HO CHI MINH CITY NATIONAL UNIVERSITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE & ENGINEERING

MULTIDISCIPLINARY PROJECT

Grassquitto

Greenhouse Automation System

Instructor:	Prof. Mai Đức Trung	
Students:	Ôn Quân An	1852221
	Huỳnh Thiện Khiêm	1852154
	Bùi Minh Kiệt	1852115
	Nguyễn Hoàng Long	1852164

HO CHI MINH CITY, MAY 2021

Contents

Document History

Date	Ver.	Changes	Changed by
06/04/2021	1.0	- Create report template - Initialize sections and functional requirements description	Huynh Thien Khiem
06/04/2021	1.1	- Write topic introduction and illustration	Bui Minh Kiet
06/04/2021	1.2	- List out planned devices - Describe input device's intended usage	On Quan An
07/04/2021	1.3	- Describe output device's intended usage - Add non-functional requirements description	Bui Minh Kiet
07/04/2021	1.4	- Report initial work distribution	Nguyen Hoang Long
08/04/2021	1.5	- Review and Add details for device description	Huynh Thien Khiem
12/04/2021	2.0	- Initialize use-case templates - Describe use-case for Multi-mode irrigation	Huynh Thien Khiem
13/04/2021	2.1	- Describe use-case for System reports and statistics - Describe use-case for Customisable periodic reminders	On Quan An
13/04/2021	2.2	- Describe use-case for Temperature regulation and Light level regulation	Bui Minh Kiet
14/04/2021	2.3	- Add use-case diagram	Nguyen Hoang Long
20/04/2021	3.0	- Initialize mockup report template	Huynh Thien Khiem
22/04/2021	3.1	- Add mockup for Home screen	Huynh Thien Khiem
22/04/2021	3.2	- Add mockup for Statistics Logs screen - Add mockup for Environmental setting screen	Bui Minh Kiet
22/04/2021	3.3	- Add mockup for Reminders screen (Main and Add new)	On Quan An
22/04/2021	3.4	- Add mockup for Statistics Graph screen	Huynh Thien Khiem
23/04/2021	3.5	- Review mockup and organize screen flow	Nguyen Hoang Long
24/04/2021	3.6	- Add mockup for Login screen	Huynh Thien Khiem
02/05/2021	4.0	- Initialize report template for week 5	Nguyen Hoang Long
03/05/2021	4.1	- Add activity diagram: subsystem functionality	Huynh Thien Khiem
04/05/2021	4.2	- Add activity diagram: Reminder - Add activity diagram: Statistics	On Quan An
04/05/2021	4.3	- Add database overview	Bui Minh Kiet
05/05/2021	4.4	- Add deployment view	Bui Minh Kiet
05/05/2021	4.5	- Review diagrams and explanation	Huynh Thien Khiem
11/05/2021	5.0	- Revise use-case description for current changes	Huynh Thien Khiem
11/05/2021	5.1	- Revise EERD diagram	Nguyen Hoang Long
11/05/2021	5.2	- Customise document history	Bui Minh Kiet
12/06/2021	6.0	- Update document to current development status	Huynh Thien Khiem
17/06/2021	6.1	- Update app demo and manual	Huynh Thien Khiem
20/06/2021	6.2	- Revise requirements and use cases	Bui Minh Kiet
21/06/2021	6.3	- Redo deployment and activity diagrams	Huynh Thien Khiem Bui Minh Kiet

1 Introduction to the topic

Greenhouses are used to regulate climatic conditions for plants that are sensitive or not native to the local natural climate. As external variables such as sunlight, wind and rain vastly affects the greenhouse in unpredictable ways, keeping a greenhouse in the right conditions has always been a labour-intensive task.
As such, modern commercial greenhouses are high-tech production facilities with equipment such as automatic motorised sunscreens, artificial climate control, lighting, etc. and may be controlled by a computer to optimise conditions for plant growth. Different techniques are used to estimate how optimal a greenhouse's conditions are, such as air temperature, humidity and internal pressure changes due to vapours.
For this project, our aim is to develop a prototype of such a system, utilising advances in Internet and wireless connectivity to enhance convenience and flexibility.

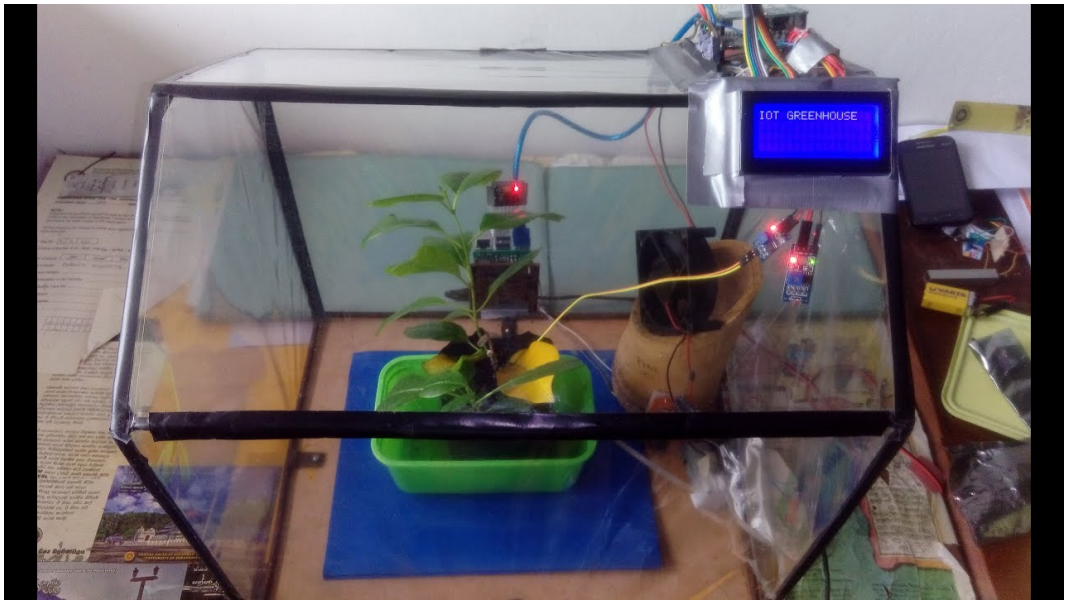

Figure 1: A sample model of an IoT-enabled greenhouse.

2 Requirements

2.1 Functional

2.1.1 Internet-of-Things aspect

- Automatically irrigate greenhouse plants according to current soil moisture and plant type.
- Maintain ideal climatic conditions including temperature, venting, and lighting via different means: manually, automatically or periodically by schedule.
- Report issues that need manual intervention, such as inability to maintain conditions within threshold due to extreme external conditions.

- Provide live climate reports to the user via some form of display (graph view and log view).
- Periodically remind user to tend to manual tasks, such as checking water supply and cleaning windows.

2.1.2 Mobile application aspect

- Allow manual user control of specific greenhouse functionality, like plant watering, light switching and alarmed services.
- Provide current climate measurements as well as historical readings.
- Statistical analysis and reports on the system's condition and performance.

2.2 Non-functional

2.2.1 Internet-of-Things aspect

- Run 24/7 with minimal scheduled downtime and preferably no unscheduled downtime.
- Device handling packaged in one or more MQTT server(s).
- Actuator delay time should be no more than five seconds.
- Persistently log historical readings for diagnostic purposes that keep tracks of the system's status over at least 180 days.
- Scalable and efficient database system that has a capacity of at least 500MB.
- Simple user manuals and interactions via LCD screen or device buttons.

2.2.2 Mobile application aspect

- Available and usable for every age, which particularly targets farmers, gardeners, and agricultural managers.
- Legible UI under all weather conditions.
- UI must also be easily operable with gloves.
- Run on Android.
- Connection with server should take under 2 seconds while basic interface interaction should take under 500ms.
- Support local data storage of up to 100MB as well as on database server.
- User shall be able to authenticate via a securely-stored password and/or patterns.

3 Devices

3.1 DHT11

- **Application:** Measure temperature and report to the IoT server (its atmospheric humidity data is not used in our system). Temperature is a vital statistic for many greenhouse maintenance operations, including, but not limited to, whether to irrigate or not, how much to irrigate, whether to deploy sunscreens, etc.
- **Input:** Air temperature - the sensor is mounted above-ground, preferably suspended under the volumetric centre of the greenhouse.
- **Output:** Temperature value, reported to the controller, which relays to the IoT server.

3.2 Soil moisture sensor

- **Application:** Measures water level within the soil. This is the statistic that most directly affects irrigation decisions.
- **Input:** Soil moisture - the sensor is planted underground.
- **Output:** Local soil moisture, reported to the server as above.

3.3 Light sensor

- **Application:** Measure sunshine intensity and detect daylight. Sunshine intensity also affects whether to irrigate or not. Daylight detection on the other hand dictate whether to turn *grow lights* on or off.
- **Input:** Light level - the sensor is placed outside, in exposure to the surrounding environment settings.
- **Output:** Processed by the controller to return sunshine level and additionally a *grow light* control signal when the light level dips below a set threshold.

3.4 Relay circuit

- **Application:** Used to control the mini-pump.
- **Input:** On/off (controlled by the irrigation logic).
- **Output:** None (electronically). Lets power through on *On* inputs.
- **Operation:** Switches the pump's electrical circuit.

3.5 Mini-pump

- **Application:** Pumps water to irrigate the greenhouse.
- **Input:** None (runs as long as there is electrical power).
- **Output:** None (electronically). Moves fluids.

3.6 RC servo 590

- **Application:** Deploy and retract sunscreens, if equipped. The sunscreen is assumed to be either a foldable fan-like structure that rotates 180 degrees, or a hinged fabric wall.
- **Input:** Angle data from MQTT.
- **Output:** None (electronically). Opens and closes the sunscreen mechanism.

3.7 Single 2-colour LED

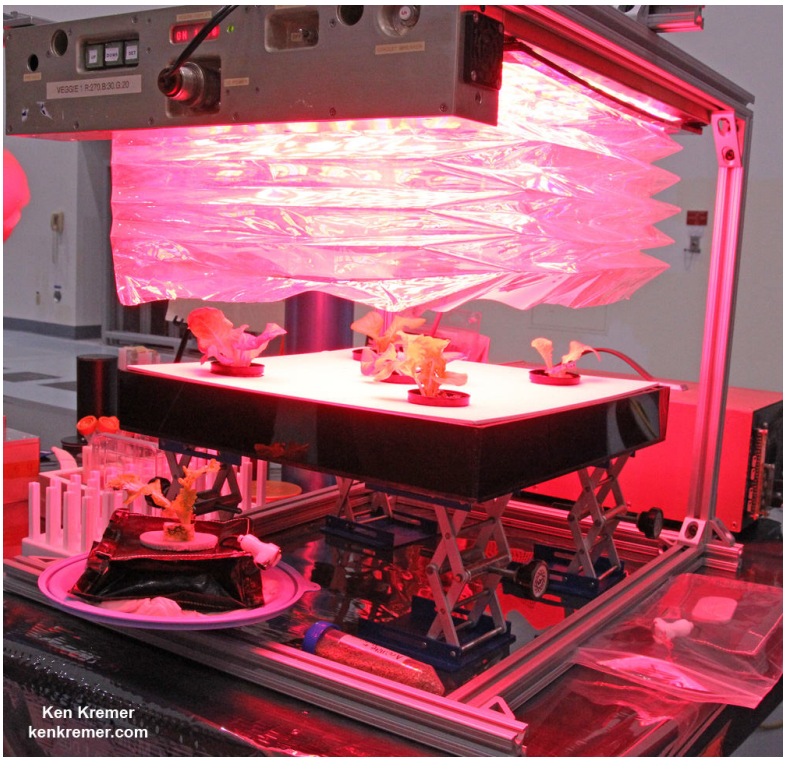

Figure 2: Example of an extremely power-efficient grow lamps setup on the International Space Station. Far-red wavelengths are vital to plants, the others less so [Dem+16].

- **Application:** Serves as a mimic for real grow lamps. In our system, it emits red light.
- **Input:** 0-2 as control signal. 0 is off, 1 is red, 2 is blue. Here we will only input either 0 or 1.
- **Output:** None (electronically). Emits red light.

3.8 Microbit

- The hub for all sensors. It receives sensor readings through the expansion circuit board and perform preliminary processing (such as unit conversion, threshold and the like).

- Pre-processed data are then sent to the server for logging and broadcasting to all smartphones currently connected to it.

3.9 Expansion circuit board

The backbone for Microbit and other sensors.

3.10 Adapter 5V

This is a continuously-operated and stationary system. As such, a stable continuous power supply is preferred over batteries.

4 Use case details

4.1 Use case 1: Multi-mode irrigation

Use case ID	1
Use case name	Multi-mode irrigation
Actor	User, server, and devices (moisture sensor, pump)
Description	The system provides three modes of irrigation - a fully automatic mode in which the system takes care of maintaining a preset level of soil moisture, a scheduled mode that follows a user-specified daily, weekly or monthly schedule, and a fully manual mode that allows the user to explicitly start and stop the irrigation. In addition, a fallback mechanism can effect a stronger correction if measurements fall into the danger zone while in any of the three modes, if enabled. All modes provide moisture level warnings and correction notifications to the user.
Preconditions	The system is running normally, and is equipped with at least a pump, a soil moisture sensor, and a water source, preferably one that does not need manual refilling.
Normal flow	The expected course of actions would be for the user to delegate the irrigation to the system. 1. User opens the mobile application. 2. User selects ”Irrigation” from the list of features on the main screen. 3. User selects ”Automatic” in the Mode section. 4. User specifies the target moisture level, according to an on-screen table of recommended values or a slider for custom values. 5. System evaluate every incoming measurement, taking action if necessary. That is, If the moisture level is below that of the target level by 50%, it irrigates for 5 seconds.

Exceptions	*At step 5, if the system detect abnormal changes (or a lack of change) in soil moisture while irrigating:* (a) System does not detect slower than expected change in water moisture while irrigating, or no detected changes. (b) System sends a push notification to the smartphone application, warning that there are problems with the water source (exhausted or clogged) or the pump. (c) If the fallback mechanism is enabled, system triggers a stronger action (a 20-second pump).
Alternative flows	*At step 3, if the user chooses irrigation to be scheduled:* (3.1) User specifies whether the system should automatically react to extreme conditions while in this mode, just warn the user, or simply ignore and stick to the schedule. (3.2) User chooses to add a new task. (3.3) User selects the length of the schedule (within one day, one week or one month). The corresponding form pops up. (3.4) User specifies when to irrigate within that schedule and how much water to dispense (i.e. how long to pump for this task). (3.5) System irrigates on that schedule's basis. In case automatic actions are enabled, system monitors soil moisture every time it receives new measurement and might raise warnings to the user through notifcations. *At step 3, if the user chooses irrigation to be fully manual:* (3.1) User specifies whether the system should warn of dangerous conditions, such as extremely low or high moisture levels. (3.2) If warnings are enabled, system monitors soil moisture as regularly as possible and may warn via push notifications. *Details regarding the application's user interface may change as development proceeds.*
Postcondition	Soil moisture data is logged as often as the current plan checks soil moisture. For example, if scheduled irrigation was enabled without also enabling automatic actions, soil moisture will only be logged upon scheduled checks. Also stored is a record of all instances of irrigation, and information regarding the events that triggered said irrigation instances, such as scheduled triggers, automatic actions responding to environmental conditions, or manual control.

4.2 Use case 2: System reports and statistics

Use case ID	2
Use case name	System report and statistics
Actor	User
Description	The application allows user to view recorded data in the form of summary report (over a time period) with aggregate calculations and graph overview of past recordings.
Preconditions	The mobile app is connected to the database and the system is still actively recording statistics.
Normal flow	1. User opens the mobile application 2. User selects ”Statistics” from the list of features on the main screen. 3. System extracts the daily data as the default report mode, and present to the user. The report includes line graphs on light level, moisture level, and temperature level during the time period; notable actions taken manually or automatically on the greenhouse system; and latest greenhouse’s status (last received measurements). 4. User may choose to select the preferred time period by choosing ”Time period” on the option bar. 5. System provides a detailed calendar for user to choose the starting and end date for report. 6. User selects his/her preferred time period and presses ”Confirm”. 7. System reloads the report page with proper graph view and action tracking of the specified time period. 8. User presses ”Done” to return to the main page.
Postconditions	None
Exceptions	*Exception 1: at step 4, the time period that user specified is not in the database system.* (5.1) System shows Error window and propose user to choose an appropriate time span.
Alternative flow	*At step 3, if the user chooses ”Statistics to report” on the option bar:* (3.1) System presents a list of data for recording (by default, every option is checked) for user to choose what to report. (3.2) User checks/unchecks items accordingly and presses ”Confirm”. (3.3) System saves the configuration, and only reports the specified items next time.

4.3 Use case 3: Temperature regulation

Use case ID	3
Use case name	Temperature regulation
Actor	User, server, and devices (DHT11, RC servo)
Description	With temperature control, the system also provides 3 options: an automatic mode where the system tries to maintain the temperature of the greenhouse at an ideal level, a scheduled mode that follows a user-specified daily, weekly or monthly schedule (with or without a safety fallback to automatic mode), and a fully manual mode that allows the user to explicitly open or close the sunscreens, which are opaque ceiling-mounted covers (here we use the motor to simulate a rolling sunscreen), whenever the user wish to. If the user chooses to go with the scheduled mode, it is preferred that a safety fallback is always available. Storms or unpredictable weather fluctuations may unmitigable and usually violent fluctuations in greenhouse temperature. The manual mode provides optional temperature warnings to the user. The examples above may change as research and development proceeds.
Preconditions	The system is running normally, and is equipped with a the DHT11 sensor, optionally with sunscreens connected to actuators(for deploying and retracting).
Normal flow	*The expected course of actions would be for the user to delegate the temperature control to the system.* 1. User opens the mobile application. 2. User selects ”Temperature control” from the list of features on the main screen. 3. User selects ”Scheduled mode”. 4. User sets safety fallback mode to either ”do nothing”, ”warn” or ”warn and take action”. 5. System provides a range-based slider representing 24 hours of a day. 6. User selects his/her preferred time period when the sunscreens are open or closed. *Details regarding the application's user interface may change as development proceeds.*
Postconditions	Temperature (measured by the sensor) and sunscreen condition is logged on a periodical basis into the system's database. Furthermore, actions on moderating the temperature level (either by user or system) is also tracked and kept in the history log.

Alternative flow	*At step 3, if the user chooses "Automatic":* (3.1) User specifies the target temperature range. (3.2) System starts checking temperature level every time new data is received. For example, if the temperature range is below that of the target level by a sufficiently large margin (from 3 to 5 degrees Celsius), it opens the sunscreens until the sensor reports a level within that margin. *At step 3, if the user chooses "Manual":* (3.1) User specifies whether the system should warn of dangerous temperatures. (3.2) If warnings are enabled, system monitors temperature regularly and may warn via local notifications. *Details regarding the application's user interface may change as development proceeds.*

4.4 Use case 4: Light level regulation

Use case ID	4
Use case name	Light level regulation
Actor	User, server, and devices (traffic light, light sensor, two-colour LED)
Description	Similar to temperature control, we can moderate the light level in the greenhouse with 3 options: automatically, with proper configuration on how system should react to environmental settings; on scheduled, usually based on time period in day and/ or days in week; or manually, where user can control the grow lamps directly. Sunscreens may be advisable in case the greenhouse requires an upper limit to light level during daytime. The system may also warn user on extreme cases such as broken lighting equipment or unfavourable light levels that are outside of the system's regulation capabilities.
Preconditions	The system is running normally, and is always equipped with a light sensor and a relay circuit attached to the plant lighting source.

Normal flow	*The expected course of actions would be for the user to delegate the light control to the system.* 1. User opens the mobile application. 2. User selects ”Light level regulation” from the list of features on the main screen. 3. User selects ”Scheduled mode”. 4. User sets safety fallback mode to either ”do nothing”, ”warn” or ”warn and take action”. 5. System provides a range-based slider representing 24 hours of a day. 6. User selects his/her preferred daily time period during which the grow lamps are on. *Details regarding the application’s user interface may change as development proceeds.*
Postconditions	Light level (measured by the sensor) and lighting configuration is logged on a half-hourly basis into the system’s database. Furthermore, actions on moderating the light level (by either user or system) is also tracked and kept in the history log.
Alternative flow	*At step 3, if the user chooses ”Automatic”:* (3.1) User specifies the target light level range. (3.2) System starts checking light level every unit of time every time new data is received. If the light level does not fit in the configured target range, the system will act until the sensor records suitable measurements accordingly. *At step 3, if the user chooses ”Manual”:* (3.1) User specifies whether the system should warn of exceptions, like low light level or potential broken lighting system. (3.2) If warnings are enabled, system monitors light level regularly and may warn via local notifications. *Details regarding the application’s user interface may change as development proceeds.*

4.5 Use case 5: Customisable periodic reminders

Use case ID	5
Use case name	Customisable periodic user reminders
Actor	User and server

Description	Offer various reminder functionality to keep user in a scheduled check routine as well as proper awareness of the greenhouse system. The default reminder system notifies user of garden checking twice daily at 6A.M and 19P.M. May be customised to remind on different to-dos and status update.
Preconditions	The mobile app is running normally and the time system is functioning properly
Normal flow	1. User opens the mobile application. 2. User selects ”Reminder settings” from the list of features on the main screen. 3. The app redirects to a list of current reminders, each detailing their time of going off, the content to remind of, and whether it is active or not. 4. User chooses ”Add new reminder” on the option bar. 5. System presents a form of creating a new reminder. The form includes title, optional description, and scheduled time. 6. User completes the form and presses ”Confirm”. 7. System adds the new reminder into the list, and will trigger the notification accordingly at the time scheduled.
Exceptions	*Exception 1: at step 6, user presses ”Confirm” without having filled in the title and/or time scheduled.* (6.1) System automatically fills with current time, ”Untitled” for title, and ”No description given” for description.

4.6 Use case Diagram

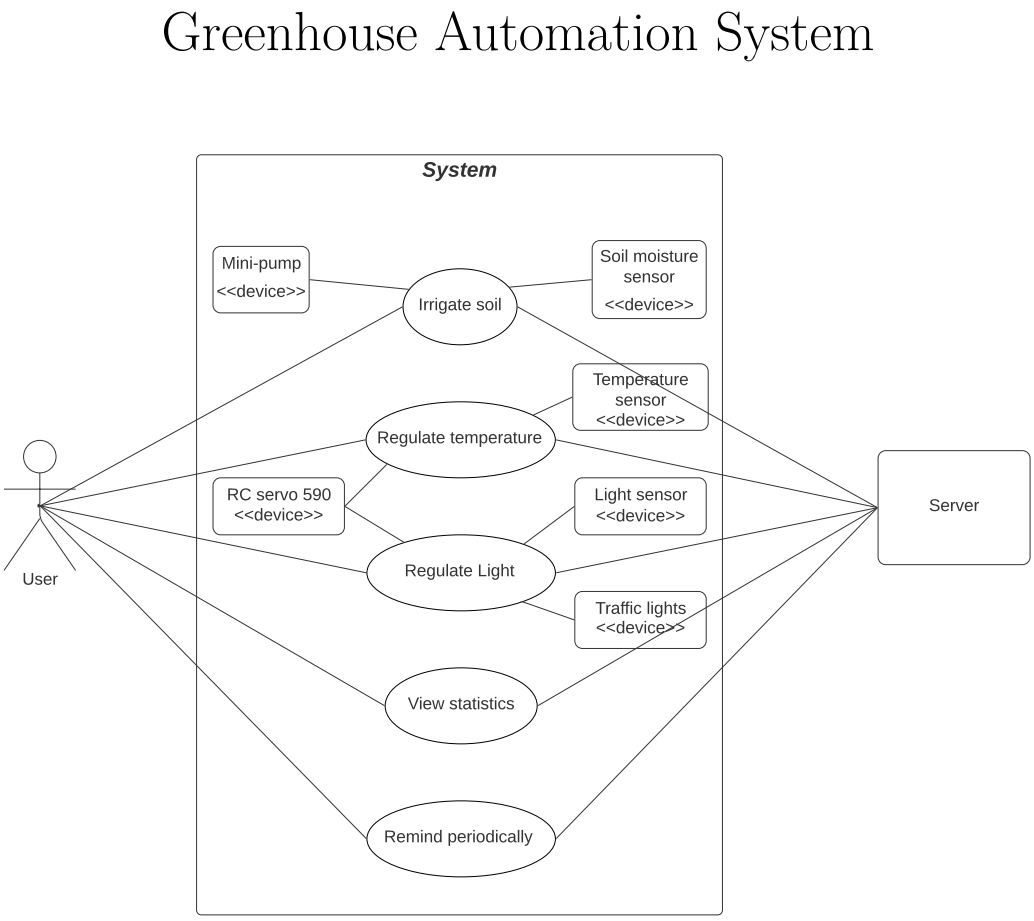

Figure 3: Use case Diagram

5 Work distribution

- **On Quan An:** Reminders UI/backend, scheduling UI.
- **Huynh Thien Khiem:** UI head, scheduling logic and other assorted backends.
- **Bui Minh Kiet:** Backend head, SQLite database design/implementation and MQTT.
- **Nguyen Hoang Long:** Statistics export logic, administrative tasks.

6 General design

Here we use the term "server" and "broker" interchangeably.

6.1 Authentication

To enable flexible MQTT configurations, we propose a local-profile design in which greenhouses are given locally-stored *profiles*. Each profile contains server authentication data (of which there might be more than one server) and device-server pairings. This design enables all possible combinations of server-device distributions, such as X devices on server A, Y devices on server B, etc. There are at most **six** servers, which in that case, would be arranged as one for each device that we use.
Each profile is self-contained. All gathered sensor readings and activity records are stored per-profile. This has multiple benefits, such as added partitioning and security in the case that the mobile device (Android smartphone, tablet, smart fridge or Chromebook) is shared by multiple family members for their own greenhouses.

6.2 Business logic

Based on the use-case specification, we decided to split the business logic (the workings that concern the greenhouse) into three *subsystems*: Irrigation, Lighting and Temperature.

- The Irrigation subsystem reads from the soil moisture sensor and controls the pump's relay accordingly.
- The Lighting subsystem reads from the light sensor and switches grow lamps (the 2-colour LED) on and off.
- The Temperature subsystem reads from DHT11 and controls the servo connected to the foldable sunscreen.

Subsystems share the same MQTT and database backends. The MQTT backend takes care of all connections to MQTT servers (concurrently) and abstracts this from the rest of the logic by exposing a simple per-device subscription *data streams* and publish functions, i.e. the irrigation subsystem upon receiving data from the moisture sensor's data stream (from the backend) will act by calling the publish function of the backend, specifying which device and what data to send.

7 Mockups

Note: Some of these mockups no longer reflect our current design and are only kept for archival and comparative purposes.

7.1 Login and Registration screen

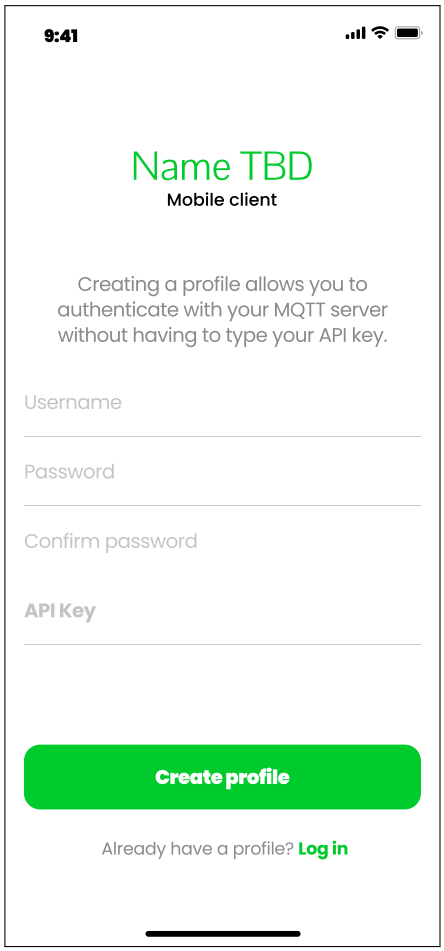

Figure 4: The registration screen. User is prompted to enter their local profile username, preferred password, confirmed password, and the actual API key that is used to login to MQTT server

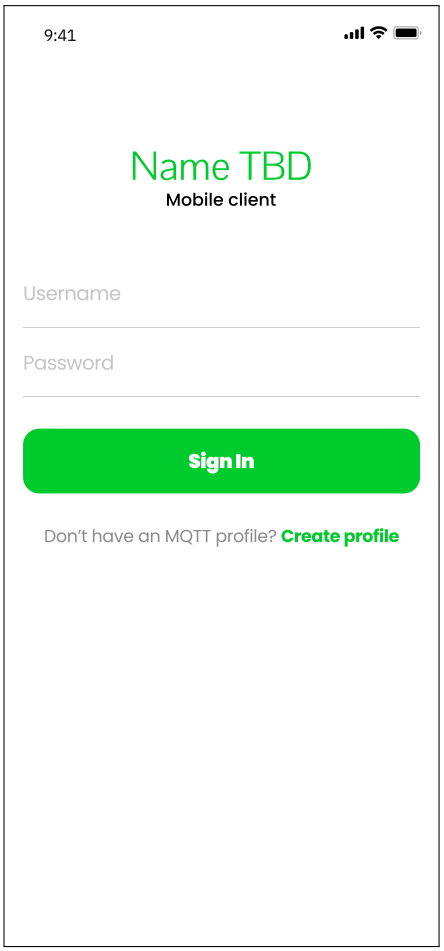

Figure 5: The login screen. User need to enter Adafruit server username and their created password. Upon logging in, the username token will be used to connect to Adafruit server and other online services.

The login and registration provides basic security and persistence option for user. The username and API key used must be a legitimate Adafruit credentials, since we intend to make it consistent with the IoT server for convenience. However, user can used their preferred password (which is more easily memorized than the API key), which is used as a second authentication factor in the mobile app. This password can be text that is at least 7 characters long, and is going to be hashed and stored in the database for confidentiality.

7.2 Home screen

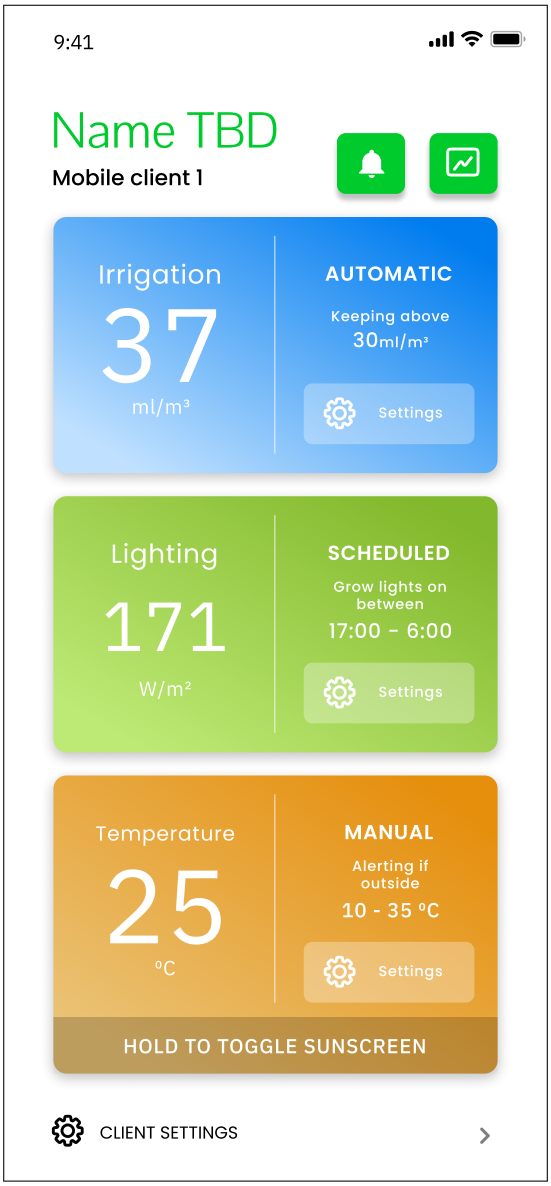

Figure 6: The home screen. In this example, the irrigation subsystem is on full automatic mode, the light level regulation subsystem is scheduled to switch on grow lights at night, and the temperature regulation subsystem is set to be manually controlled.

We propose a minimal, visual-mnemonic, gesture-centric design to serve as our home screen. The three main regulation functions of our system - irrigation, light regulation and temperature regulation) are each given a large card, coloured semantically (blue represents water, green represents chlorophyll, and orange represents heating) and containing very large readings of instantaneous measurements. The cards intuitively informs the viewer within one glance the current status of the greenhouse as well as which actions needs to, can be, or will be taken right now and in the future.

The interactivity of this screen is mostly gesture-based in a normal workflow: tap on a card (*not necessarily within the Settings button*) to open the settings screen for that subsystem; hold a card to trigger the corresponding manual action (if that subsystem is in manual mode). Gestures allow one widget to serve multiple functions, and the reduction in the number of widgets on screen at a given moment allows each widget to be larger and more legible under typical working conditions (such as out in the field under direct sunlight).

In case the device has a shorter aspect ratio, this screen could be scrolled vertically. However, we expect most modern smartphones in all market segments to have at least a 18:9 aspect ratio, which would fit our entire screen without scrolling.

Also in this screen is a link to the client settings page (which allows the user to adjust the address of the MQTT server and the related client-side settings), and a button to the statistics screen.

7.3 Statistics screen

This screen can be reached via the corresponding from the home screen, and is dedicated to the displaying of past sensor readings and logs of actions taken by the system or the user.
It consists of two tabs - one for graphs and one for logged events. The temporal range of displayed data can be adjusted with the so-labelled button, which pops up some form of widget that allows selecting a range of days.

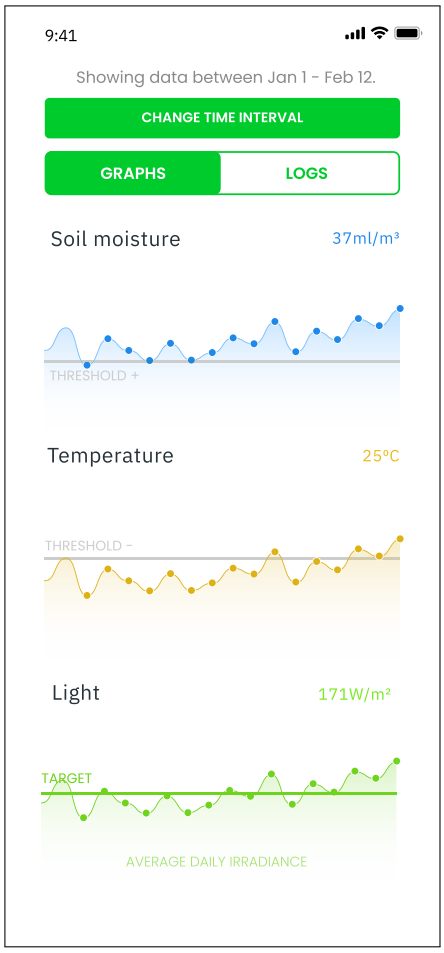

Figure 7: The statistics screen, showing its graphs. Each subsystem's graph carries their indicative colour schemes, allowing for visual intuition without needing to read the titles.

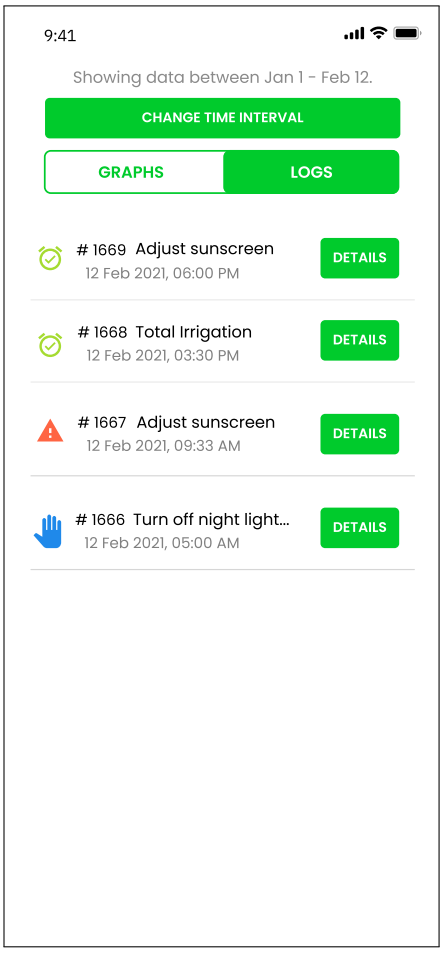

Figure 8: The statistics screen, showing its logs.

The graphs tab is currently designed to contain one graph for each subsystem, coloured in that subsystem's colour scheme so as to be visually intuitive, i.e. allowing the user to instantly identify which subsystem a graph belongs to just by glancing. Normal textual titles are still provided.
The graphs presented here may not represent our final vision of how the application should look like. Ideally, it should also contains tick marks, grids and/or other visual guides to help

better interpret the graphs. The logs screen also shows data within the specified range. Each logged event is displayed in the list with a leftmost icon denoting type (scheduled, warning, and manual events are depicted in this mock-up, in that order), a log index, a brief description, a full timestamp and a button to display detailed information in a pop-up. The topmost is the latest.

7.4 Environmental regulation settings screens

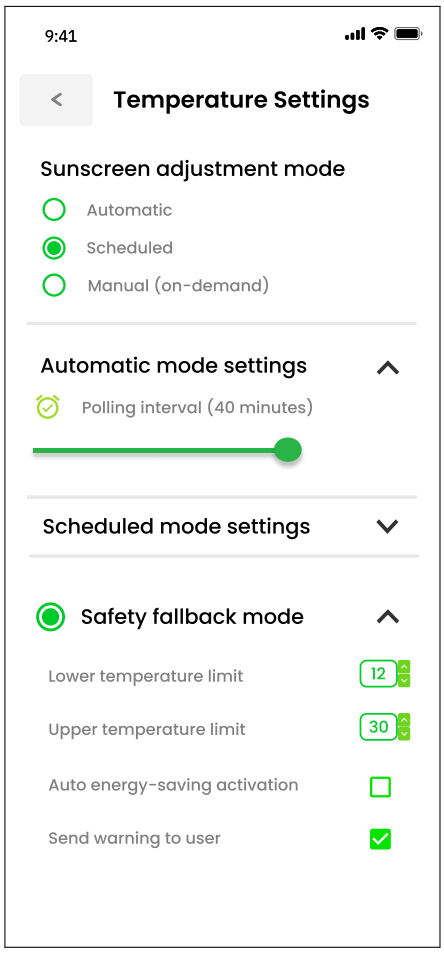

Figure 9: The temperature screens, showing its current customisation and available options. Screens for light and moisture are mostly similar

This section shows the intended mock-up for the use case relating to the regulation of environmental variables, such as temperature, light, and moisture. Since all of these screens are basically similar (except the wording and some specific constraint on each device), we will demonstrate the working mock-up for temperature screen as an example.

These screens can either be navigated to by clicking on the respective environmental status dashboard on the main screen, or by using the drawer options. User shall see a setting screen with customisable options to choose how they want to regulate the greenhouse system.

There are three modes available to regulate the temperature sunscreen adjustment:

1. **Manual**: this allows user to manually choose when to adjust the sunscreen, as well as how they want to adjust it. If this option is chosen, the main screen dashboard will offer a manual sunscreen adjustment button for user to control and regulate the temperature themselves.

2. **Automatic**: this customises the system to regulate the sunscreen automatically on a time-based basis. If this is chosen, the *Automatic mode settings* below shall be adjusted to user's will. User could drag and modify the polling interval (the time span between each temperature regulation). Note that if user does not choose *automatic*, this customisation will be ignored.

3. **Scheduled**: this allows user to schedule a specific sets of time to adjust the sunscreen. If this is chosen, the *Scheduled mode settings* shall be adjusted to user's will (see the **Alarm screen** for more detail). Note that if user does not choose *scheduled*, this customisation will be ignored.

Except while in the fully automatic mode, there will be another setting for **Safety fallback**, which could be turned on to perform troubleshooting and warning notification when

the temperature goes out of limit range. System would then adjust the sunscreen as well as send user some warning indicating the incidents. Moreover, depending on the environmental variables, other safety detection can be set: energy-overload for light sensor, water source depletion for moisture, and so on.

7.5 Reminders

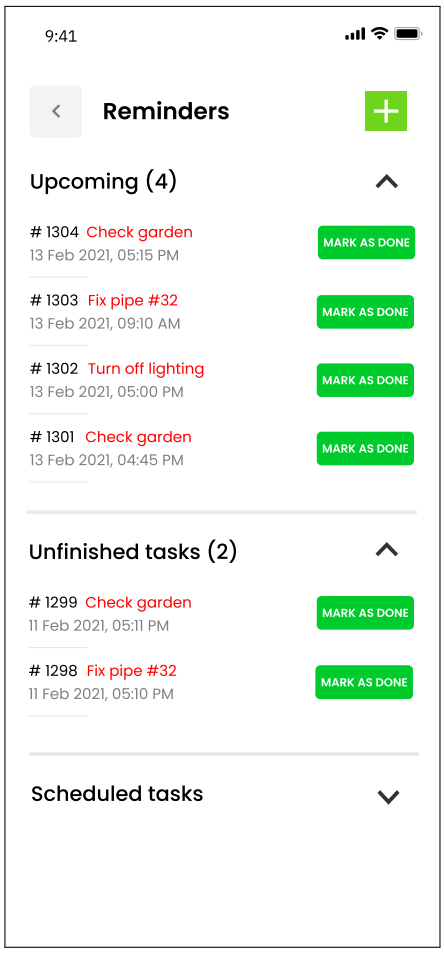

Figure 10: The reminders screen.

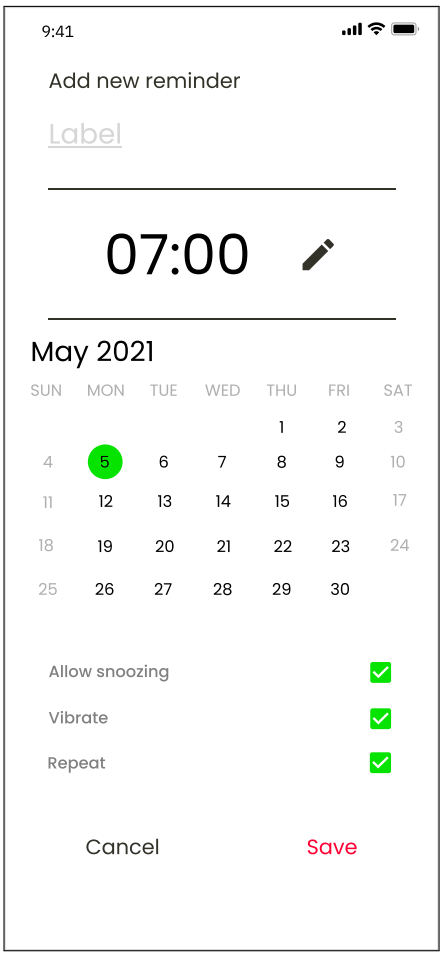

Figure 11: How adding a new reminder looks like.

The Reminders function's main purpose is to help user check what they should do in that day. It could be reached via the corresponding button on the home screen. It comes with an alarm that notifies user at preset day and time. On the main screen, there are divided in 3 sub tasks so user can remember and prepare themselves:

- **Upcoming**: This section tells user what they will need to finish in near future so they can arrange time, prepare tools. This will help a lot in scheduling time for user.

- **Unfinished tasks**: This displays tasks that user should finish on the given day. Each task has a label to describe what should be done and at what time. Then user can tap ”Mark as done” after he has finished it.
- **Scheduled tasks**:

When new tasks appear and need to be done soon, user can use this feature to push a notification on the day and time they set. For this feature, it can be divided into 3 part:

1. **Labelling**: It is the title of the reminder. User can edit it such that when it shows notification, he will remember what need to be done. For this label, it should be short, descriptive text.
2. **Set time and date**: The app use 24-hours model, which is very familiar in eastern countries. User choose preferred time and date in the calendar.
3. **Choose options**: These options help user to set the notifications as their will. It has option to allow vibration or even snoozing.

8 Technical diagrams

8.1 Deployment View

This section portrays our system as a package diagram. The illustrated deployment is presented here, involving two main modules: **System** and **Server / Internet**.

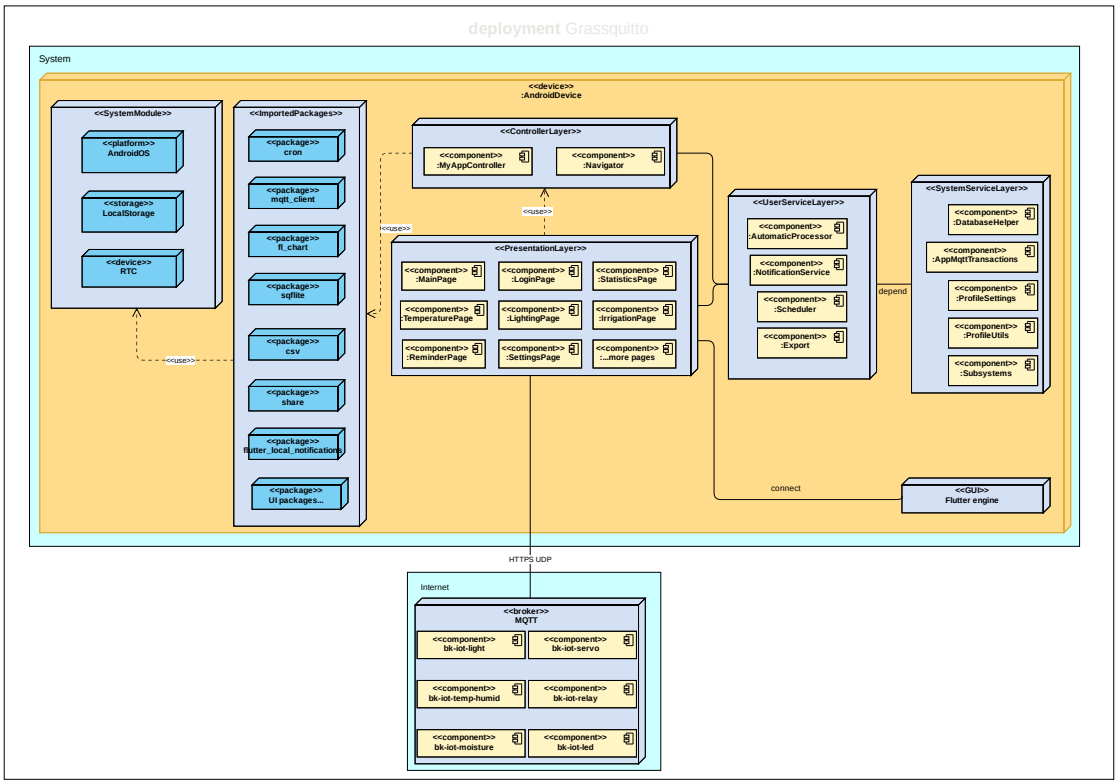

Figure 12: Deployment view of our system

In the System module, we deploy all packages inside our Android Device, which is expected to be a Flutter mobile app project. First and foremost, the project uses available **System Modules** such as the Android Operating System, Local Storage, Real-Time Clock, etc. These are integral for the Flutter framework and its related **Imported Packages**. We have utilized a number of Flutter packages, most importantly including utility ones like *cron* for scheduling tasks, *mqtt_client* for handling MQTT connection, *fl_chart* for scaling data into graphs, *sqflite* for Database management, *csv* for local file export, *flutter_local_notifications* for mobile app reminders and warnings. Furthermore, there are also various different UI libraries to improve user experience, notably *day_night_time_picker, animations, google_fonts, flutter_circular_slider*.

Now, for GRASS app, the central logic controller will reside in the **ControllerLayer**, which includes the main App Controller and the Screen Navigator. This layer depends on two other implementation layer, which are:

- **UserServiceLayer**, which contains back-end utilities, such as notification service, scheduling, file export and automatic data processing. Note that this layer depends on the services provided by a deeper **SystemServiceLayer** that handles transactions in the background,

such as the Database Helper, the MQTT Transactions, the Profile Settings / Utilities and the environment-specific Subsystems.

- **PresentationLayer**, which provides front-end UI/UX states of the app widgets. Moreover, the Presentation Layer is indeed supported by the device's screen, which forms the actual User Interface and Experience in deployment. The main screens are as listed in the use cases, which are the main page, the login/registration page, the statistics/log page, the environment-setting pages (temperature, lighting, irrigation), the reminder page and the settings page. There are also auxiliary pages and supporting transition screens that form the basic presentation of the app

,

These system modules can communicate and control the online servers we intend to implement, which is indeed the **MQTT** broker on Adafruit. The MQTT Transactions will connect to necessary feeds located on the server, which as detailed by our device listing will consist of 6 major input/output feeds:

- *bk-iot-temp-humid*
- *bk-iot-soil*
- *bk-iot-light*
- *bk-iot-servo*
- *bk-iot-relay*
- *bk-iot-led*

8.2 Implementation View

The implementation view of our system will be illustrated under several activity diagrams, each demonstrating a certain use-case or groups of use-case we have presented a few weeks before.

8.2.1 Activity Diagram: Subsystem functionality

As described in the use-case document, our system consists of three subsystems: irrigation, temperature regulation and light level regulation. Although they perform different tasks, their workflow and behaviour are similar. As such, we have opted to group the diagrams for these subsystems into one for the sake of brevity.

Our system is fully implemented in Dart, which has a sometimes-asynchronous model of execution. For ease of understanding, we have separated all possible execution threads into their own swim lanes, although they in practice might not have a full thread (they might instead be concurrent functions via preemptive multitasking).

The smartphone application is designed to be run headless most of the time (in the background, without the UI). There are two ways of exiting the application: by tapping Back on the home screen, or by tapping Return anywhere. Tapping Return simply exits the UI and allows the application to continue running in the background. Tapping the Back button indicates that the application is to end monitoring and log out of the current session.

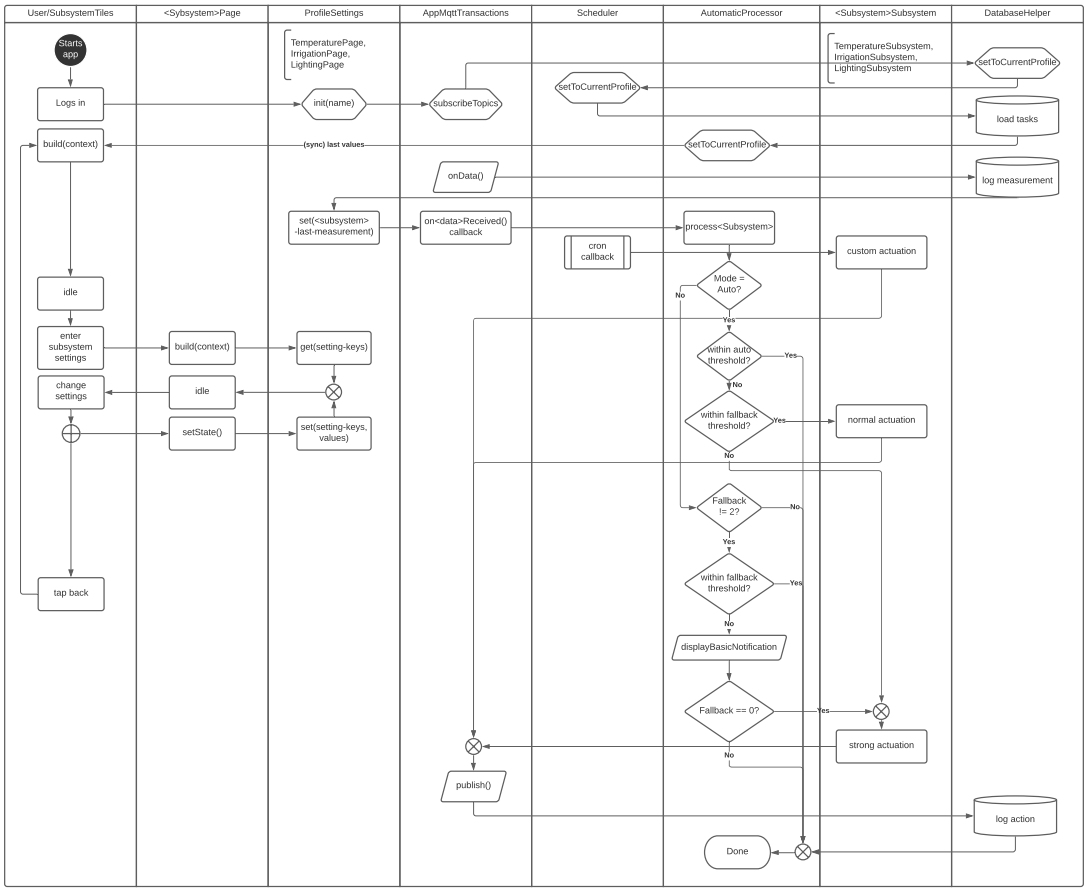

Figure 13: How a typical subsystem works.

8.2.2 Activity Diagram: Reminder

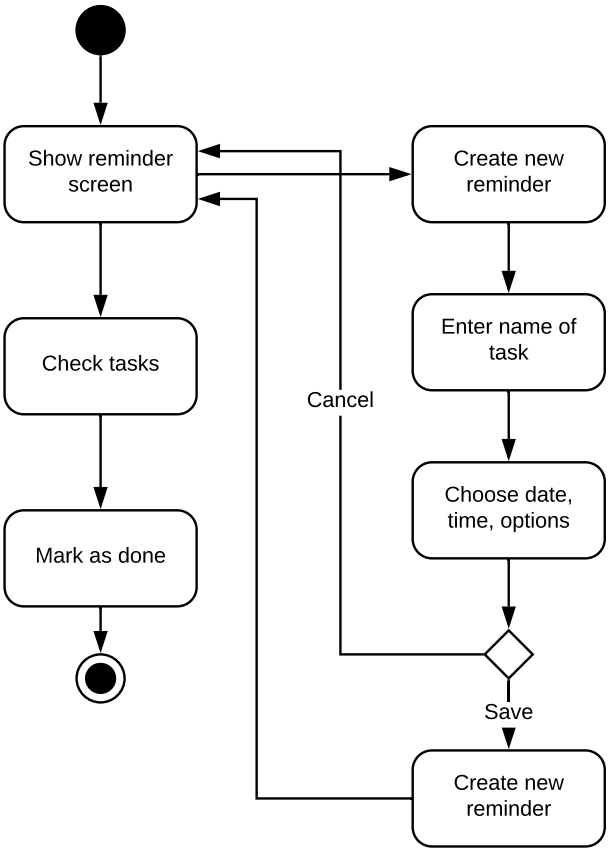

Figure 1: Activity diagram of Reminder functionality

As the start, when users open Reminder screen, it displays Reminder's main screen which has multiple tasks in different categories. When a task is done, user can choose "Mark as done" button to remove it from "Unfinished tasks". Also, a user can add new reminder by choosing add more reminder button, and is navigated to new screen. In the create new reminder screen, users need to choose date and time of reminder, edit the name as descriptive as possible so they can easily be reminded to the task, choose some options of reminder and finally press save to finish or cancel to drop new reminder.

8.2.3 Activity Diagram: Statistics

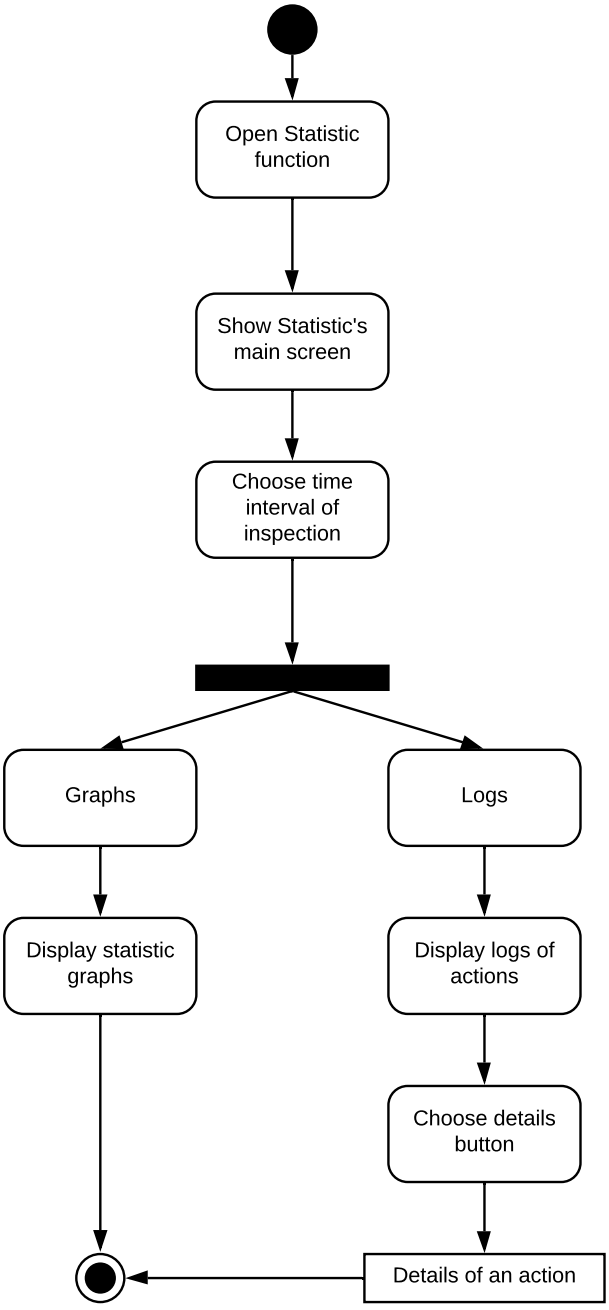

Figure 2: Activity diagram of reading Statistics

When users open Statistics screen, the first thing users will see are Change Time Interval button and the option to switch between displaying Graphs or Logs. As a default, the Graphs page is displayed and shows the statistics graphs of the three subsystems. Users may press the Change Time Interval button to choose a different time interval (rather than, for example. the interval

between Jan 1 to Feb 12 in our mock up of Week Four). Users can also choose to view the logging records by pressing the Logs button and logs of past actions will be displayed. In addition, if users wish to see further information, the Details button will show all details they ought to see.

9 Database Overview

We will also outline our database design using the familiar Enhanced Entity-Relationship Diagram (EERD). The diagram will illustrate the major entities stored in our database, as well as its respective attributes and relationships among each other.

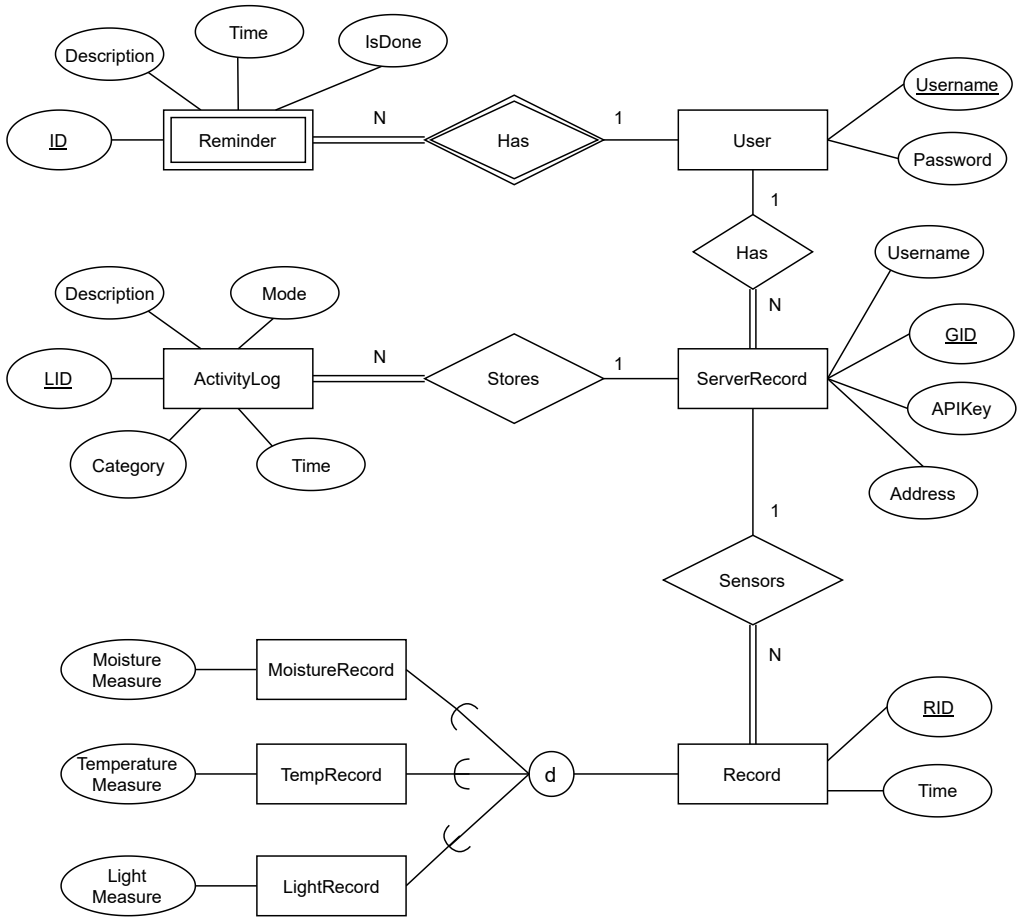

Figure 14: Enhanced Entity-Relationship Diagram for our system

As we can see here, there are core strong entities in **User** (the account created for each person, which identifies his gardens and reminder lists on the app), **Garden** (which represents a Greenhouse garden, maintained by a unique Adafruit topic, and belongs to one user), **Activity-**

Log (which stores the activities performed over the garden since beginning, namely irrigation, sunscreen adjustment, light switching and safety fallback actions), **Record** (which is further divided into **MoistureRecord**, **TempRecord**, **LightRecord**, each of which store data subscribed from the Adafruit topic of a specific garden). Furthermore, each user also has a list of **Reminder**, which is a weak entity dependent on User ID and is a representation of User's current reminder list.

In the mobile application, we will implement this EERD using SQLite (as represented before in the deployment view), which captures fairly well the ideas in the diagram, along with some module-specific adjustment.

10 Finished product

We have implemented our Android application using Dart and Flutter. Below are screenshots of the product, demonstrating a typical workflow:

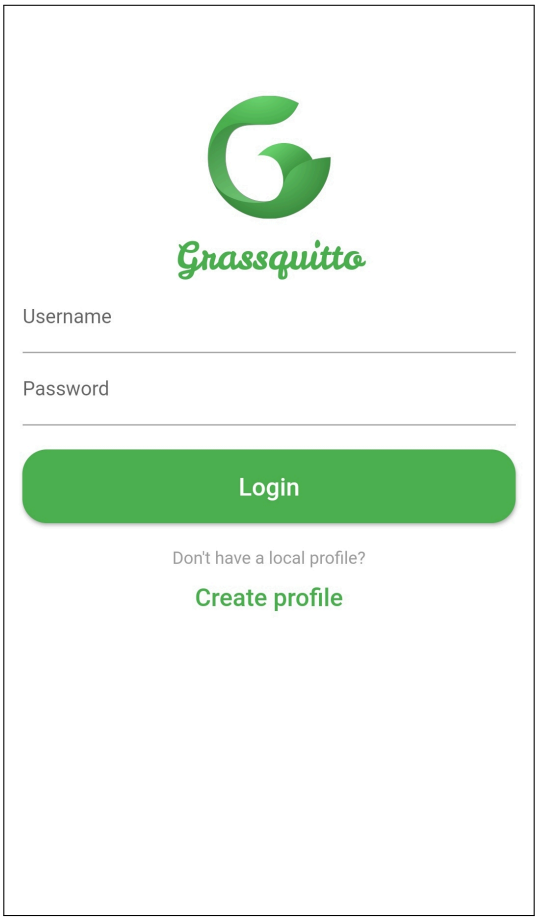

Figure 15: The login screen. It also allows the user to create a new profile if they so wished.

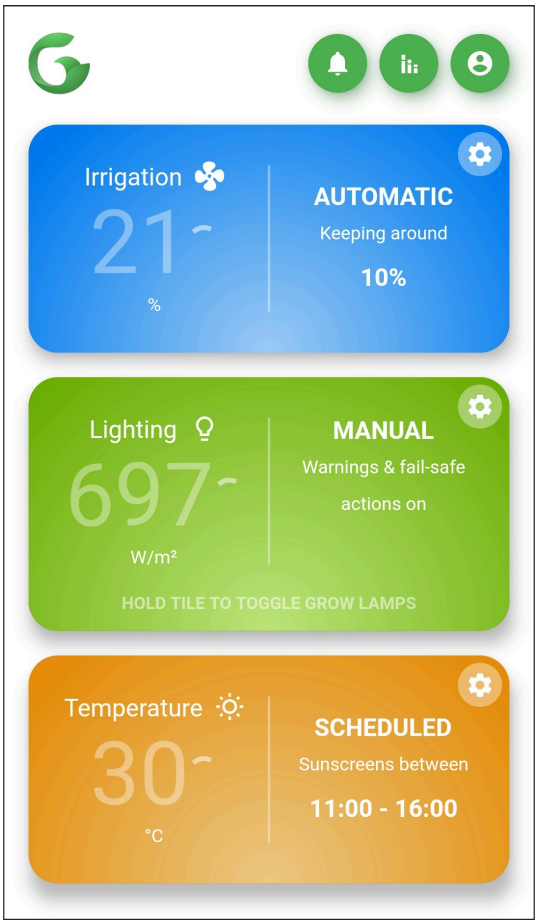

Figure 16: The main screen. Note how the design has changed from the mock-up. **The numbers here are partly transparent with loading spinners next to them**, denoting that they are the last values retrieved from the previous session and not real time numbers (as is the case when first starting up the application). Multiple elements in this screen are animated: the propeller blade next to "Irrigation" would turn while the pump is running. The light bulb next to "Lighting" switches between its outlined and filled version to denote whether the grow lamps are off or on, respectively. The sun icon next to "Temperature" denotes that the sunscreen is currently retracted (the sun is shining through) and switches to a cloudy symbol when the sunscreen is deployed. If in manual mode, these tiles serve as manual buttons, all requiring long presses to avoid accidental activations. When long-pressed, each tile *grows in size and brightens*, signifying that the action has been published to the MQTT broker. Their icons will only change when a reply has been received, denoting a successful activation. Lastly, note how the "Client settings" section has been moved to the top-right, now appearing as a green button with an avatar symbol.

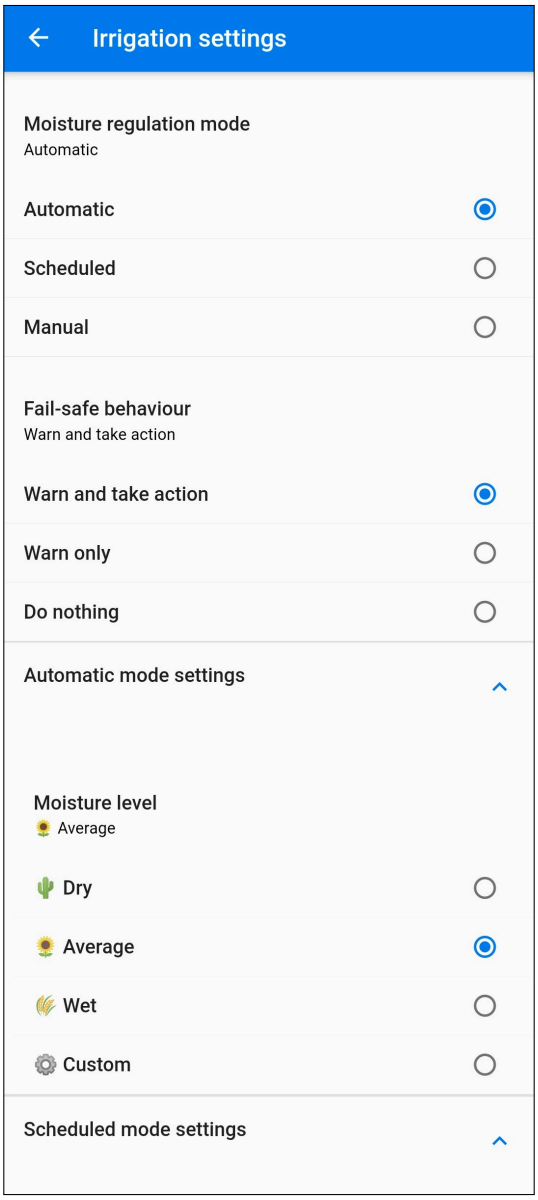

Figure 17: When the small cog icon at the top right of the ”Irrigation” tile is pressed, the tile morphs to fill the entire screen and fades into this settings screen, visually signifying a ”containing” or ”internal” relationship. Note how the colour scheme matches that of the tile. The list of recommended moisture levels are also denoted with small symbols of typical plants of that category. A slider would appear if Custom is selected. The only item in the Scheduled mode settings (not visible here) section leads to the next screen.

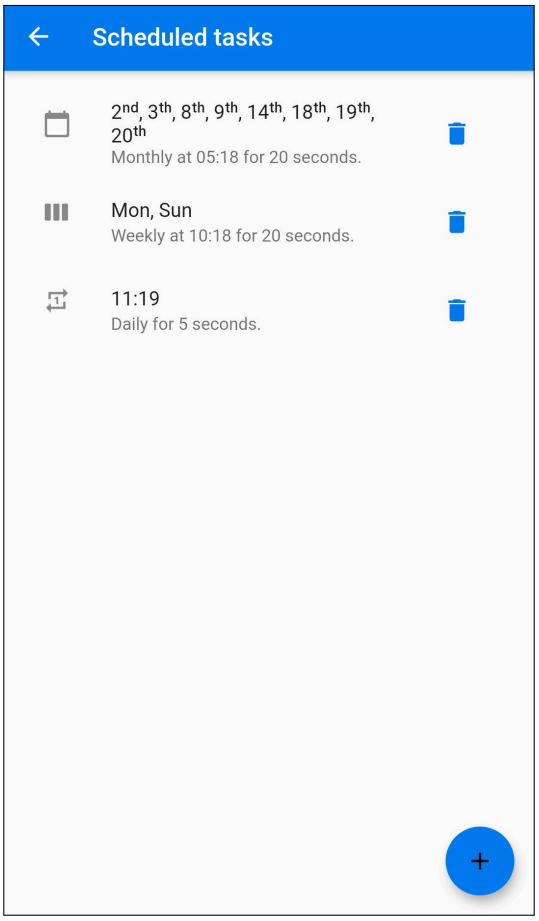

Figure 18: Task list for the scheduled mode of the irrigation subsystem. This list is sorted firstly by period duration (monthly, to weekly, to daily), then by first activation within that period (a monthly task with its first trigger on the 2nd day of a month would be placed above that with the first trigger on the 3rd, for example), followed by pumping duration (ascending). A small Floating Action Button (FOB) at the bottom right allows adding new tasks. It expands to three smaller buttons for daily, weekly and monthly task creation.

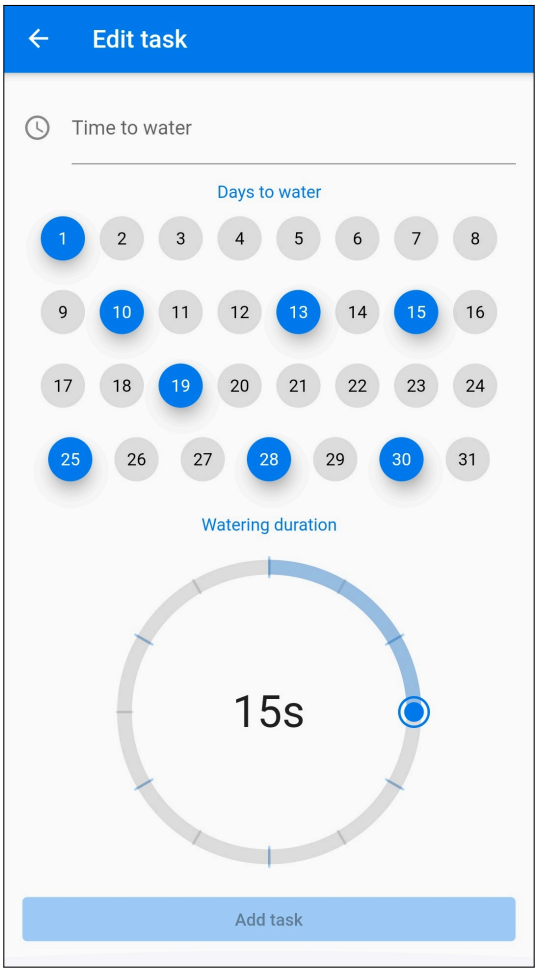

Figure 19: Adding a new monthly-type scheduled task. The user must select a time within a day to activate, as well as which days of the month this task is applicable for. The slider defaults to five seconds and can reach up to sixty (if scrolled all the way around). The Add button is currently disabled due to the user not having selected the time-of-day.

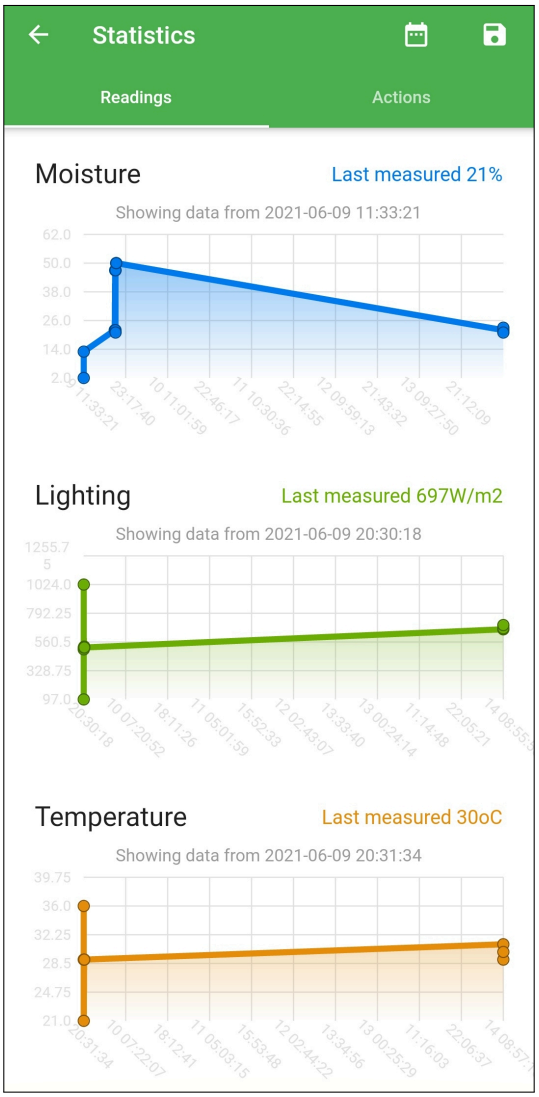

Figure 20: The statistics screen, showing its graphs tab. Each graph is automatically trimmed x-wise to the maximum extent of the available data that fits within the selected time frame (for example, if we only have data starting from 7AM of Tuesday, then the graph will begin at 7AM instead of daybreak even if the selected time frame is simply "From Tuesday till now"). Note how the graphs are coloured to represent their subsystem's theme for direct visual correspondence and legibility.

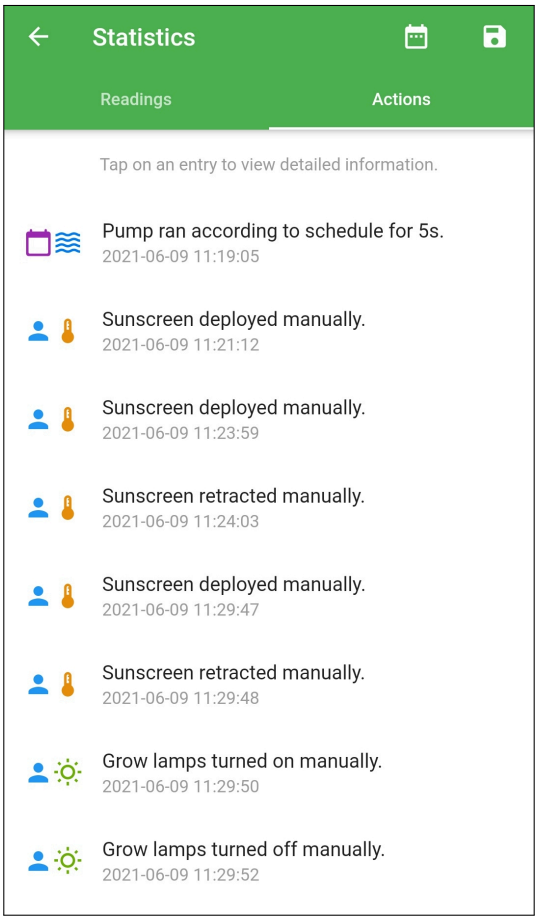

Figure 21: The statistics screen's logs tab, showing *actions taken*. Each entry has two icons, the first denoting which type of action it was (scheduled, manual, or safety fallback) and the second denoting which subsystem it originated from (temperature, moisture or lighting). Descriptions are automatically generated depending on these two icons.

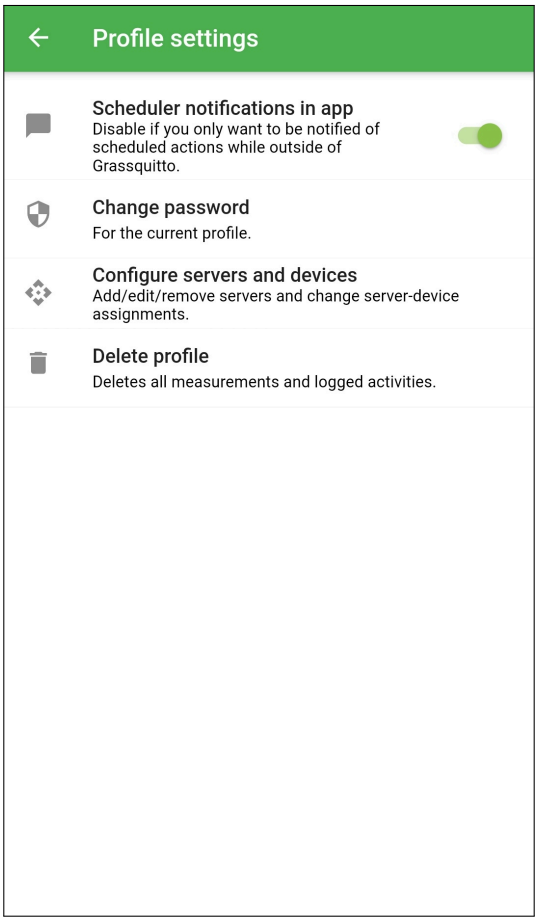

Figure 22: Settings screen for current profile. We are most interested in the server & device setup option.

Figure 23: Tapping that option leads to a two-step wizard, of which setting up server authentication is the first step. Here the user is prompted to add, edit or remove servers from their profile. These servers will be linked to devices later on.

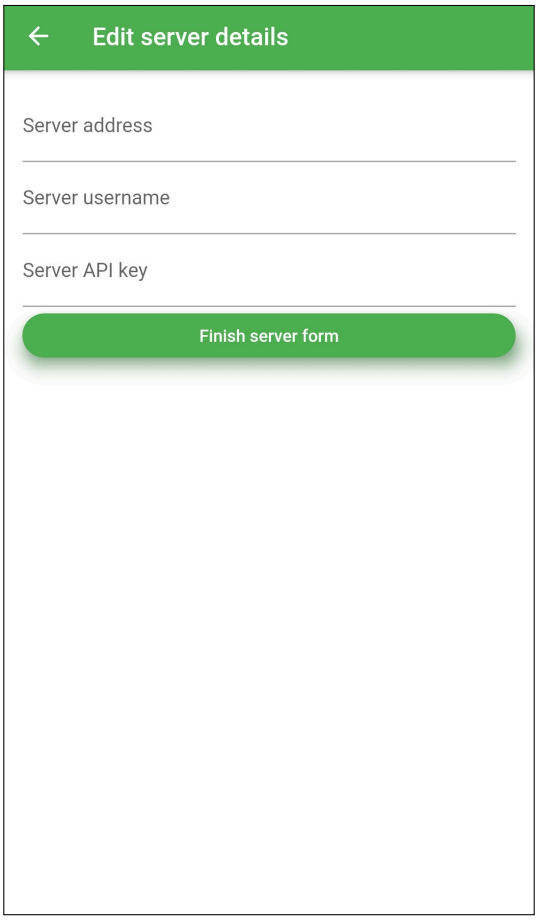

Figure 24: Adding a server requires its domain (for example, io.adafruit.com), your username (for example, CSE-BBC1) and API key.

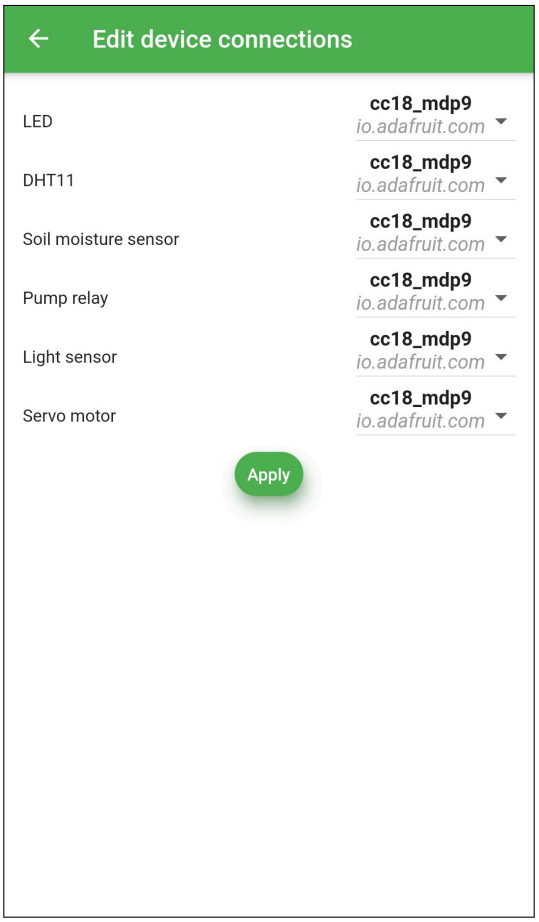

Figure 25: The second step of the wizard. Here each device is linked to the server that hosts their feed. For example, in this screenshot, all six devices are hosted on one server. This design allows for all server-device splits, such as for running on the CSE-BBC test servers.

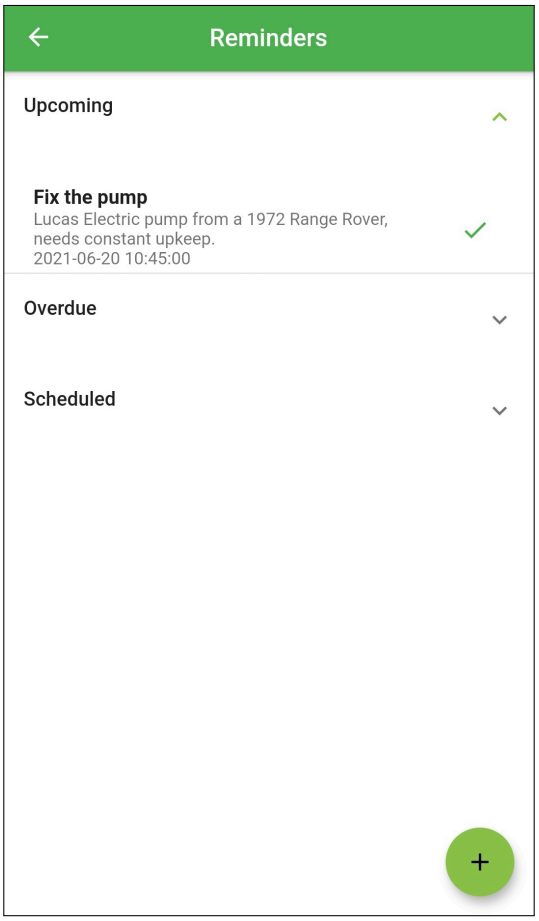

Figure 26: The reminders screen. Reminders are automatically filtered into three categories: Upcoming (due within a day or two), overdue (self-explanatory) and scheduled (the rest of the reminders, far away into the future). As is with the other editable list screens, the FAB at the bottom-right allows adding entries.

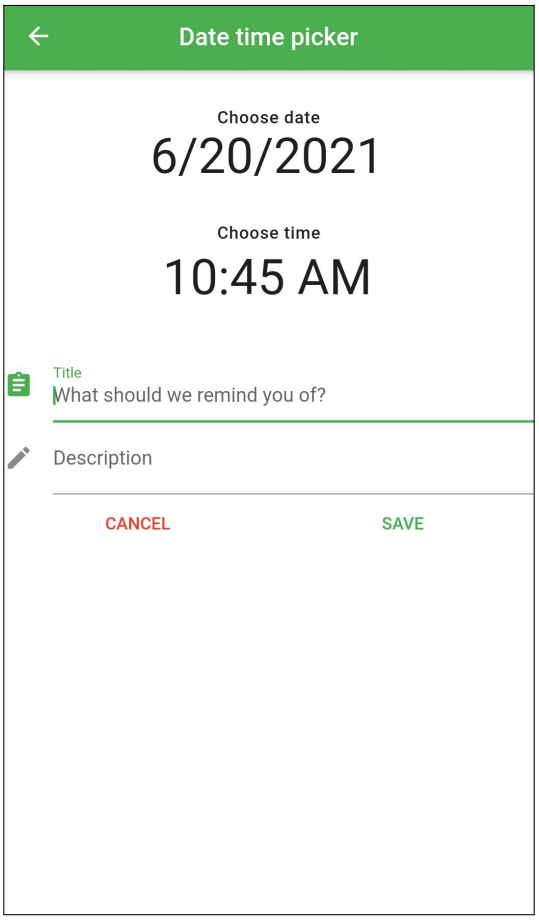

Figure 27: Adding a new reminder.

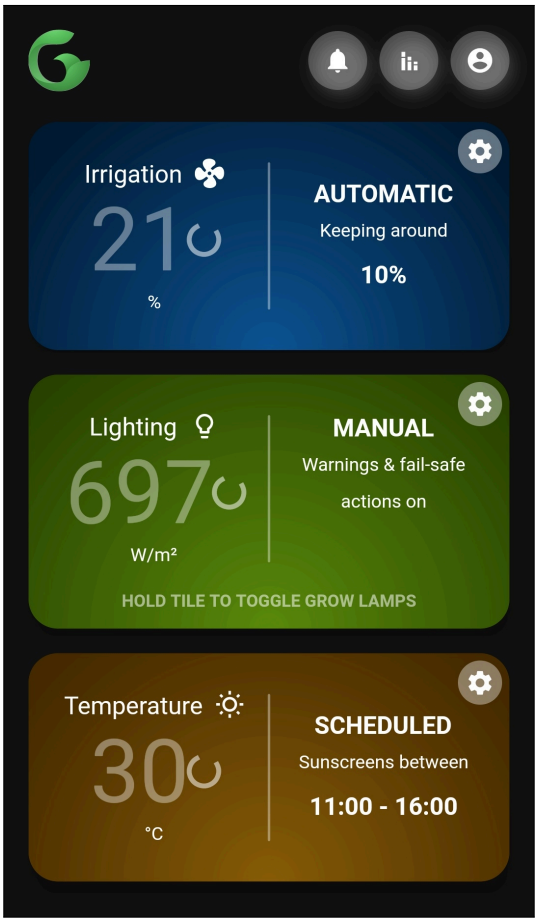

Figure 28: On Android 10+, our application follows system-wide dark mode. Significant work has been done to ensure that the widgets continue to look legible and consistent with their light-mode counterparts, while also preserving eyesight in darker environments. Here is our main screen when the device is in dark mode.

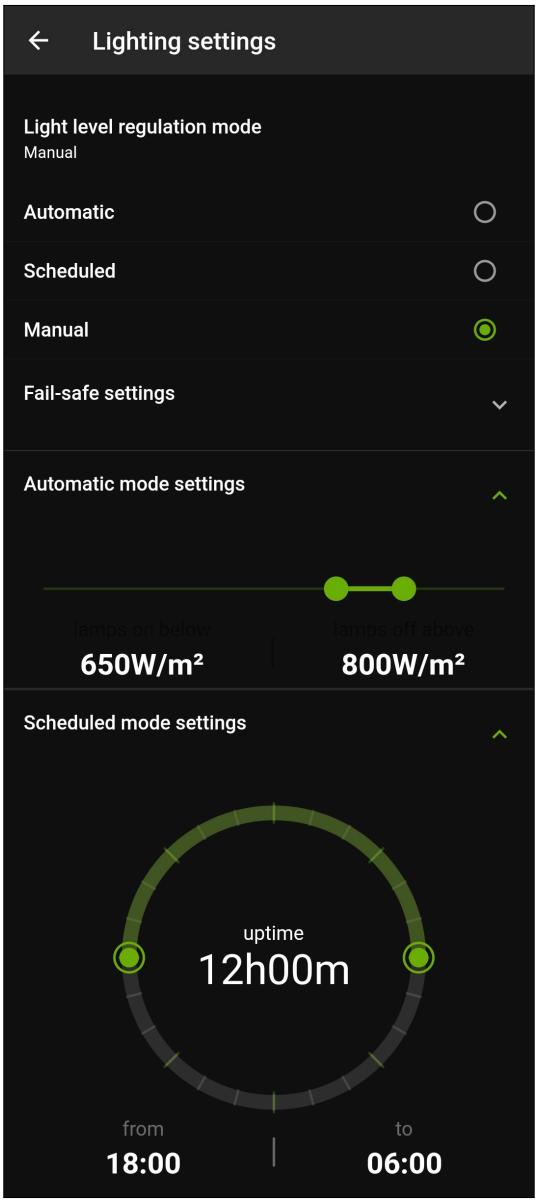

Figure 29: Our lighting subsystem settings screen in dark mode. Again note how the colour scheme matches that of its tile in the main screen.

References

[Dem+16] Sabine Demotes-Mainard et al. “Plant responses to red and far-red lights, applications in horticulture”. In: *Environmental and experimental Botany* 121 (2016), pp. 4–21.