TED UNIVERSITY

Computer Engineering Department

CMPE 492 Senior Project – Low Level Design Report - AgroAutomaTED

by

Korhan Deniz AKIN

Ahmet Can ÖZTÜRK

Kaan BUDAK

10.03.2024

1. Introduction

1.1. Object Design Trade-offs

1.1.1. Using A Lot Of Sensors:

In the AgroAutomaTED project, multiple sensors were used instead of a single sensor. This choice increases the efficiency and reliability of the system by ensuring more comprehensive and accurate data collection. However, using multiple sensors also has some disadvantages:

- **Higher cost**: Using multiple sensors is expensive.
- **More complex installation**: Installation and integration of multiple sensors is more complex than a single sensor.
- **Storage**: Sensor data is planned to be stored in the database server. However, the size of the data is unpredictable since data is received frequently. Therefore, it requires a lot of space

In the AgroAutomaTED project, the advantages outweigh the disadvantages of more comprehensive and accurate data collection. Therefore, the use of multiple sensors was preferred.

1.1.2. Central Processing Unit Selection:

In the AgroAutomaTED project, the choice of the central processing unit was an important design decision. Initially, using a less powerful processor such as the Arduino UNO was considered. This option is lower cost and offers a simpler installation. However, since the artificial intelligence model (AI) was planned to be implemented in the system, the use of a more powerful processor was required. A more powerful processor allows the AI model to run faster and more efficiently, increasing the overall performance of the system.

For Arduino

- **Cost**: More powerful processors are more expensive than Arduino. Since the budget of the project was limited, Arduino, which was more affordable in terms of cost, was preferred.

- **Complexity**: More powerful processors require a more complex installation and programming process. Arduino was preferred to complete the project more easily and quickly.
- **Artificial Intelligence Model**: The artificial intelligence model will be run on the cloud system. Therefore, Arduino's processing power will not be a hindrance to run the AI model.

Cloud System Usage

The reasons for running the artificial intelligence model on the cloud system are as follows:

- **More processing power**: Cloud systems offer much more processing power than Arduino. This will enable the AI model to run faster and more efficiently.
- **Greater flexibility**: Cloud systems can be easily integrated with different AI models. This will make it easier to adapt the project to future needs.
- **Less cost**: Cloud systems can be used at a lower cost than the cost of purchasing and installing a powerful processor.

1.1.3. Communication Protocols:

Instead of serial communication, wireless communication protocols such as Wi-Fi are used in the system through the NodeMCU 1.0 module. This choice significantly increases the range and flexibility of the system. Users can access the system and monitor data from anywhere with internet access. However, some security risks brought by wireless communication should also be taken into consideration. Wi-Fi networks can be sensitive to unauthorized access and compromise data security. Therefore, necessary precautions must be taken to ensure the security of the system.

Advantages

- **Wider range**: Wireless communication allows data transmission over a wider area, unlike serial communication. In this way, users' data can be monitored and controlled even if they are remote from the system.

- **Greater flexibility**: Wireless communication allows users to access the system via wireless devices such as mobile devices or tablets. This makes the system more useful and accessible.

Disadvantages

- **Security risks**: Wireless communication poses more security risks than serial communication. Care should be taken against security vulnerabilities such as unauthorized access and data theft.

Wireless communication protocols significantly increase the range and flexibility of the AgroAutomaTED project when used in conjunction with the NodeMCU 1.0 module. Users can access the system and monitor data from anywhere with internet access.

1.2. Interface documentation guidelines

In this document classes, attributes, and methods are all named in camel case format according to the naming standard followed throughout the report. Some class names start with lowercase letters, but others begin with capital letters. Class interface descriptions follow the structure given below:

class ClassName
Explanation of the class
Attributes
typeOfAttribute nameOfAttribute
Methods
returnType and methodName

1.3. Engineering standards (e.g., UML and IEEE)

In this report, we have employed UML principles [1] to construct class interfaces, diagrams, scenarios, and subsystem compositions. UML stands as a widely recognized and user-friendly method for crafting such diagrams, a practice endorsed by the Software Engineering Course in Ted University curriculum. Adhering to IEEE citation guidelines [2], our decision to utilize UML in the ensuing sections aligns with Ted University's preference for this approach—a widely embraced methodology across various domains.

1.4. Definitions, acronyms, and abbreviations

AI: Artificial Intelligence
DHT11: Humidity and Temperature Sensor
HC-SR04: Distance Sensor
NPK: Nitrogen, Phosphorus, Potassium
NodeMCU 1.0: Wi-Fi Module
WebSocket: WebSocket is a persistent communication protocol within a server and clients. It uses a single TCP protocol and the connection is not closed except the server or client interrupts it. Therefore, it provides faster data transmission compared with other protocols.
Firebase Storage: A database by Google which can hold large amounts of data.

2. Packages

2.1. Hardware Components' Software Implementation Class Diagram

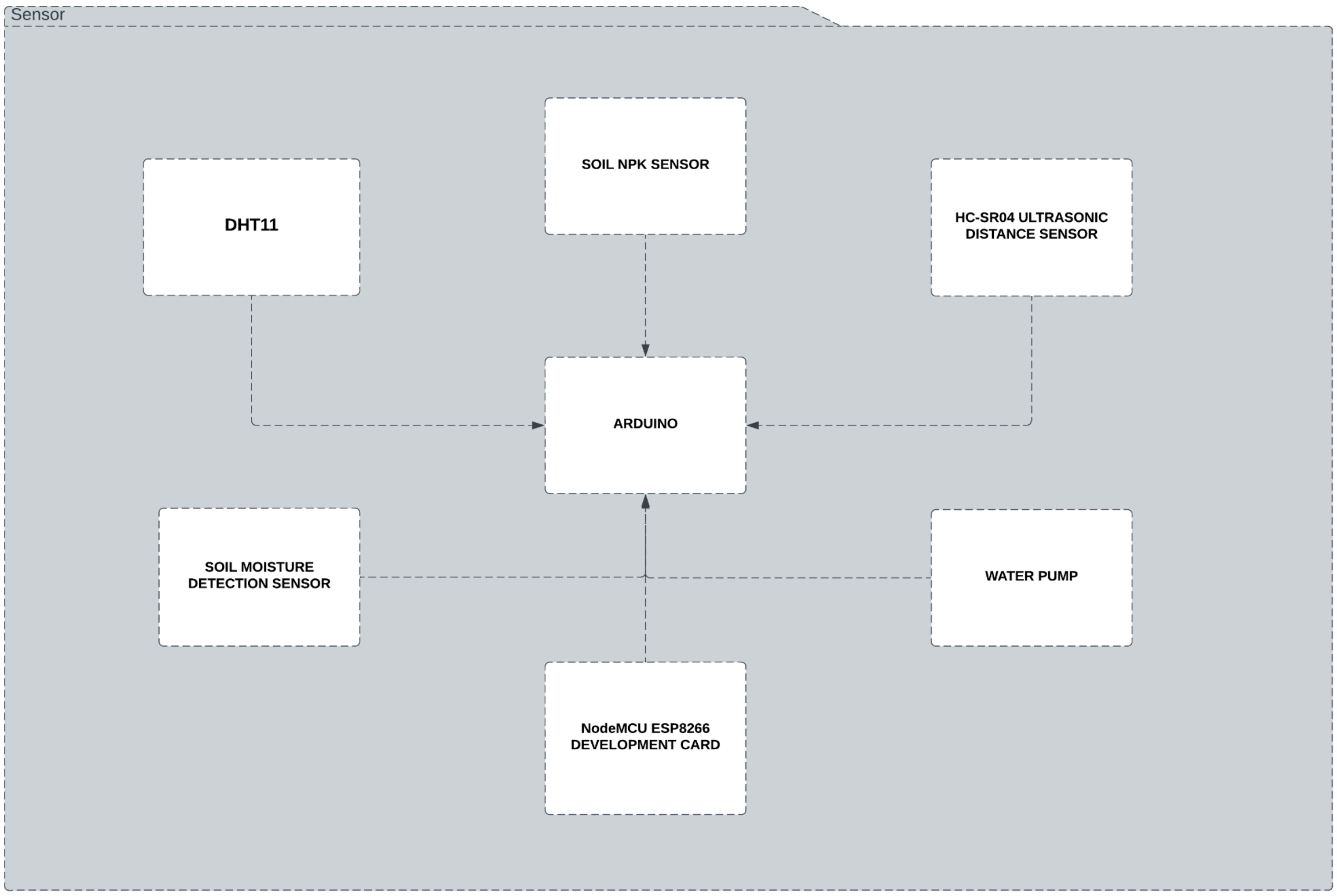
Sensor
SOIL NPK SENSOR
DHT11
HC-SR04 ULTRASONIC DISTANCE SENSOR
ARDUINO
SOIL MOISTURE DETECTION SENSOR
WATER PUMP
NodeMCU ESP8266 DEVELOPMENT CARD

Diagram 1. This diagram shows the relations between software implementations of the hardware components.

2.2. Backend Implementations Class Diagram

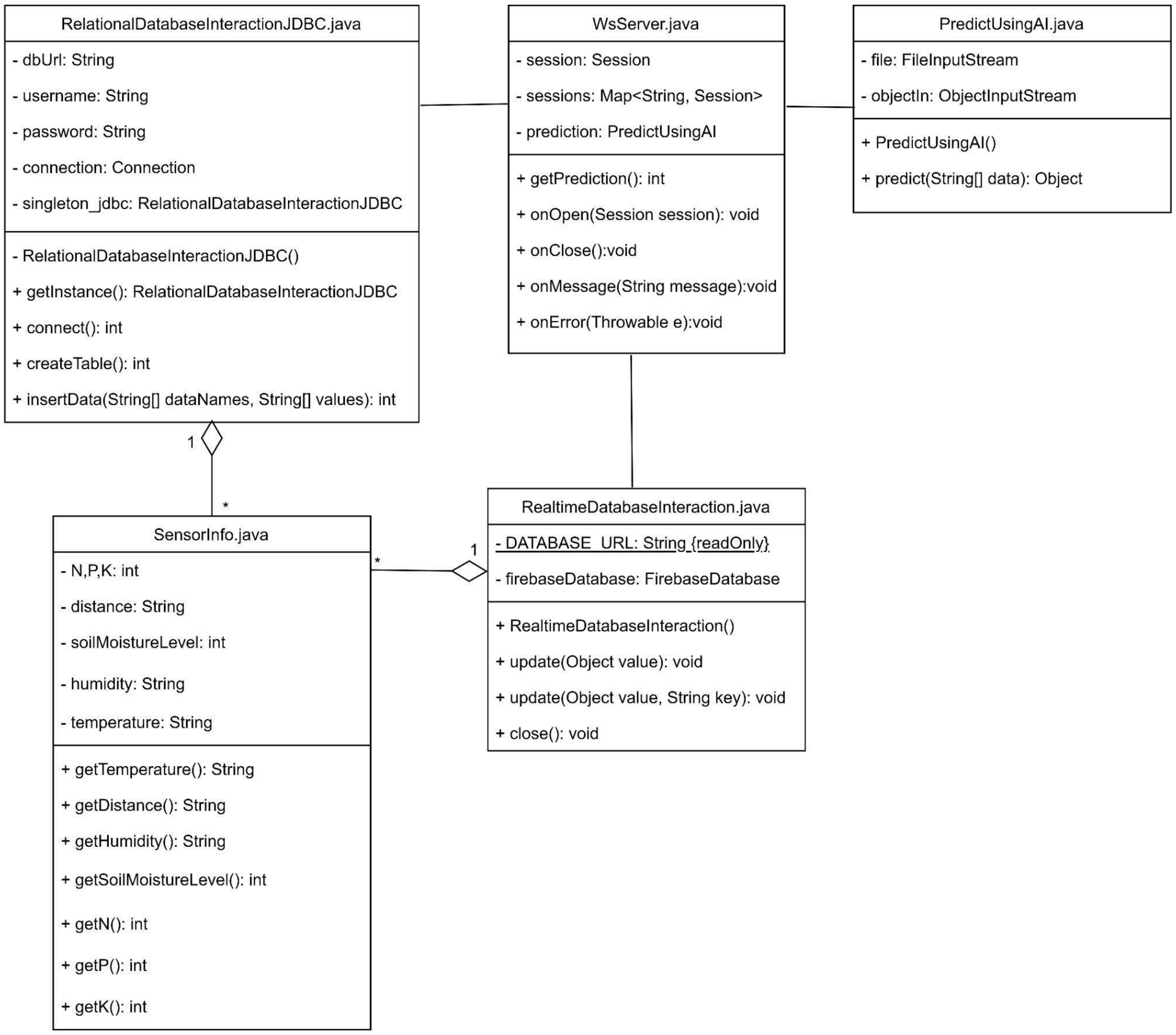
RelationalDatabaseInteractionJDBC.java
- dbUrl: String
- username: String
- password: String
- connection: Connection
- singleton_jdbc: RelationalDatabaseInteractionJDBC
- RelationalDatabaseInteractionJDBC()
+ getInstance(): RelationalDatabaseInteractionJDBC
+ connect(): int
+ createTable(): int
+ insertData(String[] dataNames, String[] values): int
WsServer.java
- session: Session
- sessions: Map<String, Session>
- prediction: PredictUsingAI
+ getPrediction(): int
+ onOpen(Session session): void
+ onClose():void
+ onMessage(String message):void
+ onError(Throwable e):void
PredictUsingAI.java
- file: FileInputStream
- objectIn: ObjectInputStream
+ PredictUsingAI()
+ predict(String[] data): Object
1
*
SensorInfo.java
- N,P,K: int
- distance: String
- soilMoistureLevel: int
- humidity: String
- temperature: String
+ getTemperature(): String
+ getDistance(): String
+ getHumidity(): String
+ getSoilMoistureLevel(): int
+ getN(): int
+ getP(): int
+ getK(): int
*
1
RealtimeDatabaseInteraction.java
- DATABASE_URL: String {readOnly}
- firebaseDatabase: FirebaseDatabase
+ RealtimeDatabaseInteraction()
+ update(Object value): void
+ update(Object value, String key): void
+ close(): void

Diagram 2. This diagram demonstrates the UML class diagram of our Backend implementations

2.3. Frontend Implementations Class Diagram

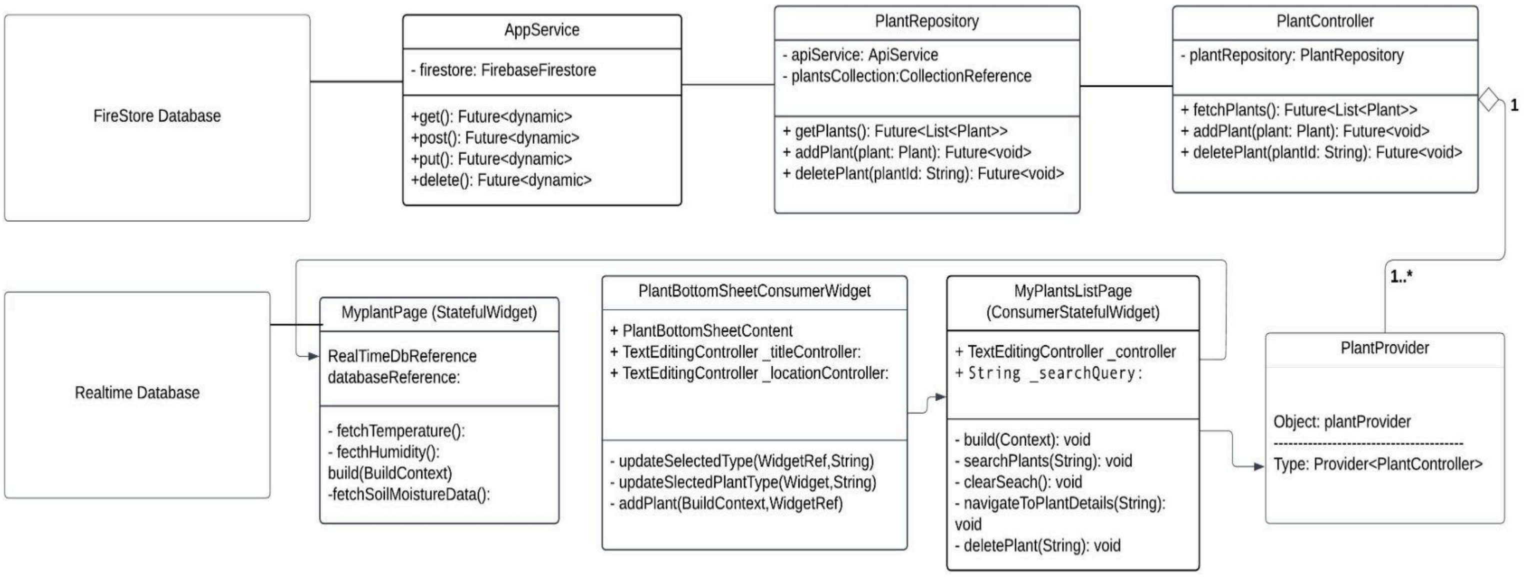
FireStore Database
AppService
- firestore: FirebaseFirestore
+get(): Future<dynamic>
+post(): Future<dynamic>
+put(): Future<dynamic>
+delete(): Future<dynamic>
PlantRepository
- apiService: ApiService
- plantsCollection:CollectionReference
+ getPlants(): Future<List<Plant>>
+ addPlant(plant: Plant): Future<void>
+ deletePlant(plantId: String): Future<void>
PlantController
- plantRepository: PlantRepository
+ fetchPlants(): Future<List<Plant>>
+ addPlant(plant: Plant): Future<void>
+ deletePlant(plantId: String): Future<void>
1
1..*
Realtime Database
MyplantPage (StatefulWidget)
RealTimeDbReference
databaseReference:
- fetchTemperature():
- fecthHumidity():
build(BuildContext)
-fetchSoilMoistureData():
PlantBottomSheetConsumerWidget
+ PlantBottomSheetContent
+ TextEditingController _titleController:
+ TextEditingController _locationController:
- updateSelectedType(WidgetRef,String)
- updateSlectedPlantType(Widget,String)
- addPlant(BuildContext,WidgetRef)
MyPlantsListPage
(ConsumerStatefulWidget)
+ TextEditingController _controller
+ String _searchQuery:
- build(Context): void
- searchPlants(String): void
- clearSeach(): void
- navigateToPlantDetails(String):
void
- deletePlant(String): void
PlantProvider
Object: plantProvider
Type: Provider<PlantController>

Diagram 3. This diagram visualises the UML class diagram of our Frontend implementations which is a mobile application.

3. Class Interfaces

3.1. UI/UX Interfaces

3.1.1. Flutter SignUp Page Component

Purpose: The sign-up page component facilitates user registration within the mobile application.

Functionality:

- Allows users to input their first name, last name, address, email, and password.
- Validates user inputs and ensures password confirmation.
- Communicates with Firebase Authentication and Firestore to create user accounts and store user details.
- Provides visual feedback through snack bars for errors and successful registrations.

Dependencies:

- Utilizes Firebase Authentication for user authentication.
- Communicates with Firestore for storing user details.
- Requires custom widgets class which are AppTextField, AppButton, AppSnackBar and AppText.

Interactions:

- Interacts with various text fields and buttons to capture user input and trigger sign-up actions.
- Communicates with Firebase services for user authentication and data storage.

3.1.2. Flutter Login Page Component

Purpose: The login page component facilitates user authentication and access to the application.

Functionality

- Allows users to input their email and password for authentication.
- Validates user credentials and handles authentication errors.
- Provides options for password recovery through a "Forgot Password?" link.
- Offers the ability to switch between dark and light themes.

Dependencies:

- Uses Firebase Authentication for user login functionality.

Interactions:

- Interacts with text fields for capturing user email and password inputs.
- Communicates with Firebase Authentication for user authentication.
- Displays feedback through snack bars for authentication errors.
- Allows users to navigate to the sign-up page for account creation.
- Requires custom widget classes which are AppTextField, AppButton, AppSnackBar, AppText and AppTheme.

3.1.3. Flutter Home Page Component

Purpose: The Home Page component provides the initial landing page for users to log into the application. It provides a welcoming message along with user-specific notifications fetched from the Firebase system.

Functionality

- Retrieves notification data from Firebase and displays it on the page.
- Utilises a bottom navigation bar for easy access to other pages within the application.

Dependencies:

- Uses Firebase services, which are Firebase Cloud Messaging (FCM), for fetching and displaying notifications.
- Uses Firebase Firestore to retrieve user data.
- Requires custom widget classes which are AppAppBar, AppBottomNavigationBar, AppText,

Interactions:

- Communicates with Firebase services to fetch and display user-specific notifications.
- Utilizes Flutter's navigation system to transition between different pages within the application.
- Dynamically updates the UI to reflect changes in notification data or user interactions.

3.1.4. Flutter MyPlantsList Page Component

Purpose: The MyPlantsList Page component displays a list of plants associated with the current user and allows users to manage their plants.

Functionality

- Displays a list of plants filtered by the current user's ID and search query.
- Allows users to add new plants to their collection.
- Provides options to search for specific plants and delete existing plants.
- Supports navigation to individual plant details pages.

Dependencies:

- Utilizes Firebase Firestore for retrieving plant data and managing plant collection.
- Requires custom widget classes which are AppIcon, AppButton, AppAppBar and AppText.

Interactions:

- Interacts with text fields for search queries and text editing.
- Communicates with Firebase Firestore to fetch and manipulate plant data.
- Navigate to the individual plant details page when selecting a plant.
- Utilizes bottom sheet widget for adding new plants to the collection.

3.1.5. Flutter AddPlant Bottom Sheet Component

Purpose: The Plant Bottom Sheet Content component provides a modal bottom sheet for users to add new plants to their collection.

Functionality

- Displays input fields for users to enter plant details such as title and location.
- Allows users to select the plant environment (indoor or outdoor) and plant type.
- Communicates with Firebase Firestore to add new plant data to the database.

Dependencies:

- Uses Firebase Firestore for storing and managing plant data.
- Utilizes Riverpod for state management and dependency injection.
- Requires custom widgets class which are AppTextField, AppButton and AppText.

Interactions:

- Utilizes a modal bottom sheet for a seamless user experience in adding new plants.

- Communicates with parent widgets using Riverpod for state management and data flow.
- Retrieves plant types asynchronously and populates the picker widget accordingly.
- Handles user input validation and adds new plant data to the database with confirmation.

3.1.6. Flutter MyPlant Page Component

Purpose: The MyPlantPage displays various environmental data related to plants, including soil moisture, temperature, humidity, and other related information.

Functionality

- Retrieves data from Firebase Realtime Database to display real-time values for soil moisture, temperature, and humidity.
- Presents an overview of environmental conditions and plant status using icons and textual representations
- Includes a notification switch button that allows users to toggle notifications on or off.
- Supports navigation to the irrigation plan page.

Dependencies:

- Uses Firebase Realtime Database for retrieving real-time environmental data.
- Utilizes Font Awesome Icons and standard Flutter icons for visual representations.
- Requires custom widgets class which are AppIcon, AppButton, AppText, AppAppBar.

Interactions:

- Communicates with Firebase Realtime Database to fetch data asynchronously and updates UI accordingly.
- Utilizes future builders to handle asynchronous data retrieval and display loading indicators or error messages as needed.

3.1.7. Flutter Guide Page Component

Purpose: The Flutter Guide Page includes a definitive step-by-step guide for users to comprehensively learn and understand how to set up a card and utilize the application's features.

Functionality

- Presents users with a sequential and structured guide for effectively using the application.
- Each step is accompanied by clear and concise explanatory text along with visual aids when necessary.
- It includes displayed buttons allowing users to seamlessly navigate to the next step or skip to alternative sections of the guide.

Dependencies:

- Requires the AppAppBar, AppButton, AppText, AppIcon, and AppDivider project custom widgets for UI components.

Interactions:

- Responds promptly to user interactions, enabling users to effortlessly navigate between guide steps or sections.
- Makes use of responsive design concepts to modify user interface elements according to various screen sizes and resolutions.

3.1.8. Flutter User Page Component

Purpose: The User Page provides user profile information and options for managing settings related to the application.

Functionality

- The User Page provides user profile information and options for managing settings related to the application.
- Retrieves user data from the Firestore database and displays it on the page.
- Allows users to edit their profile information.
- Provides options for toggling between light and dark themes.
- Offers various settings options such as syncing, language selection, security, about page, feedback, and logout.

Dependencies:

- Uses Firebase Authentication and Firestore for user authentication and data storage.
- Utilizes Riverpod for state management and theme handling.

- Requires the AppAppBar, AppButton, AppText, AppIcon, and AppDivider project custom widgets for UI components.

Interactions:

- Communicates with Firestore database to retrieve user profile information and update settings.
- Handles user sign-out using Firebase Authentication.
- Utilizes responsive design principles to adapt UI elements based on different screen sizes and resolutions.

3.2. RetrieveSensorDataToArduino.ino

3.2.1. DHT11 Humidity and Temperature Sensor

Interface: A humidity and temperature sensor is a sensor used to measure humidity and temperature levels in the environment. The class interface contains the necessary functions to read humidity and temperature data and transfer them to other components.

Functionality: The sensor measures the humidity and temperature levels in the environment at regular intervals and transmits this data to the central control unit. Thanks to this data, it is used to monitor the growth conditions of the plants and also to adjust the irrigation system.

3.2.2. HC-SR04 Ultrasonic Distance Sensor

Interface: A distance sensor is a sensor used to detect the presence of an object near it. The class interface contains the necessary functions to detect the presence of objects and convey this information to other components.

Functionality: The sensor detects the presence in a specific area, such as a water tank or plant. This information is used to check the filling status of the water tank. The closer the water level is to the sensor, the higher the water content.

3.2.3. Soil Hygrometer Moisture Detection Sensor

Interface: A soil moisture sensor is a device used to measure the amount of moisture present in the soil. The class interface contains the necessary functions to read soil moisture and transmit this data to other components.

Functionality: The sensor reads soil moisture within a certain range and transmits this information to the central unit. This information is used to determine the irrigation needs of plants and automatically manage the irrigation process. The water saturation of the soil is determined according to certain level ranges. The user is informed according to these level ranges and the necessary adjustments are made by the system or the user.

3.2.4. NPK sensor

Interface: The class interfaces of these sensors include the necessary functions to measure soil fertility, pH level, nitrogen, potassium and phosphorus levels.

Functionality: These sensors are used to monitor soil quality in more detail and optimize plant growth. These sensors, which are planned to be integrated in future improvements, are planned to provide more comprehensive irrigation management. In this context, the previously developed artificial intelligence model tries to provide the optimal environment by using the N, P, and K data coming from the sensors.

3.3. WebsocketConnection.ino

Interface: Establishes the connection between NodeMCU ESP8266 WiFi Module to the Cloud Server.

Functionality

- NodeMCU module receives sensor data from the Arduino UNO Card via the Software Serial communication.
- Send received sensor data to the WebSocket server.

Dependencies:

- WebSocketsClient.h: A library to connect the WebSocket server.
- ESP8266WiFi.h: A library to connect to a WiFi network.

WebSocketConnection.ino
Establishes the connection between NodeMCU ESP8266 WiFi Module to the Cloud Server.

Attributes
const char* ssid
const char* password
const char* webSocketServerAddress
const uint16_t webSocketServerPort
const String webSocketEndpoint
String temperature
String humidity
int soil_moisture_level
String distance
int N
int P
int K
Methods
void setup()
void webSocketEvent(WStype_t type, uint8_t * payload, size_t length)
void loop()
void sendDataAsMessage(String temperature, String humidity, int soil_moisture_level, String distance, int N, int P, int K)

3.4. WsServer.java

Interface: The purpose of this class is to create another endpoint of the WebSocket server. This endpoint relies on the Backend side of the project. After receiving the data via the Websocket message it writes them to the Firebase Realtime database and normal database.

Functionality

- This class receives sensor data from the NodeMCU module via WebSocket messages.
- This data will be processed by the backend application which consists of artificial intelligence models.
- After this process sensor data (received as WebSocket messages) will be written to the "Firebase Storage"

Dependencies:

- javax.websocket library.

WsServer.java
The purpose of this class is to create another endpoint of the WebSocket server.
Attributes
private Session session
private final Map<String, Session> sessions
Methods
public void onOpen(Session session)
public void onClose()
public String onMessage(String message)
public void onError(Throwable e)

3.5. SensorInfo.java

Interface: Hold sensor data.

Functionality

- Comprised of sensor attributes, getters and setters.

SensorInfo.java
Hold sensor data.
Attributes
private String temperature
private String humidity
private int soilMoistureLevel
private String distance
private int N,P,K
Methods
public String getTemperature()
public String getHumidity()
public int getSoilMoistureLevel()
public String getDistance()
public int getN()
public int getP()
public int getK()

3.6. RealtimeDatabaseInteraction.java

Interface: From the “WsServer.java” class, the sensor data is transferred to the “RealtimeDatabaseInteraction.java” class. This class is responsible for sending them to the Firebase Realtime Database.

Functionality

- Writes sensor data to Firebase Realtime Database.

Dependencies:

- com.google.firebase.database

RealtimeDatabaseInteraction.java
This class receives sensor data from the WsServer.java class and writes them to the Firebase Realtime Database Server
Attributes
private static final String DATABASE_URL
private FirebaseDatabase firebaseDatabase
Methods
public RealtimeDatabaseInteraction()
public update(Object value): void
public update(Object value, String key): void
public close(): void

3.7. RelationalDatabaseInteractionJDBC.java

Interface: Similar to the RealtimeDatabaseInteraction.java class, it also writes to the database which is relational such as MySQL.

Functionality

- Take sensor data from the WsServer.java class and write them to the database

Dependencies:

- java.sql.*

RelationalDatabaseInteractionJDBC.java
Write sensor data to a relational database to store them for a long-term manner.
Attributes
private String dbUrl
private String username
private String password
private Connection connection
private RelationalDatabaseInteractionJDBC singleton_jdbc
Methods
private RelationalDatabaseInteractionJDBC()
public RelationalDatabaseInteractionJDBC getInstance()
public int connect()
public int createTable()
public int insertData(String[] dataNames, String[] values)

3.8. PredictUsingAI.java

Interface: This class uses artificial intelligence models to predict whether irrigation must occur or not and give recommendations to users about the soil.

Functionality

- Make predictions using artificial intelligence models.

Dependencies:

- net.razorvine.pickle

3.9. Model

User Class
The User class in this Flutter application serves as a data model representing user entities, encapsulating essential user attributes. It provides a constructor for creating User objects with required parameters and includes methods to serialize (toJson()) and deserialize (fromJson()) user objects to and from JSON format, enabling seamless data interchange with external sources like APIs and databases. This class facilitates the organization, manipulation, and transfer of user data within the application.
Attributes
Public String userId
Public String address
Public String email
Public String firstName
Public String lastName
Methods
fromJson and toJson methods.

Plant Class
The Plant class in the system for plants project serves as a data model representing plant entities. It encapsulates essential attributes. The constructor allows for creating Plant objects with required parameters and defaults the notification parameter to true if not explicitly provided. The class also includes methods to serialise (toJson()) and deserialize (fromJson()) plant objects to and from JSON format, facilitating seamless data exchange with external sources such as databases or APIs. This class plays a crucial role in organizing, managing, and exchanging plant-related data within the smart irrigation system, contributing to its functionality and data management capabilities.
Attributes
Public String title
Public String type
Public String location
Public bool isIndoor
Public bool notification
Public String isInuserIddoor
Methods
fromJson and toJson methods.

PlantType Class
The PlantType class is designed to represent various types of plants in our dataset within a system, encapsulating important attributes. This class facilitates the organization and management of plant-related data by providing a structured blueprint for defining different plant types and their associated characteristics. The fromJson() method, enables the conversion of JSON data into PlantType objects, allowing seamless integration with external data sources or APIs. Conversely, the toJson() method enables the serialization of PlantType objects into JSON format, enabling efficient storage, transmission, and exchange of plant-type information. Overall, the PlantType class plays a pivotal role in the management and representation of plant types within the system, contributing to its functionality and data management capabilities.
Attributes
Public String name
Public Float temperature
Public Float humidity
Public Float soilMoisture
Methods
fromJson and toJson methods.

4. Glossary

- API: Application Programming Interface. A set of rules and protocols that allows different software applications to communicate with each other.
- Arduino UNO: A microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as, 6 analog inputs).
- Cloud Server: A server hosted in a data center and accessible via the internet, providing various services and resources on-demand.
- Cloud System: A system of servers or resources accessed via the internet, providing storage, computing power, and various services on-demand.
- DHT11: A digital temperature and humidity sensor that utilizes a capacitive humidity sensor and a thermistor to measure the surrounding air and output the relative humidity and temperature.
- Distance Sensor: A sensor used to measure the distance of an object from the sensor's surface, often utilizing ultrasonic or infrared technology.
- ESP8266: A low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability produced by Espressif Systems.
- Firebase Authentication: A service provided by Firebase that allows developers to authenticate users using passwords, phone numbers, or popular identity providers like Google, Facebook, and Twitter.
- Firebase Storage: A cloud storage solution provided by Google Firebase for storing user-generated content, such as photos and videos, and sharing them across apps.
- Firestore: Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud Platform. It keeps your data in sync across client apps through real-time listeners and offers offline support for mobile and web.
- Flutter: An open-source UI software development kit created by Google, used to develop cross-platform applications for mobile, web, and desktop from a single codebase.
- HC-SR04: An ultrasonic distance sensor module that provides accurate distance measurements via ultrasonic waves.

- IEEE: Institute of Electrical and Electronics Engineers. A professional association for electronic engineering and electrical engineering that develops international standards in a broad range of industries, including software engineering.
- IEEE Standards: Standards developed by the Institute of Electrical and Electronics Engineers, ensuring interoperability, functionality, and performance of various technologies and products.
- NPK: Refers to the macronutrients Nitrogen (N), Phosphorus (P), and Potassium (K), essential for plant growth and health.
- NodeMCU 1.0: A Wi-Fi-enabled development board featuring the ESP8266 Wi-Fi module, providing access to the internet and allowing communication with other devices wirelessly.
- NoSQL Database: A type of database that provides a mechanism for storage and retrieval of data that is modelled in means other than the tabular relations used in relational databases.
- Realtime Database: A cloud-hosted database provided by Firebase that stores data as JSON and synchronizes it across connected clients in real-time.
- Serial Communication: A method of transmitting data between digital devices in which the bits of a byte are sent sequentially over a single wire.
- Soil Moisture Sensor: A sensor used to measure the moisture content of soil, providing data on soil hydration levels.
- Temperature Sensor: A sensor used to measure the ambient temperature of the environment.
- WebSocket: A communication protocol that provides full-duplex communication channels over a single TCP connection, enabling interaction between a web browser (or other client application) and a web server with lower overheads.

5. References

[1] IBM, "UML - Basics":http://www.ibm.com/developerworks/rational/library/769.html. [Accessed 29-Feb-2024].

[2] “ACM Code of Ethics and Professional Conduct": https://www.acm.org/code-of-ethics [Accessed 29-Feb-2024]