of software [10]. Whether it is an application, service, or some type of data, an API can make meaningful interactions easier. The ESP8266 API contains libraries for configuring the Wi-Fi module on the gates. Instead of writing and reading abstruse values to the memory module, the API offers abstracted functions that simplify the process [11]. This will be a great tool when creating a more complex system such as a mesh Wi-Fi network.

Flutter is Google's open-source framework used to build user interfaces [12]. When it comes to mobile phones, the two main operating systems are IOS and Android. Despite both operating systems running on mobile devices, developing for either platform is usually significantly different from the other. However, Flutter is cross-platform, so it can compile to work on both IOS and Android, allowing developers to create one app that can then run on multiple platforms.

A database is a "collection of information" that is usually designed to provide efficient retrieval of specific data [13]. When it comes to servers and software, databases are often implemented using standardized technologies such as Structured Query Language (SQL), which is utilized in many databases as a common means to query, add, remove, and change data in a database [14].

In summary, this project will utilize various technologies including Flutter, databases, and mesh Wi-Fi networks. All of these technologies will come together in the GateMate project to allow farmers to adjust their gates remotely.

3.2 Related Work

Over time, technology has played a crucial role in rice farming. Impacting both sustainability and profitability, farmers have been looking to technology to improve their methods and reduce costs [15]. In the past 20 years, farmers have reduced water usage by 53% and land usage by 35%, partly due to the shift from the constant flooding of fields to intermittent irrigation [15]. This newer approach to irrigation has even more room for improvement through further technological development, reducing wasted water while still allowing the fields to produce the highest yields possible.

Researchers have developed automated systems similar to this GateMate project, but they fall short of being ideal solutions. One software system, Delta Plastics' *Pipe Planner*, allows users to manually draw fields on a map interface and place gates on the fields wherever the user selects [16]. Firstly, *Pipe Planner* displays the farms previously added by the user along with the option of creating a new field. When the user prompts the application to add a new field, a map is displayed to the user that allows them to enter the location of their desired field whether that be a furrow or levee. The user is given the option to either draw a field or upload a file to automatically generate field boundaries. It also allows the user to specify a water source within the field view. However, while this application allows users to note locations of levees, it does not tell users the ideal place to position gates to regulate the water levels, nor does it provide automatic raising or lowering of these gates to ensure appropriate water levels. The GateMate will not only allow users to select their field but also will provide information as to the best locations to place gates on those fields and automatically modify the height of the gates based on water levels, the weather forecast, and conditions for best growth. While the *Pipe Planner* has

features that will not be implemented on the GateMate software such as setting flow rates of nearby furrows, the GateMate will provide predictive information to place gates at locations to optimize water usage and allow farmers to either utilize this information regarding raising and lowering the gates or manually change the height of the gates, two key features that are not provided by the Delta Plastics' *Pipe Planner*.

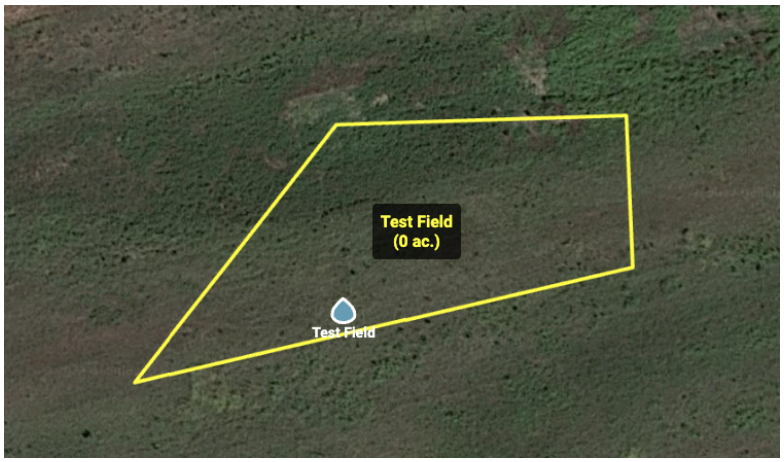

Figure 2: *Pipe Planner*'s Field Creation [16]

Other systems have been created that interface with devices out in the field similar to the GateMate design. A Mississippi State University irrigation specialist, Drew Gholson, utilized ultrasonic water level sensors, actuated values, and a pump controller to attempt a proof of concept in automating the process of irrigating fields [17]. His system allowed users to "remotely monitor water levels" and turn wells on or off from a smartphone or computer so that farmers would not have to drive out to specific fields to turn off wells when a certain threshold was reached [17]. This system was based solely on a 4-inch flood threshold level, but it would automatically turn off wells when that level was reached so that excess water would not be wasted. It would also allow farmers to check water levels and well status remotely. In testing, the system averaged 223 bushels per acre as opposed to 219 bushels per acre for farmer irrigated fields, and, in terms of water used, the automated fields averaged 18.8 acre-inches compared to the farmer-irrigated fields, which averaged 28.6 acre-inches [17]. While crop yield was not significantly increased, excess water usage was reduced, thus saving farmers money and time from not having to manually turn off the wells. While effective, this system still relied heavily on the farmer to make modifications based on incoming weather, their different topology, or desired threshold levels; GateMate, however, will account for weather, topology, and optimal AWD practices to almost entirely automate the process so that minimal effort is required for micromanaging field levels. The farmer would still have the capability to use their smartphone or computer to check and maintain water levels, but the system for checking and holding water levels will be much more dynamic than the Mississippi State product. The gate in this project would allow for precise modifications of how much water is being let into the field that would be automatically set based on various conditions as opposed to the farmer having options of only turning on or off the well that is allowing water to flow. Overall, the system to be designed will account for much more data, allowing the system to be truly automated, and have greater potential to save water and labor compared to the system designed by Gholson.

Another company, MimosaTEK, which is a Vietnamese start-up, developed technology that can be used to address some challenges that are faced by farmers when they attempt to implement alternate wetting and drying (AWD). With a smartphone, a farmer can monitor the actual and recommended water levels, which allows them to "determine the best time to irrigate the rice and the optimal amount of water to apply" [18]. Water pumps can be operated by this smartphone application with mobile or internet connectivity and electricity. This system runs "reliably with an uptime close to 100 percent," indicating that it would be likely to be trusted by farmers if they can accurately and precisely see water levels on their fields at any given time. Their results showed that utilizing IoT allowed for 13-20% reduced water usage compared to farmers using conventional AWD [18]. This system utilizes a smartphone and cloud-based management software, which will be similar to the GateMate software. An electrical engineering team has already installed hardware that utilizes Wi-Fi for connectivity, so this can be used to get information on water levels and modify the amount of water that is being let into the fields by the gates. However, the MimosaTEK system can only turn a water pump on or off rather than slightly modifying gates to allow certain amounts of water to flow to individual pools within the field, which is a key characteristic for the proposed GateMate device.

RiceAdvice, a mobile app by AfricaRice, provides "farm-specific advice on rice management practices" [19]. Guidelines are generated for farmers based off of a short interview, and it provides services such as identifying the best choice of fertilizer, determining when and how much fertilizer should be used, and allowing farmers to set their target yield levels. With the "personalized and profitable recommendations," users have increased paddy yield by 0.5-1 tons/hectare as well as increased profits by $100-200 per hectare per growing season [19]. Like RiceAdvice, GateMate will be mobile friendly, providing information to farmers in an easily accessible format. While RiceAdvice provides guidance similar to what GateMate intends to deliver, actions will automatically be taken based upon the guidance. In the case of GateMate, the gates will be moved up or down to modify water levels based on guidance from information on weather, topology, and best growing conditions rather than simply providing advice to the farmer to himself implement the prescribed course of action. This automation will allow farmers to easily achieve cost savings while reducing labor.

Another system developed out of a University in Thailand also uses a mobile application to help farmers with their rice farming. This application focused on providing basic information for farmers as well as information from a pH sensor and temperature sensor that would automatically monitor rice water quality [20]. This system has quite different functionality than the one that will be built, but both have the same underlying purpose of aiding rice farmers. The project developed by the University in Thailand communicated with the mobile application and a cloud database [20]. Other teams have already developed much of the GateMate's hardware, so this aspect of the project will be primarily focused on the mobile application portion and utilize the pre-built hardware to operate in a similar way to the device from the University of Thailand. The GateMate software will also contain a much broader scope, incorporating the recommendations and automations into a more advanced software that accounts for weather, topology, and best growth practices to most effectively make quality recommendations and automate the hardware accordingly based on the recommendations from the data.

A mobile app developed by ICAR-NRRI, or the Indian Council of Agricultural Research - National Rice Research Institute, called *riceXpert* allows farmers to see information in real time on "insect pests, nutrients, weeds, nematodes and disease-related problems" and more [21].

Farmers can use the application as a diagnostic tool in the field and find quick solutions to issues that arise by "sending text, photo and recorded voice" [21]. This app focuses more heavily on diagnosing issues after they arise instead of helping the farmers to take preventative measures. While *riceXpert* can be a powerful tool for farmers, providing an application such as GateMate can help prevent these issues from occurring, specifically when it comes to incorrect water levels affecting the production from the field. Both the *riceXpert* and the GateMate applications will provide farmers real-time information, but GateMate will allow the farmer to manually change the gate height to distribute water throughout the field or change the height automatically rather than just provide information to the farmer about solutions to improve their farming technique.

While there are various similar applications to aid rice farmers in their labor-intensive work, nothing quite compares to the GateMate device. The software for these gates allows for automated water level control based on data from weather information, best growth conditions for the plants, and the topology of the field. This automation will not only save farmers time in going out to manually control the gates but also save them money by eliminating excess water usage and ensuring prime conditions for growth to produce the maximum quantity from each field. The combination of multiple factors in this software will allow GateMate to be an effective aid to farmers more so than any of the aforementioned applications.

4.0 Approach and Design

4.1 Requirements, Use Cases, and Design Goals

Requirements

- Provide a detailed view of farm field map with topographical data
- Determine and identify the precise placements of GateMate to optimize water management
- Visually display all GateMate placements for future modifications
- Access weather information for the appropriate respective location so that GateMate can notify the user of potential water level changes
- Account for conditions of plant growth to distribute the water more effectively
- Wirelessly control GateMate devices from remote locations with any smart device
- Allow GateMate to automatically notify users of weather conditions to allow the user to manually modify gate heights appropriately
- Allow users to view water levels at each gate
- Be able to control GateMate devices by user-defined groupings

Use Cases

- Wireless control will allow farmers to control gate heights remotely, whether they be on site or otherwise.
- Determining the most effective placements for the gates in terms of water management to allow farmers to place gates only once without having to move them regularly to find the best placement.
- Utilizing weather information, crop growth conditions, and current water levels to determine optimal heights for the gates to remove guesswork for farmers seeking to optimize gate heights to minimize watershed.

- Allowing farmers to see water levels allows them to use real-time data to make decisions on raising or lowering gates in a holistic manner, whether they make adjustments remotely or manually.

Design Goals

- Modularize code on crop growth information and on positioning of gates so that specific modules can be further optimized by agricultural experts
- System should be resilient enough that if a gate is taken offline for maintenance the network of other gates remains perfectly functional
- Mobile application functionality should be powerful yet intuitive

4.2 High Level Architecture

The GateMate project includes various components that must all work together in unison to create an effective solution. The electrical hardware component has been completed, but the mechanical engineering design and software design are in progress of being developed. Specifically, the GateMate software must communicate with a user and the hardware that is controlling the device in the field. Due to the complex nature of the problem, the architecture of the software must be well designed to ensure the successful implementation of all required tasks.

The GateMate software will be developed using Flutter, allowing the software to be suitable for any device a user might own, including iOS, Android, or Windows devices. This software will interact with hardware built previously, including an MSP430 microcontroller and an ESP8266 Wi-Fi module, which will provide the interface for connecting the GateMate devices to the GateMate software. A high-level design implementation architecture highlighting the application design process is shown below in Figure 3. The overarching business logic involved with processing gate and geographical data to determine gate locations will be discussed first.

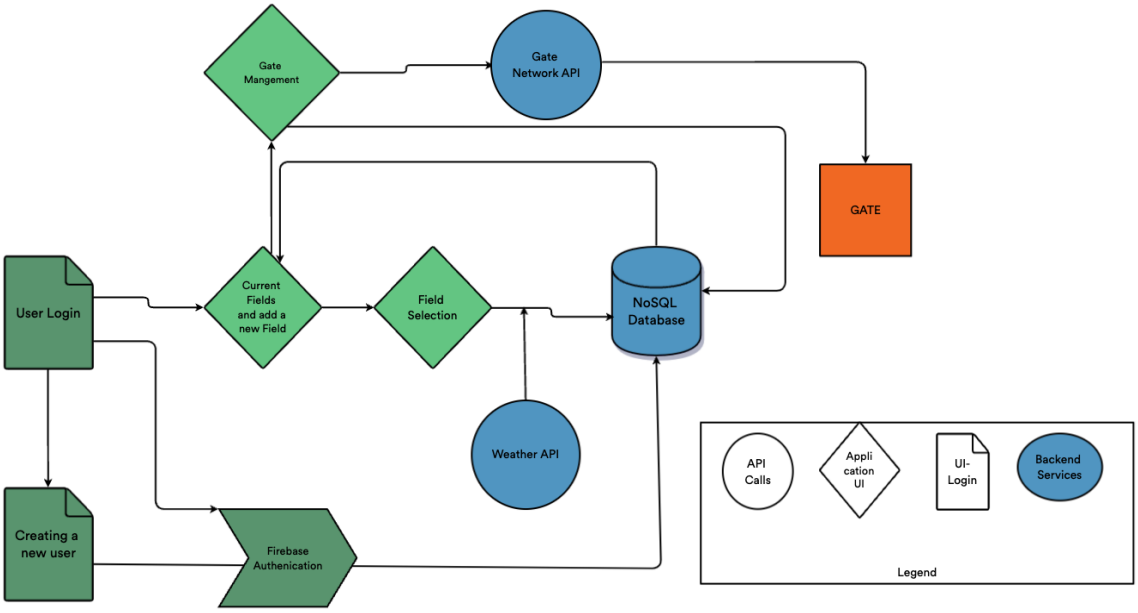

Figure 3: GateMate Application Architecture

A central server will retrieve gate-specific data for each individual gate. This includes data collected from the water sensor attached to the gate, the current height at which the gate is set, the location of the gate, and the elevation of the gate. The server will use this data to actively provide suggestions on appropriate gate heights. The necessary user information will also be stored on the central server (including secured authentication details, field boundaries, and levee locations). The server will poll the GateMate devices for updates, with the potential to detect possible gate failures in the future. The server will also periodically poll the Open-Meteo Weather Forecast API [25] in order to notify the user of relevant changes in the forecast. A robust weather and climate API is readily accessible through sources like Google Earth Engine, a geospatial processing service, which contains a petabyte-scale catalog of free-to-use geospatial datasets. This includes access to weather datasets containing forecast information regarding precipitation, temperature, humidity, and wind.

Furthermore, the server-side software will determine optimal gate placements in a field based on the topography of the land. The field will be designated by the user in the mobile application. Once a field is established and the gates are placed, the back-end will automatically make adjustments to gate heights and, when necessary, provide recommendations to the user in the form of displaying notifications using the mobile application. The process by which the server makes these decisions ought to be completely modular; great care will be taken to ensure there is room for further optimizations, such as machine learning algorithms, which can be implemented at a later time. The server will also contain endpoints for reading and writing gates and fields stored in the firebase database.

Establishing a connection between our application and the ESP8266 Wi-Fi module attached to the present GateMate prototype will be necessary to remotely control the gate from a smartphone. This implementation process will involve the use of both a Raspberry Pi and the ESP PainlessMesh library, which provides the basic software and hardware resources that simplify application development [20].

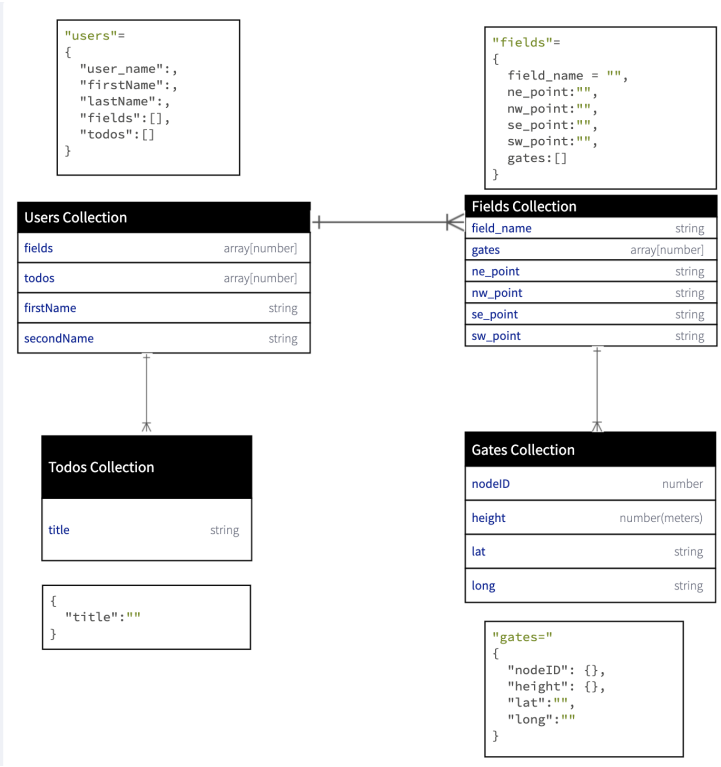

Figure 4: Proposed Database Schema

Figure 4 illustrates the GateMate database schema through Google's Firebase Firestore platform, which utilizes a NoSQL non-relational, document-based database. One of the advantages of using non-relational databases is their flexibility. They do not have a fixed structure, which leaves room for our project's needs to evolve. Document-based databases store data in flexible documents, where a document stores information about an object and stores its metadata in field value pairs. As such, three JSON document formats are illustrated above corresponding to the *Users*, *Fields*, and *Gates* tables above. It is important to note that these are tabular representations of firebase collections, where each entry within each collection corresponds to a JSON document entry. Tables are shown to illustrate the different properties of each of these documents within their respective collections and the relationship between these objects. The *Users* object contains fields for a user's first and last name, and a fields array that contains the ids of the fields created by the user. The *Users* object has a one-to-many relationship with the *Fields* object and, as such, *Fields* is added as an additional property for the *Users* document. The *Fields* object contains properties for the user id, field name, and a series of string objects that define latitude and longitude points. These four points correspond to the four vertices of a field boundary input by the user. For example, the "nw_point" corresponds to the northwest longitude and latitude location point of that particular vertex. This should be easily extensible to more complex polygons depending on development progress. The *Users* object also has a one-to-many relationship with *Todos*. Todos are defined by a set of notification alerts that would be sent to the user. This object only has a "title" property, which specifies the content of a particular notification alert. The *Fields* object also has a one-to-many relationship with the *Gates* object and has a gates array property that holds the ids of the gates within that field. The *Gates* object contains properties for its name, height, elevation, latitude, and longitude. These latitude and longitude properties correspond to the latitude and longitude location of a particular gate.

The ID of the user will be the same as the user ID generated by the front-end application's Firebase Authentication service. Firebase Authentication provides back-end services, easy to use SDKs, and prebuilt libraries to authenticate users to a particular application. Once a user signs up, a user's document will be created using the ID generated by the Firebase Authentication service. With this approach, neither the server nor the database will be responsible for handling user saved states with usernames and passwords. Firebase Authentication will take care of this in the background. It is important to note that when it comes to flexibility, NoSQL database schemas are much less costly to revise, which will be useful in the implementation process. The database will be hosted on Google's Firestore platform, with key capabilities including schema flexibility, real-time updates, and offline support.

As for the user interface (UI), it should be intuitive and easy to use while providing the broad range of functionality illustrated in the following figures. Ideally, the map present in the UI would be able to generate a topographical view of the area. With a navigation drawer, the user will have various options to change to another view relating to information about the selected field.

The navigation drawer will allow users to open pages allowing them to view notifications in the action center, add a gate to a selected field, view information about the gates in the field, and access settings (Figure 5). The "Action Center" tab will show scheduled gate changes and relevant weather alerts as well as allow users to add new actions for which they want to receive notifications. The "Add Gate" tab allows users to add a gate to their field by entering latitude and longitude values. The "Gate Management" tab allows users to view the suggested gate locations for a field on a topographical map. The "Manage Multiple Gates" tab allows a user to control the heights of multiple gates at once rather than having to do this task for each individual gate. Upon clicking a marker, the user can view the positioning of the gate in the field and change the height of the gate by entering a specific height. The "Settings" tab allows users to view existing fields or add a new field by navigating to another page that prompts users for four coordinates representing the corners of the field. It also allows the user to configure ideal water levels for best growth conditions based on the specific crop variety so that the height of the gates will be modified accordingly for the respective field. This tab also allows users to select how often they would like to receive notifications.

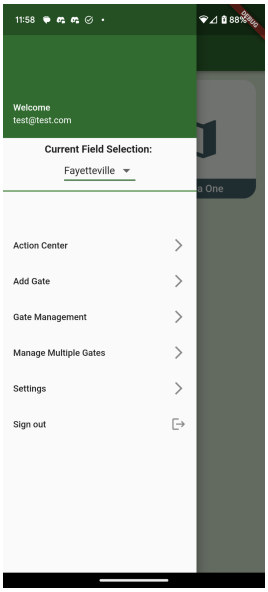

Figure 5: Navigation Drawer

After logging in, users will see the home screen, which displays the fields they have created and allows them to select one (Figure 6). Upon creating an account or logging into an existing user account with no associated fields, a user will land on an empty home screen and must add fields to their account to interact with gates on these fields.

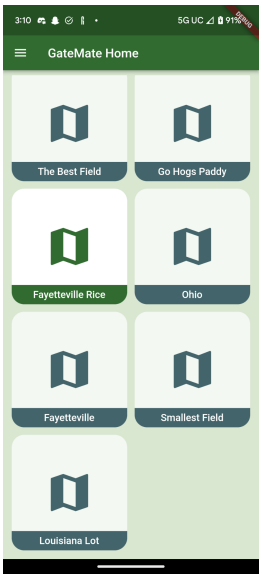

Figure 6: Home Page

By navigating to the settings page using the navigation drawer, a user can add a field and corresponding gates. They will be able to enter specific coordinates for the boundary points of the field, as seen in Figure 7. A potential add-on feature can include the ability for the user to upload a shapefile, which is a geospatial vector data format for geographic information software, instead of manually inputting the latitude and longitude of the corners of the fields themselves.

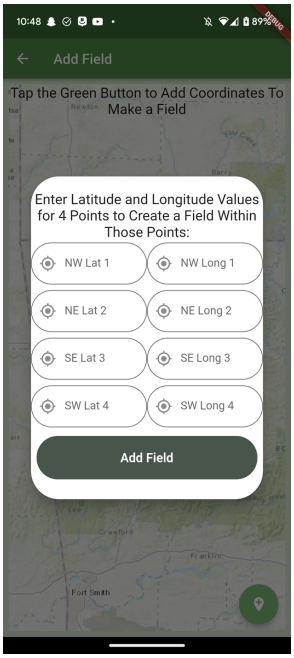

Figure 7: Manually Entering Coordinates for Field

Once the coordinates are selected, the back-end server calculates optimal approximate placements, and the user is shown a colored grid indicating where the field changes in height. This process is done by normalizing the elevations across the entire field to the lowest point and dividing the field into sections of close elevations. Since rice typically has a tolerance of water levels of approximately 2-3 inches, these sections are allowed to vary in elevation by the same amount. The colored tiles rendered on the screen represent these different elevation zones. This allows the user to have control over placements based on their specific field requirements while providing a guide for optimal placements. On this page, a user can tap on the map and add gates to the field that are saved to the database for the respective field. These gates' heights and specific location can be modified later by the user. This user interface is shown in Figure 8 below.

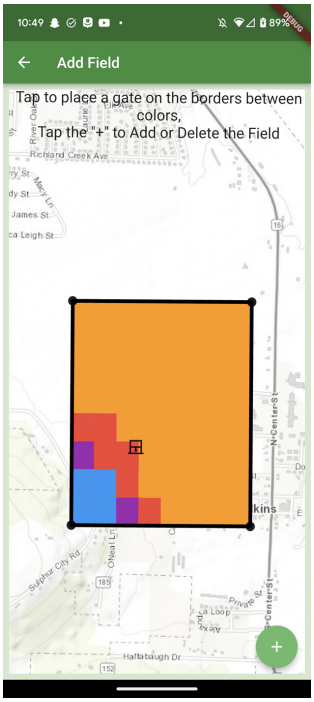

Figure 8: Add Field View

Once the user has selected their field and confirmed that the gate locations are acceptable, this information will be saved to their account. The user can immediately see the addition to their saved fields. Upon clicking on a field name, the user will be able to see the field and gate locations on a topographical map, along with an option to update the gate location, update the gate height, or remove the gate, as shown in Figure 9.

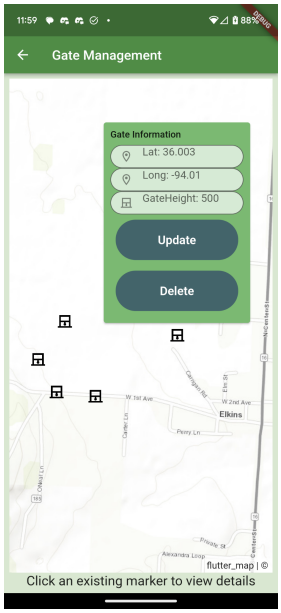

Figure 9: Gate Management View

A feature suggested by our sponsors was having the ability to control multiple gates. This functionality is available to users on the “Manage Multiple Gates” page, where they can select multiple gates and modify the heights of said gates all at once. A confirmation dialog ensures the user is selecting the desired gates before changing the heights to help reduce errors. This page and dialog are seen in Figure 10.

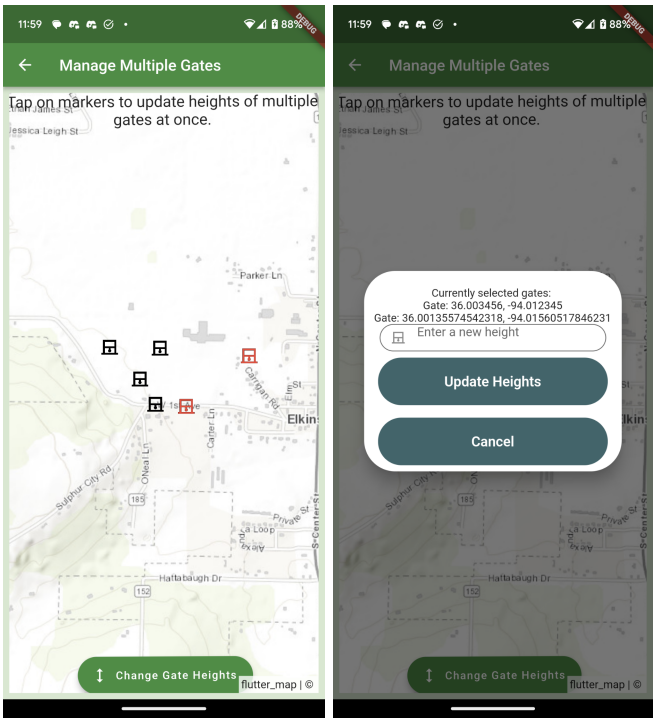

Figure 10: Manage Multiple Gates View

A possible stretch goal for the project would be displaying the paths of water flow throughout the rice paddy based on the topography of the field and the layout of the levees and gates. This information would be a useful informational tool for users and could be added as an option via a floating action button on the Gate Management screen. If a user would prefer to take a more hands-on approach to gate management, this insight could prove to be crucial to making informed decisions. An example UI for this feature is shown in Figure 11 below.

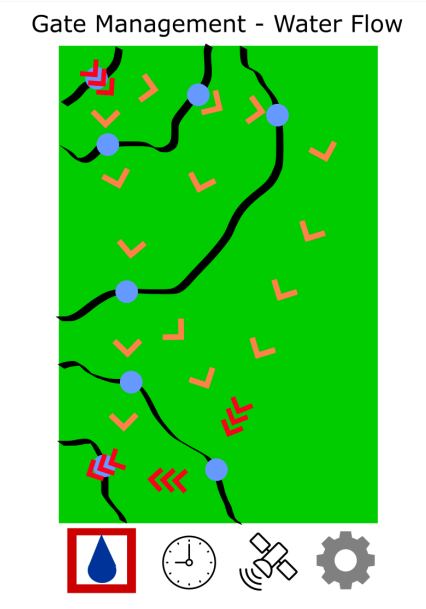

Figure 11: Water Flow Behavior UI

Figure 12 depicts the scheduled actions/notifications page, also known as the "Action Center". This page allows a user to view all notifications regarding gate status. These notifications are populated in the database by the application server. Unfortunately, time constraints precluded the integration of error checking with the mechanical engineering team's prototype. The floating action button shown in Figure 12 allows the user to add reminders to their action center. In the future, when scheduled gate changes are supported by the back-end, this would allow the user to schedule gate height changes for the future. Finally, the settings page (shown in Figure 13) allows the user to set desired water heights, define new fields or switch to others, and manage notification settings.

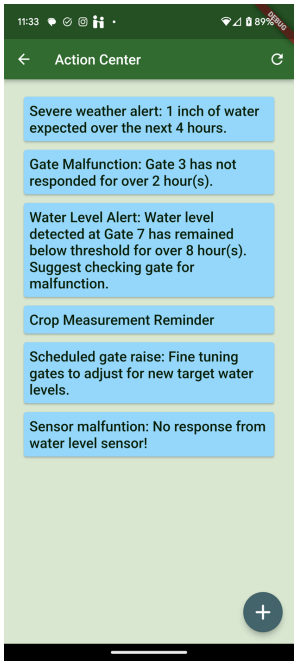

Figure 12: Action Center UI

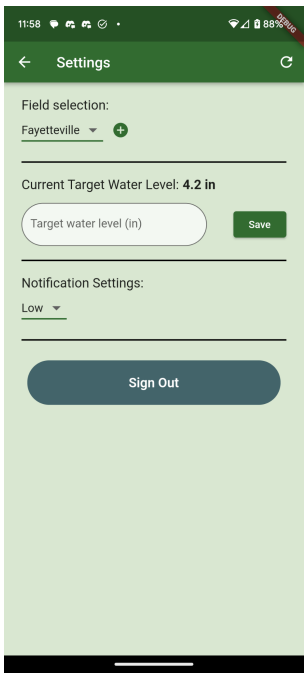

Figure 13: Application Settings UI

The design of the mobile application's architecture follows the Model-View-Viewmodel (MVVM) approach. This design pattern is characterized by subscribers (view components or, in Flutter's case, widgets) that listen for changes in the application's state and respond with their own changes [23]. This is particularly well-suited for event-driven applications. The application's viewmodels are implemented using Flutter's *ChangeNotifiers*. Every time the data handled by these viewmodels is modified, the *ChangeNotifiers* notify subscribe widgets. Flutter then rebuilds each affected widget using the new data.

The application instantiates each viewmodel as a singleton. To accomplish this, we use the GetIt package [22]. Every viewmodel is registered with GetIt at app startup; most are registered lazily and will therefore not be instantiated until they are actually needed. Widgets then attach listeners to data they want to observe after accessing an instance of the relevant viewmodel using GetIt. This approach precludes redundant data storage (all in one place as singletons), reduces the number of network calls needed to view remote data (data persistence), and provides a formal, unified interface for additional use cases as the application grows.

Weather events such as significant rainfall are important for rice farmers. Not only could these negatively affect the water levels of the paddy (potentially damaging crops), but rainfall presents an opportunity to further conserve water resources. For this reason, the GateMate mobile application will notify users of significant rainfall in the forecast. The application server exposes an endpoint that fetches weather data from the Open-Meteo API. The mobile client application periodically queries this endpoint, which provides forecasted rainfall in one-hour increments for the next three days. The client then, using a push notification, alerts the user of the first hour for which there is an amount of rain forecasted to be greater than half of an inch, although future work could allow the user to configure this threshold. Furthermore, a process to detect sustained rainfall that, per hour, remains under the threshold could be implemented in the

future. Because this process needs to occur in the background while the user is potentially away from their phone, we utilize the workmanager package [24]. This package allows the app to schedule a recurring background task that is executed even when the user's phone is not in use. The task is currently configured to execute every three hours, but future improvements could allow the user to configure this frequency. It is also configured to retry three times upon failure, starting with a five-minute delay and doubling upon each failure. Future work could notify the user of persistent failures using push notifications. Finally, one background task is scheduled for each field tied to the user. If these tasks were to get out of sync with one another, this approach could lead to increased battery consumption due to potentially waking the user's device multiple times. Instead, one task could be scheduled, which would iterate through multiple fields.

4.3 Risks

Risk	Risk Reduction
Circuit on gate depletes battery too fast	Optimize code to allow the processor to sleep or stay in a low-power mode as often as possible
Lose connection to a gate	Field Module checks the mesh network on intervals and prints a warning if a potential disconnect is detected. Future work would include adding functionality such that this warning could be sent to mobile devices too.
Farmer places gates in location other than suggested	Allow user to manually select where the gate was placed as opposed to the suggested location
Map information not updated for recent topography changes	Allow the user to confirm the map matches the field topography. Future work would allow for the user to update topographical features.
Sensors stop working (break, become blocked, disconnected)	Have the software on the gates detect when a sensor might be broken. Have the gates then send a notification indicating a possible problem

4.4 Tasks

1. Learn about MSP430 Development and what methods exist for connecting an MSP430 to a mobile device for controlling the gates
2. Confirm with Mr. White that the application design will complete all tasks required
3. Confirm software is compatible with and accounts for all requirements from the Mechanical Engineering device design
4. Rigorously define the data to be stored in the database
5. Complete final proposal document
6. Complete final proposal slides
7. Design a central server that accomplishes the following:

- **7.1** Storing the database
- **7.2** Facilitating communication between mobile application and GateMate devices
- **7.3** Autonomously monitoring and managing GateMate devices

8. Create a server-side application that communicates with GateMate devices through their microprocessor's API
9. Create basic software for MSP430 that reacts to signals from server
10. Test ability to connect software to MSP430 to get sensor information
11. Create logic that will get current water levels from the sensors on the device
12. Create basic user interface for gate management
13. Create basic user interface for home screen
14. Create basic user interface for settings
15. Create basic user interface for managing multiple gates
16. Create basic user interface for users to see and select their saved fields
17. Create basic user interface for user login
18. Create module for logic to determine gate placement
19. Test module to determine gate placement
20. Create a service to connect user interface interactions with the server
21. Connect logic for gate placement to UI to display gates to user
22. Test UI to ensure correct information is pulled from logic module
23. Create module for logic to determine gate height
24. Test module for logic determining gate height
25. Connect logic for gate height to UI so user can see information
26. Test UI to ensure correct information is pulled from logic module
27. Work with ME team to ensure information pulled from device is accurate
28. Create UI for users to create a field area on the map by entering four coordinates
29. Create logic to store field location information for the user
30. Create functionality for users to select and update multiple gates
31. Create back-end logic that can update the height information in the database

32. Create functionality for users to manually raise/lower gates
33. Test logic to raise/lower gates
34. Create functionality for users to manually place gates
35. Test logic to manually place gates
36. Ensure manually placed gates can be updated to new locations and saved
37. Handle disconnect from gate network
38. Document final results and expected operation

4.5 **Schedule**

Tasks	Tentative Dates	Status
Learn about MSP430 Development and what methods exist for connecting an MSP430 to a mobile device for controlling the gates	10/24-11/7	Complete
Confirm with Mr. White that the application design will complete all tasks required	10/24-11/25	Complete
Confirm software is compatible with and accounts for all requirements from the Mechanical Engineering device design	10/24-11/25	Complete
Rigorously define the data to be stored in the database	11/15-11/21	Complete
Complete final proposal document	11/15-11/22	Complete
Complete final proposal slides	11/15-11/22	Complete
Design a central server	11/21-12/9	Complete
Create a server-side application that communicates with GateMate devices through their microprocessor’s API	1/15-1/21	Complete
Create basic software for MSP430 that reacts to signals from server	1/15-1/21	Complete
Create logic that will get current water levels from the sensors on the device	1/22-1/28	Awaiting MEEG*

Test ability to connect software to MSP430 to get sensor information	1/22-1/28	Awaiting MEEG*
Create basic user interface for gate management	1/22-1/28	Complete
Create basic user interface for home screen	1/22-4/23	Complete
Create basic user interface for settings	1/22-1/28	Complete
Create basic user interface for managing multiple gates	1/22-1/28	Complete
Create basic user interface for users to see and select their saved fields	1/22-2/12	Complete
Create basic user interface for user login	1/29-2/4	Complete
Create module for logic to determine gate placement	1/29-2/4	Complete
Test module to determine gate placement	2/5-2/11	Complete
Create a service to connect user interface interactions with the server	2/5-2/11	Complete
Connect logic for gate placement to UI to display gates to user	2/5-2/11	Complete
Test UI to ensure correct information is pulled from logic module	2/12-2/18	Complete
Work with ME team to ensure information pulled from device is accurate	2/12-2/18	Awaiting MEEG*
Create module for logic to determine gate height	2/12-2/18	Awaiting MEEG*
Create back-end logic that can update the height information in the database	2/19-2/25	Complete
Test module for logic determining gate height	2/19-2/25	Awaiting MEEG*
Create a service to connect user interface interactions with the server	2/19-2/25	Complete
Connect logic for gate placement to UI to display gates to user	2/19-2/25	Complete

Test UI to ensure correct information is pulled from logic module	2/25-3/4	Complete
Create module for logic to determine gate height based on sensor information	2/25-3/4	Awaiting MEEG*
Test module for logic determining gate height based on sensor information	2/25-3/4	Awaiting MEEG*
Connect logic for gate height to UI so user can see information	2/26-3/4	Complete
Test UI to ensure correct information is pulled from logic module	2/26-3/4	Complete
Work with ME team to ensure information pulled from device is accurate	3/5 - 3/11	Awaiting MEEG*
Create UI for users to create a field area on the map by entering four coordinates	3/5 -3/11	Complete
Create logic to store field location information for the user	3/5 -3/15	Complete
Create functionality for users to select multiple gates	3/13-3/18	Complete
Create back-end logic that can update the height information in the database	3/13 - 3/20	Complete
Create functionality for users to manually raise/lower gates	3/13 - 3/25	Complete
Test logic to raise/lower gates	3/15 - 3/21	Complete
Create functionality for users to manually place gates	3/13 - 4/18	Complete
Test logic to manually place gates	4/13 - 4/18	Complete
Ensure manually placed gates can be updated to new locations and saved	4/20-4/24	Complete
Handle disconnect from gate network	4/20 - 4/24	Complete
Document final results and expected operation	4/20 - 4/24	Complete

***** *The tasks denoted with "Awaiting MEEG" are tasks that were not able to be completed by the time this report was submitted due to time constraints with the MEEG team's integration testing. These tasks may*

be done before final presentations assuming all of the Mechanical Engineering team's testing goes according to plan.

4.6 **Deliverables**

- Design and usage documentation: contains a list of the major software and hardware components utilized along with how the software interacts with the mechanical engineering team's physical GateMate prototype that will be built concurrently with the software.
- Documentation for deployment of elevation API and swapping the geological survey data currently included.
- Flutter code: the software used to tell the server to raise and lower gates, including the code for the various user interfaces and logic modules.
- Database schema and sample data: the data used for the logic modules.
- Field Module code: code running on a Raspberry Pi that acts as an in-between for the back-end and gate network.
- ESP code: code that runs on the circuitry that directly controls the gate. This software is responsible for all functionalities present at an individual gate. This includes mesh network communication, processing of commands from the field module, and raise/lower controls to be integrated with the MSP430 whenever the gate is ready.
- Final report: report detailing the completed work on the GateMate software and how it fulfilled the required objectives.

5.0 Key Personnel

Students

Jackson Bullard - Bullard is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed courses in big data analytics, computer networks, mobile computing, database management systems, and cryptography. He has conducted research using distributed machine learning techniques and privacy-preserving technologies. Additionally, he has designed techniques for computation using the algorithmic self-assembly of natural systems. He will be responsible for developing the user-facing mobile application.

Nathaniel Fredricks - Fredricks is a senior Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has worked two summers as a web development intern at J.B. Hunt and one summer as a custom development intern at Affirma that. He has conducted research related to computer architecture and array processors. Fredricks will be responsible for developing software for MSP430 to raise/lower gates and communication between the back-end and gates.

Jose Martinez - Martinez is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed coursework in software development and database management. Martinez has relevant work experience at the University of Arkansas as a Research Lab Assistant for the AI and Computer Vision Lab and

as a Teaching Assistant for the Programming Paradigms Course. Martinez also completed a Software Engineer Internship with Oracle in Summer 2022, successfully gaining experience in back-end microservices and database work. Martinez will be responsible for implementing the necessary back-end database services work for connecting the mobile application and the middle back-end handler for handling communication between the server and the underlying database being implemented.

Carissa Patton - Patton is a senior Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas. She has completed courses including Software Engineering and Wearable and Ubiquitous Computing which will provide a solid background for the software development part of the application. She has also included courses such as Low Power Digital Systems and Circuits and Electronics which allows for understanding of hardware components of the overall systems. Additionally, Patton has completed a Software Development Internship at ArcBest in Summer 2021 and an IT internship at Phillips 66 in Summer 2022. Patton will be responsible for assisting in the design and implementation of communication between the gates and the server, as well as designing the front-end user interface.

Ivris Raymond - Raymond is a senior Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas. She has completed relevant coursework in Software Engineering, Embedded System Security, and Low Power Digital Systems. She has also conducted research related to embedded and IoT systems and completed an internship at Phillips 66 in summer 2022. Raymond will be responsible for assisting in designing and implementing the intercommunication between the gates and the server. Raymond will also be designing the software for the back-end that will run on the server acting as the messenger between the mobile application and gates.

Project Sponsors

Jason Bailey - Bailey is the Senior Design Project Coordinator for the Mechanical Engineering Department at the University of Arkansas. He has been the consultant for the mechanical engineering of the gate, especially with regards to the non-computational hardware responsible for controlling the gate, including sensors, servos, and more.

Matthew Patitz - Patitz is an Associate Professor in the Computer Science & Computer Engineering Department at the University of Arkansas. He will help with scope declaration, help in resolving any issues that might arise with communication with other sponsors, and will provide guidance on any challenges that arise in the implementation of software if necessary.

Timothy White - White is an agriculturist and conservationist who proposed the idea of this GateMate project. From his experience with extensive labor in an agricultural context relating to regulating water for fields, he proposed GateMate as a way to decrease the labor required and conserve water by removing the guesswork on when to regulate water levels. White has been the expert relating to anything regarding farm or plant specific questions, as well as providing guidance as to what features the final software application should include.

6.0 Facilities and Equipment

The Mechanical Engineering team will concurrently create a GateMate device that will interact with the proposed created software. They work in the mechanical engineering building on campus, and close collaboration between the two teams will be necessary to ensure holistic success. An electrical engineering team from a previous semester has created hardware that will raise and lower the gates of the mechanical engineering design, and our proposed project will interact with this hardware. Specifically, this hardware is an MSP430 microcontroller and an ESP8266 Wi-Fi module, both of which are further discussed in section 3.1.

The CSCE team has also added a Field Control Module that connects to the gate mesh network and the internet. This module needs to be on the field, but only close enough to be able to connect to a gate node. For the work done on this project, this device was modeled with a Raspberry Pi 4B. Any embedded microcontroller with Wi-Fi should be capable of serving the purpose of this device provided a Python implementation is available for its Instruction Set Architecture (ISA).

7.0 References

[1] The Importance of Rice, http://www.knowledgebank.irri.org/ericeproduction/Importance_of_Rice.htm

[2] Arkansas Rice Facts, https://www.arfb.com/pages/arkansas-agriculture/commodity-corner/rice/

[3] Total U.S. Rice Production Value from 2000 to 2021, https://www.statista.com/statistics/190474/total-us-rice-production-value-from-2000/

[4] Hardke, Jarrod. "Arkansas Rice Production Handbook." Little Rock, Ark: Cooperative Extension Service, University of Arkansas, 2013.

[5] Sustainable Agriculture and Water, https://www.nature.org/en-us/about-us/who-we-are/how-we-work/finance-investing/naturevest/sustainable-agriculture-water/

[6] Water Use by Rice, https://rice.ucanr.edu/Water_Use_by_Rice/

[7] Using Alternate Wetting & Drying (AWD) Rice Flood Management, https://www.uaex.uada.edu/publications/pdf/FSA62.pdf

[8] Google, Earth Engine Data Catalog, https://developers.google.com/earth-engine/datasets

[9] "What is a mesh network?" Google Nest Help. https://support.google.com/googlenest/answer/7182746?hl=en

[10] "What is an API?" University of California San Francisco, n.d. https://developer.ucsf.edu/what-api.

[11] "Wi-Fi — ESP8266 RTOS SDK Programming Guide." espressif. n.d. https://docs.espressif.com/projects/esp8266-rtos-sdk/en/latest/api-reference/wifi/esp_wifi.html.

[12] "Flutter - Build apps for any screen." Flutter. n.d. https://flutter.dev/.

[13] "What is a database?" University System of Georgia. n.d. https://www.usg.edu/galileo/skills/unit04/primer04_01.phtml#:~:text=A%20database%20is%20a%20collection%20of%20information%20organized%20to%20provide%20efficient%20retrieval.

[14] "What is SQL?" IBM. March 3, 2021. https://www.ibm.com/docs/en/developer-for-zos/14.0.0?topic=support-what-is-sql.

[15] "The Power of Automation in Rice Production." Columbia Okura. Columbia Okura, n.d. https://columbiaokura.com/thought-leadership/power-automation-rice-production/.

[16] Revolution. "Pipe Planner." Delta Plastics Home, n.d. https://www.deltaplastics.com/pipe-planner.

[17] Boyd, Vicky, "Siri, Turn Off Pump: Research Examines Potential Water Savings of Automated and Remote Irrigation Systems," Rice Farming, March 10, 2021. https://www.ricefarming.com/departments/feature/siri-turn-off-pump/.

[18] Choudhary, Vikas, and Karin Fock, "Precision Agriculture for Smallholder Farmers in Vietnam: How the Internet of Things Helps Smallholder Paddy Farmers Use Water More Efficiently." World Bank Blogs, April 23, 2020. https://blogs.worldbank.org/eastasiapacific/precision-agriculture-smallholder-farmers-vietnam-how-internet-things-helps.

[19] "RiceAdvice." AfricaRice. The Consortium of International Agricultural Research Centers, n.d. https://www.africarice.org/riceadvice.

[20] Kularbphettong, Kunyanuth, Wannaporn Phoso, and Pattarapan Roonrakwi. "The Automation of Mobile Application to Manage the Rice Fields." TEM Journal: Technology, Education, Management, Informatics 8, no. 3 (August 2019): 866–71. https://www.temjournal.com/content/83/TEMJournalAugust2019_866_871.pdf

[21] "ICAR-NRRI Developed Mobile App 'RiceXpert.'" National Rice Research Institute. ICAR-National Rice Research Institute , n.d. https://icar-nrri.in/icar-nrri-developed-mobile-app-ricexpert/.

[22] "Get_it: Dart Package." Dart Packages, 13 July 2021, https://pub.dev/packages/get_it.

[23] Davidbritch. "The Model-View-Viewmodel Pattern - Xamarin." Xamarin | Microsoft Learn, https://learn.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm.

[24] "workmanager: Dart Package." Dart Packages, 20 Oct 2022, https://pub.dev/packages/workmanager.

[25] "Open-Meteo Forecast API." Open-Meteo Documentation, 4 Apr 2023, https://open-meteo.com/en/docs