NEXUS

Stay Connected, Stay in Control

Team Members:

Mohamed Mohamed Elsayed Hassan
Mahmoud Mohamed Elsayed Mohamed
Mostafa Hassan Abd-El Wahab Hussain
Ahmed Hamdi Fahmi Abd-Elkader
Mohamed Ayman Mohamed Attia
Ahmed Abdo Ali Mohamed
Mohamed Ali Mohamed Abd-Elaziz
Ziad Hammad Wasfy Mohamed
Mohamed Ahmed Fouad Youssef
Ahmed Mohamed Ibrahim Hassan

Supervised By:

Dr. Mohamed khairy Shahin

ACKNOWLEDGMENT

The earlier five years were a long journey of education, that only gains its full meaning when we can change the world to a better place through acting upon the knowledge we gained in this journey, only then we can say that we deserved being called engineers. This book is a manual for a graduation project that aimed to give a hand in taking care of our children. hoping it can really make the world, or even a small part of it, a safer place.

"THE GREAT AIM OF EDUCATION IS NOT KNOWLEDGE BUT ACTION." *-Herbert Spencer*

To our families and friends for their endless support and motivation, and finally to **Dr. Mohamed Khairy Shahin**, our supervisor, for their continuous guidance and infinite belief in what we can do.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Project overview....16

1.2 Block diagram....17

2. Hardware Components

2.1 ESP32....18

2.2 ESP32 PERIPHERALS....19

2.3 PN532 NFC Module....25

2.4 GPS MODULE....30

2.5 SIM800L GSM/GPRS Module....34

2.6 ESP32 libraries....38

3. Software Components

3.1 Mobile development....42

3.2 Flutter architecture....43

3.3 Pros & Cons of flutter framework....45

3.4 FLUTTER Dart....46

3.5 Flutter state management....49

3.6 Flutter packages....50

3.7 Connecting Firebase to mobile application....54

4. Cloud FireStore

4.1 Key capabilities....58

4.2 How does it work?....59

4.3 Firebase used services in mobile application....65

4.4 Firebase Packages....68

5. System Architecture

5.1 Main system components....69

5.2 Main devices....70

5.3 System diagrams

5.3.1 Use case diagram....81

5.3.2 Data flow diagram level 0....83

5.3.3 Parent activity diagram....85

5.3.4 Teacher activity diagram.....87
5.3.5 Canteen worker activity diagram.....89
5.3.6 ERD diagram.....91
6 Mobile Application
6.3 Parent Interface.....93
6.4 Teacher Interface.....104
6.5 Allergens research.....110
6.6 Canteen worker Interface.....119
7 Appendices
7.3 Microcontroller Code
7.1.1 Bracelet Device Code.....126
7.1.2 Canteen Device Main Code.....132
7.1.3 Gate Device Main Code.....133
7.2 **Mobile Application Code**
7.2.1 Main code.....135
7.2.2 Parent Home Screen Code.....138
7.2.3 Teacher Home Screen Code.....150
7.2.4 Canteen Home Screen Code.....153
8 Future Work
.....165
9 conclusion
.....167
10 References
.....168

Table of Figures

Fig 1.1 17
Fig 2.1 18
Fig 2.3 19
Fig 2.4 26
Fig 2.5 27
Fig 2.6 28
Fig 2.7 29
Fig 2.8 30
Fig 2.9 30
Fig 2.10 31
Fig 2.11 32
Fig 2.12 33
Fig 2.13 34
Fig 2.14 35
Fig 2.15 36
Fig 2.16 37
Fig 3.1 42
Fig 3.2 43
Fig 3.3 46
Fig 3.4 54
Fig 3.5 55
Fig 3.6 55
Fig 3.7 56
Fig 3.8 56
Fig 4.1 59
Fig 4.2 60
Fig 4.3 61
Fig 4.4 61
Fig 4.5 62
Fig 4.6 62

Fig 4.7.....63
Fig 4.8.....63
Fig 4.9.....64
Fig 4.10.....64
Fig 5.1.....70
Fig 5.2.....73
Fig 5.3.....74
Fig 5.4.....76
Fig 5.5.....77
Fig 5.6 78
Fig 5.7.....79
Fig 5.8.....80
Fig 5.9.....81
Fig 5.10.....83
Fig 5.11.....85
Fig 5.12.....87
Fig 5.13.....89
Fig 5.14.....91
Fig 6.1.....93
Fig 6.2.....94
Fig 6.3.....94
Fig 6.4.....95
Fig 6.5.....95
Fig 6.6.....96
Fig 6.7.....96
Fig 6.8.....97
Fig 6.9.....98
Fig 6.10.....98
Fig 6.11.....99
Fig 6.12.....99
Fig 6.13.....100
Fig 6.14.....100

Fig 6.15.....101
Fig 6.16, Fig 6.17.....102
Fig 6.18.....103
Fig 6.19.....104
Fig 6.20, Fig 6.21.....105
Fig 6.22, Fig 6.23.....106
Fig 6.24, Fig 6.25.....107
Fig 6.26.....108
Fig 6.27.....108
Fig 6.28.....109
Fig 6.29.....109
Fig 6.30.....115
Fig 6.31.....116
Fig 6.32 117
Fig 6.33.....118
Fig 6.34.....119
Fig 6.35.....120
Fig 3.36.....120
Fig 6.37.....121
Fig 6.38 121
Fig 6.39 122
Fig 6.40.....122
Fig 6.41.....123
Fig 6.42.....123
Fig 6.43.....124
Fig 6.44.....124
Fig 6.45 125

1.INTRODUCTION

The Smart School System revolutionizes the traditional school environment by incorporating advanced software and hardware technologies to address critical challenges faced by schools today. This comprehensive application project aims to create a smarter and safer school experience for students, parents, and teachers. By leveraging innovative solutions, the Smart School System streamlines processes, enhances security, and improves communication, ultimately enhancing the overall educational ecosystem.

Schools have long grappled with the task of ensuring student safety and providing a conducive learning environment. Parents often worry about their children's whereabouts during school hours, especially in emergencies. Moreover, managing payments for snacks and meals within the school premises can be cumbersome and time-consuming. Additionally, maintaining effective communication channels between teachers and parents can be challenging, leading to missed opportunities for collaboration and support.

The Smart School System has been meticulously designed to tackle these prevalent issues head-on. By harnessing the power of technology, this project offers a comprehensive solution that caters to the unique needs of schools and their stakeholders.

The core feature of the Smart School System revolves around the integration of NFC (Near Field Communication) bracelets with GPS tracking capabilities. Students wear these bracelets, enabling parents and school staff to stay updated on their location in real-time. Whether it's during regular school hours or during field trips, parents can have peace of mind knowing where their child is at all times. This enhances student safety, reduces concerns, and empowers parents with the ability to make informed decisions regarding their child's well-being.

Furthermore, the NFC functionality of the bracelets allows for seamless and secure payment transactions within the school premises. Parents can conveniently connect their credit cards to the NFC chip, eliminating the need for students to carry cash. This cashless payment system not only ensures a hassle-free experience for parents and students but also provides a transparent record of transactions. Additionally, parents can specify any allergies their child may have through the application. When a student attempts to purchase a snack containing allergenic ingredients, the payment is automatically declined, preventing potential health risks.

Teachers play a crucial role in the Smart School System, as they can effortlessly manage attendance and communicate with parents. The dedicated teacher section

within the application streamlines the attendance-taking process, reducing paperwork and administrative burden. Teachers can also share important updates, notes, and grades with parents through the application, fostering better collaboration and fostering a stronger parent-teacher relationship.

To ensure data security, scalability, and accessibility, the Smart School System utilizes Cloud Firestore, a cloud-based storage solution. This enables seamless data synchronization across multiple devices and provides a reliable and robust infrastructure for storing student information, transaction records, and communication logs.

By implementing the Smart School System, schools can create a forward-thinking educational environment that prioritizes safety, convenience, and effective communication. Students can focus on their studies, parents can have peace of mind, and teachers can streamline administrative tasks, leading to an overall improved educational experience. The Smart School System sets the stage for a more connected, efficient, and secure school ecosystem, enabling students to thrive and reach their full potential.

The implementation of the Smart School System involves a collaborative effort between software and hardware components, seamlessly integrating technology into the school environment. The project utilizes the ESP32 microcontroller, which serves as the central hardware unit responsible for communication, data processing, and interaction with various peripherals.

The NFC (Near Field Communication) bracelets worn by students play a pivotal role in the system. These bracelets incorporate a GPS module and an NFC chip, enabling multiple functionalities. The GPS module tracks the students' location, providing accurate real-time updates to parents and school staff.

The NFC chip facilitates secure and convenient payment transactions within the school premises, connected to parents' credit cards. This combination of hardware components ensures a reliable and efficient system for both student tracking and payment management.

The software aspect of the project encompasses a user-friendly mobile application. This application serves as the interface for parents, teachers, and canteen workers to interact with the Smart School System. Parents can access the application to receive

location updates, manage payment settings, and communicate with teachers. Teachers can take attendance, share important information, and upload attachments such as grades and progress reports. Canteen workers have access to a dedicated section where they can view student allergies, check payment balances, and process transactions. The application is designed to be intuitive and accessible, ensuring a seamless user experience for all stakeholders.

To ensure the security and reliability of the data, the Smart School System utilizes Cloud Firestore, a cloud-based storage solution provided by Google. Cloud Firestore offers robust data synchronization, scalability, and real-time updates across multiple devices. All student information, transaction records, attendance data, and communication logs are securely stored in the cloud, accessible only to authorized users. This ensures data integrity and provides a centralized platform for efficient data management.

The Smart School System can be customized and tailored to the specific needs and requirements of each school. It offers flexibility for schools to integrate additional features or modules as deemed necessary, allowing for further enhancements and expansions in the future.

The implementation of the Smart School System brings forth numerous benefits and positive impacts on various stakeholders within the educational ecosystem.

For parents, the system provides peace of mind, knowing that they can track their child's location and receive real-time updates. Parents can be promptly informed about their child's arrival and departure from school, ensuring their safety and reducing concerns. The cashless payment system offers convenience and transparency, eliminating the need for students to carry cash and providing an accurate record of transactions. Parents can also easily communicate with teachers, receive important updates, and stay actively involved in their child's education.

Teachers benefit from the Smart School System by streamlining administrative tasks, such as attendance management and communication with parents. By automating attendance taking, teachers can save valuable instructional time and allocate more attention to classroom activities. The ability to share grades, progress reports, and

attachments with parents fosters effective communication and collaboration, enabling a stronger partnership between teachers and parents.

Students experience a safer and more efficient school environment with the implementation of the Smart School System. The GPS tracking feature ensures their safety and provides a sense of security to students and their families. The cashless payment system simplifies the process of purchasing snacks and meals, allowing students to focus on their studies and reducing the risk of carrying cash. Moreover, the system's integration promotes a sense of responsibility and accountability among students, instilling good financial habits and awareness of their dietary restrictions.
From a school management perspective, the Smart School System offers streamlined processes, reducing paperwork and administrative burdens. The centralized data storage and management in the cloud enhance data security, accessibility, and scalability. The system provides valuable insights into student attendance patterns, transaction records, and communication logs, facilitating informed decision-making and efficient resource allocation.

In conclusion, the implementation of the Smart School System brings about a transformational shift in the school environment. By leveraging technology, it enhances student safety, improves communication between stakeholders, simplifies payment management, and streamlines administrative processes. This innovative solution empowers parents, supports teachers, and creates a conducive learning environment for students. The Smart School System sets the stage for a future-ready education system, paving the way for academic excellence and holistic development.

Challenges and Considerations,While the implementation of the Smart School System brings numerous benefits, it is essential to address potential challenges and considerations to ensure its successful integration into the school environment.
One primary consideration is the need for robust cybersecurity measures. As the system deals with sensitive student information, payment details, and communication data, it is crucial to implement strong security protocols. Encryption techniques, secure

communication channels, and regular system audits must be in place to safeguard the integrity and confidentiality of the data.
Another consideration is the need for comprehensive training and support for all stakeholders. Teachers, parents, and canteen workers must receive proper training on using the mobile application, understanding the system's features, and adhering to data protection protocols. Ongoing technical support should be available to address any issues or concerns that may arise.

Furthermore, integration with existing school systems and infrastructure must be carefully planned. The Smart School System should seamlessly integrate with the school's existing management systems, such as student databases and attendance tracking systems. Compatibility and interoperability between different hardware components and software modules must be ensured to avoid any disruptions or inefficiencies in the school's daily operations.
Financial considerations also come into play. The implementation of the Smart School System requires an initial investment in hardware, software development, and infrastructure setup. Schools must carefully evaluate the return on investment and long-term sustainability of the system. It is crucial to assess the cost-effectiveness, benefits, and potential savings in administrative processes and improved parent-school communication.
Future Expansion and Possibilities, The Smart School System serves as a foundation for future expansions and possibilities. It can be continuously enhanced and adapted to meet evolving educational needs and technological advancements. Here are some potential areas for future development:

1. Integration with Learning Management Systems: The Smart School System can be integrated with learning management platforms to enable seamless access to course materials, assignments, and assessments. This integration would provide a comprehensive digital learning environment for students, teachers, and parents.
2. Smart Classroom Solutions: The system can be extended to incorporate smart classroom solutions such as interactive whiteboards, digital content sharing,

and student engagement tools. This integration would create an immersive learning environment and facilitate collaborative learning experiences.

3. Enhanced Parent-Teacher Communication: The system can include features like video conferencing, real-time messaging, and automated notifications to strengthen parent-teacher communication. This would foster greater involvement and engagement from parents in their child's educational journey.
4. Analytics and Data Insights: By leveraging the data collected through the Smart School System, advanced analytics and data insights can be generated. These insights can help identify patterns, trends, and areas for improvement in student performance, attendance, and overall school operations.
5. IoT Integration: The Smart School System can be expanded to incorporate Internet of Things (IoT) devices such as smart sensors for temperature monitoring, energy management, and security. This integration would further optimize school operations and ensure a safe and sustainable learning environment.

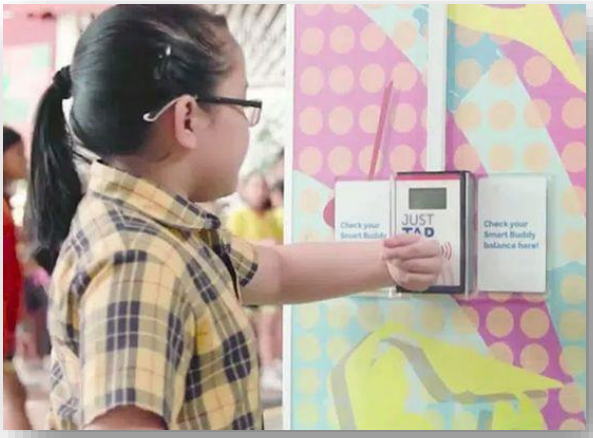
In conclusion, the implementation of the Smart School System opens up a world of possibilities for transforming the traditional school ecosystem. With careful planning, consideration of challenges, and a commitment to ongoing innovation, the system can serve as a catalyst for educational excellence, parent-school collaboration, and student success. It marks a significant step towards building smarter schools that harness the power of technology to create an enriching and empowering learning environment for all.

The NEXUS project is a groundbreaking initiative that seeks to revolutionize the way schools operate and empower parents, teachers, and students with advanced technology. With its innovative blend of hardware and software components, the

NEXUS system provides a comprehensive solution that addresses key challenges in the school environment and enhances the overall educational experience.

At the heart of the NEXUS project is the wearable NFC bracelet with GPS. This state-of-the-art bracelet serves as a personal identifier for students, allowing parents to easily track their child's location within the school premises. Through the dedicated mobile application, parents receive real-time updates on their child's whereabouts, providing them with peace of mind and a heightened sense of security. Whether it's ensuring their child has safely arrived at school or keeping track of their movements during breaks, parents can stay connected and informed throughout the day.

One of the notable features of the NFC bracelet is its integration with the school's canteen system. By incorporating an NFC chip, students can conveniently make payments for snacks and meals directly from their bracelet. Parents have the option to connect their credit card to the NFC chip, enabling a seamless and cashless transaction process. This not only eliminates the need for students to carry physical cash but also provides parents with a transparent view of their child's spending habits, promoting financial responsibility.



The NEXUS project goes beyond location tracking and payment capabilities. It addresses an essential aspect of student well-being—food allergies. With the dedicated mobile application, parents can input their child's allergies, allowing the system to automatically verify the ingredients of each purchased item. If a student attempts to buy a snack containing allergens, the payment is declined, preventing any potential health risks and ensuring a safe and inclusive environment for all students.

Teachers also benefit from the NEXUS system through the dedicated teacher section in the mobile application. This section allows teachers to efficiently take attendance by scanning the NFC bracelets, saving valuable class time. Moreover, teachers can provide personalized updates and notes to parents about their child's progress, behavior, or any specific concerns. This seamless communication channel fosters collaboration

between teachers and parents, facilitating a holistic approach to education and supporting each student's individual needs.

To enable efficient data management and accessibility, the NEXUS project employs Cloud Firestore, a cloud-based storage solution. This technology ensures that student information, attendance records, allergy data, and communication logs are securely stored and readily accessible to authorized users. Cloud Firestore not only centralizes the data but also enables real-time synchronization across multiple devices, ensuring that stakeholders have the most up-to-date information at their fingertips.

In terms of hardware, the NEXUS project utilizes the ESP32 microcontroller, NFC chip, GPS module, and NFC reader. The ESP32 acts as the central hub, facilitating seamless communication between the various hardware components and the software application. The NFC chip enables secure payments and identification, while the GPS module provides accurate location tracking. The NFC reader is employed in the gate device, allowing swift and reliable detection and authentication of student entry and exit.

In summary, the NEXUS project represents a significant leap forward in school management systems, leveraging cutting-edge technology to address critical aspects of safety, convenience, and communication. By seamlessly integrating hardware and software components, this project empowers parents, teachers, and students with a comprehensive and user-friendly ecosystem. The NEXUS system not only enhances student security and provides parents with real-time updates but also streamlines administrative tasks for teachers, ultimately fostering an optimal learning environment for all stakeholders involved.

1.1 Project Overview

The smart school system consists of a combination of hardware and software components that work together to provide a seamless experience for students, teachers, and parents.

The system is based on the use of NFC bracelets integrated with GPS that are worn by students.

Additionally, the bracelet contains a GPS module that allows parents to always monitor the location of their child, ensuring their safety and security.

The system also includes a canteen worker section responsible for handling the sale of snacks, and parents can add information to the application regarding any allergies their child may have. If a student attempts to buy a snack that contains ingredients, they are allergic to, the payment is automatically declined, ensuring their safety and wellbeing.

Furthermore, the smart school system includes a teacher section that allows them to take attendance and add notes about individual students and posting their grades, which parents can access through the application. This feature enhances communication between teachers and parents, providing them with valuable information about their child's academic performance.

The system is built on a combination of hardware components, including the ESP32 microcontroller, NFC chip, GPS module, and NFC reader. The software components of the system include a mobile application for parents, teachers and canteen workers. All data collected by the system is stored in a cloud Firestore database, providing easy access to information for all users.

In conclusion, the smart school system is a revolutionary technology that can improve the educational experience for students, parents, and teachers. It provides enhanced security and safety for students, simplifies the payment process for snacks, and improves communication between teachers and parents. The system is built on a combination of hardware and software components that work together seamlessly to provide a valuable tool for educational institutions.

1.2 Block diagram

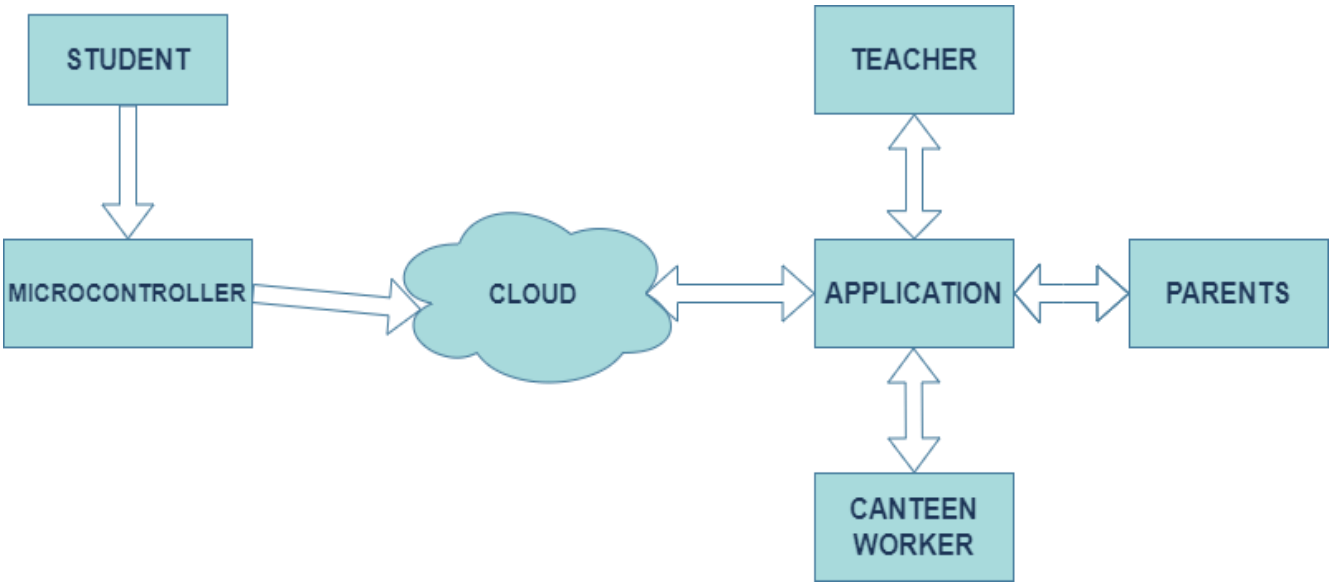


Figure 1.1

Bracelets worn by students communicate with the microcontroller **board.** The data collected from the GPS & NFC module is then sent to the mobile application via cloud fireStore.

The mobile application is responsible for displaying the exact location of student to their parents and allowing the parents to recharge the student wallet by their credit card, where the students can pay for their snacks using the NFC chip.

Additionally, the canteen worker is integrated in the App. Allowing parents to input any allergies their child may have, and the teacher section allows for attendance tracking and communicating with parents.

2.0 HARDWARE COMPONENTS

2.1 ESP32 MICROCONTROLLER

First, we talk about the main component which is ESP 32, here we explain how it operates and how it works in the project **environment.**

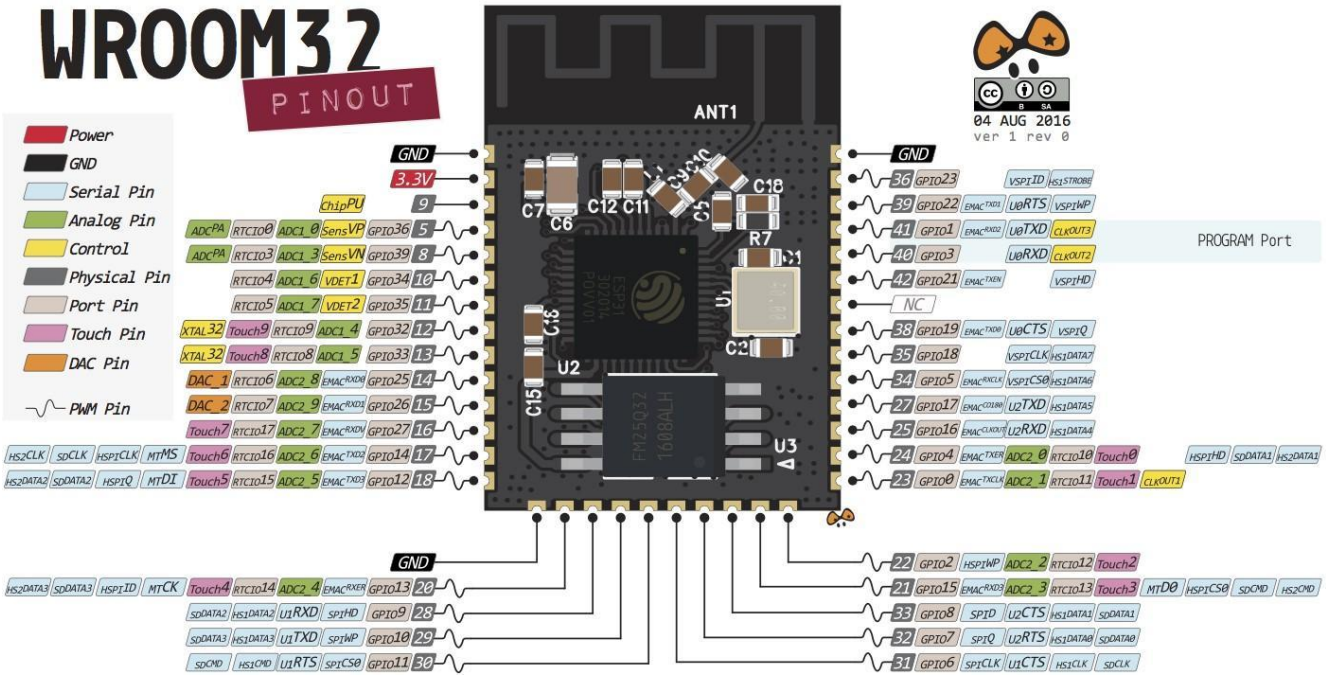


Figure 2.1

The ESP32 chip comes with 48 pins with multiple functions. Not all pins are exposed in all ESP32 development boards, and some pins cannot be used.

There are many questions on how to use the ESP32 GPIOs. What pins should you use? What pins should you avoid using in your projects? This post aims to be a simple and easy-to-follow reference guide for the ESP32 GPIOs.

The figure above (fig.1) illustrates the ESP-WROOM-32 pinout. You can use it as a reference if you're using an ESP32 bare chip to build a custom board.

All GPIOs are accessible in all development boards, but each specific GPIO works in the same way regardless of the development board you're using. If you're just getting started with the ESP32, we recommend reading our guide: Getting Started with the ESP32 Development Board.

2.2 ESP32 PERIPHERALS

The ESP32 peripherals include:

- 18 Analog-to-Digital Converter (ADC) channels
- 3 SPI interfaces
- 3 UART interfaces
- 2 I2C interfaces
- 16 PWM output channels
- 2 Digital-to-Analog Converters (DAC)
- 2 I2S interfaces
- 10 Capacitive sensing GPIOs

The ADC (analog to digital converter) and DAC (digital to analog converter) features are assigned to specific static pins. However, you can decide which pins are UART, I2C, SPI, PWM, etc. – you just need to assign them in the code. This is possible due to the ESP32 chip's multiplexing feature.

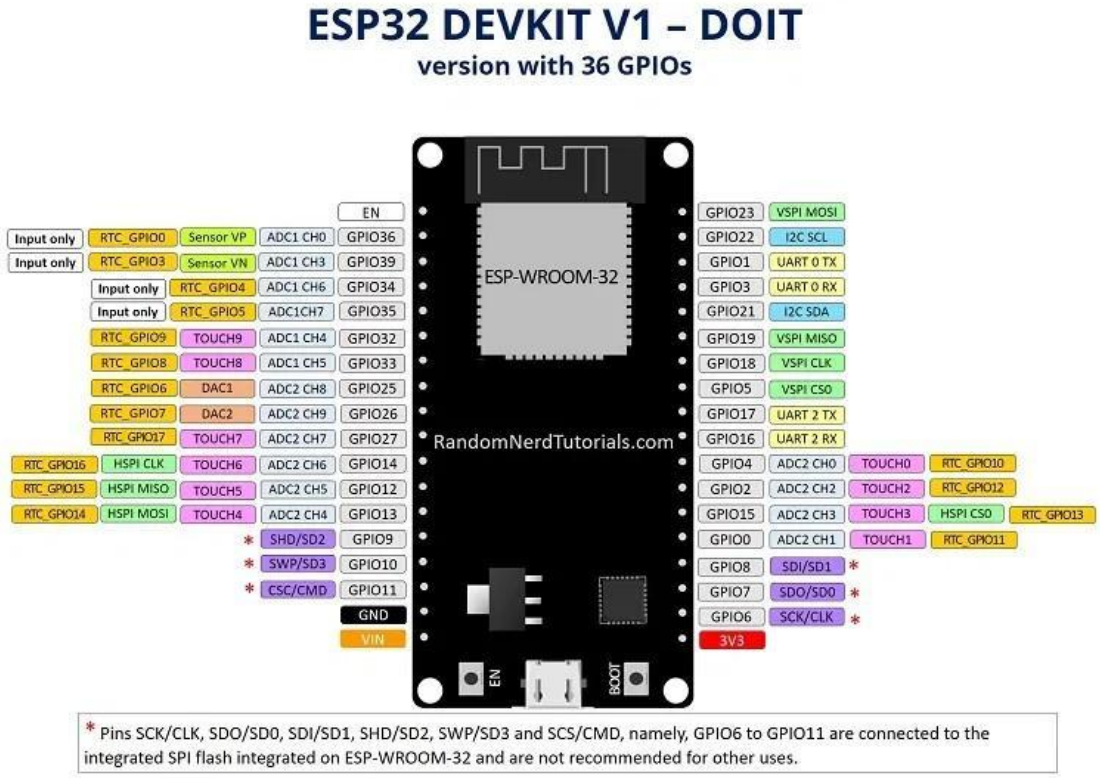


Figure 2.2

Although you can define the pins properties on the software, there are pins assigned by default as shown in the above figure(fig.2),
(This is an example for the ESP32 DEVKIT V1 DOIT board with 36 pins – the pin location can change depending on the manufacturer).

GPIO	Input	Output	Notes
0	pulled up	OK	outputs PWM signal at boot
1	TX pin	OK	debug output at boot
2	OK	OK	connected to on-board LED
3	OK	RX pin	HIGH at boot
4	OK	OK	
5	OK	OK	outputs PWM signal at boot
6	x	x	connected to the integrated SPI flash
7	x	x	connected to the integrated SPI flash
8	x	x	connected to the integrated SPI flash
9	x	x	connected to the integrated SPI flash
10	x	x	connected to the integrated SPI flash
11	x	x	connected to the integrated SPI flash
12	OK	OK	boot fail if pulled high
13	OK	OK	
14	OK	OK	outputs PWM signal at boot
15	OK	OK	outputs PWM signal at boot
16	OK	OK	
17	OK	OK	

18	OK	OK	
19	OK	OK	
21	OK	OK	
22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK		input only
35	OK		input only
36	OK		input only
39	OK		input only

Figure 2.3

Additionally, there are pins with specific features that make them suitable or not for a particular project.

The above table (fig.3) shows what pins are best to use as inputs, outputs, and which ones you need to be cautious of.

The pins highlighted in green are OK to use.

The ones highlighted in yellow are OK to use, but you need to pay attention because they may have unexpected behavior mainly at boot. The pins highlighted in red are not recommended to be used as inputs or outputs.

Continue reading for a more detailed and in-depth analysis of the ESP32 GPIOs and its functions.

Input only pins:

GPIOs 34 to 39 are GPIs – input only pins. These pins don't have internal pull-up or pull-down resistors. They can't be used as outputs, so use these pins only as inputs:

- GPIO 34
- GPIO 35

- GPIO 36
- GPIO 39

SPI flash integrated on the ESP-WROOM-32:

GPIO 6 to GPIO 11 are exposed in some ESP32 development boards. However, these pins are connected to the integrated SPI flash on the ESP-WROOM-32 chip and are not recommended for other uses. So, don't use these pins in your projects:

- GPIO 6 (SCK/CLK)
- GPIO 7 (SDO/SD0)
- GPIO 8 (SDI/SD1)
- GPIO 9 (SHD/SD2)
- GPIO 10 (SWP/SD3)
- GPIO 11 (CSC/CMD)

Capacitive touch GPIOs:

The ESP32 has 10 internal capacitive touch sensors. These can sense variations in anything that holds an electrical charge, like human skin. So, they can detect variations induced when touching the GPIOs with a finger. These pins can be easily integrated into capacitive pads and replace mechanical buttons. The capacitive touch pins can also be used to wake up the ESP32 from deep sleep.

Those internal touch sensors are connected to these GPIOs:

- T0 (GPIO 4)
- T1 (GPIO 0)
- T2 (GPIO 2)
- T3 (GPIO 15)
- T4 (GPIO 13)
- T5 (GPIO 12)
- T6 (GPIO 14)
- T7 (GPIO 27)
- T8 (GPIO 33)
- T9 (GPIO 32)

I2C:

The ESP32 has two I2C channels and any pin can be set as SDA or SCL. When using the ESP32 with the Arduino IDE, the default I2C pins are:

GPIO 21 (SDA)

GPIO 22 (SCL)

If you want to use other pins when using the wire library, you just need to call: Wire. Begin (SDA, SCL).

SPI:

By default, the pin mapping for SPI is:

Analog to Digital Converter (ADC):

The ESP32 has 18 x 12 bits ADC input channels (while the ESP8266 only has 1x 10 bits ADC). These are the GPIOs that can be used as ADC and respective channels:

- ADC1_CH0 (GPIO 36)
- ADC1_CH1 (GPIO 37)
- ADC1_CH2 (GPIO 38)

SPI	MOSI	MISO	CLK	CS
VSPI	GPIO 23	GPIO 19	GPIO 18	GPIO 5
HSPI	GPIO 13	GPIO 12	GPIO 14	GPIO 15

- ADC1_CH3 (GPIO 39)
- ADC1_CH4 (GPIO 32)
- ADC1_CH5 (GPIO 33)
- ADC1_CH6 (GPIO 34)
- ADC1_CH7 (GPIO 35)
- ADC2_CH0 (GPIO 4)
- ADC2_CH1 (GPIO 0)
- ADC2_CH2 (GPIO 2)

- ADC2_CH3 (GPIO 15)
- ADC2_CH4 (GPIO 13)
- ADC2_CH5 (GPIO 12)
- ADC2_CH6 (GPIO 14)
- ADC2_CH7 (GPIO 27)
- ADC2_CH8 (GPIO 25)
- ADC2_CH9 (GPIO 26)

ADC2 pins cannot be used when Wi-Fi is used.
So, if you're using Wi-Fi and you're having trouble getting the value from an ADC2 GPIO, you may consider using an ADC1 GPIO instead. That should solve your problem.
The ADC input channels have a 12-bit resolution. This means that you can get analog readings ranging from 0 to 4095, in which 0 corresponds to 0V and 4095 to 3.3V. You can also set the resolution of your channels on the code and the ADC range.

Digital to Analog Converter (DAC):
There are 2 x 8 bits DAC channels on the ESP32 to convert digital signals into analog voltage signal outputs. These are the DAC channels:
DAC1 (GPIO25)
DAC2 (GPIO26)
PWM
The ESP32 LED PWM controller has 16 independent channels that can be configured to generate PWM signals with different properties. All pins that can act as outputs can be used as PWM pins (GPIOs 34 to 39 can't generate PWM).
To set a PWM signal, you need to define these parameters in the code:

- Signal's frequency.
- Duty cycle.
- PWM channel.
- GPIO where you want to output the signal.

Some GPIOs change their state to HIGH or output PWM signals at boot or reset. This means that if you have outputs connected to these GPIOs you may get unexpected results when the ESP32 resets or boots.

- GPIO 1

- GPIO 3
- GPIO 5
- GPIO 6 to GPIO 11 (connected to the ESP32 integrated SPI flash memory – not recommended to use).
- GPIO 14
- GPIO 15

Enable (EN):

Enable (EN) is the 3.3V regulator's enable pin. It's pulled up, so connect to ground to disable the 3.3V regulator. This means that you can use this pin connected to a pushbutton to restart your ESP32, for example.

GPIO current drawn

The absolute maximum current drawn per GPIO is 40mA according to the "Recommended Operating Conditions" section in the ESP32 datasheet.

We **talk** about the microcontroller, how it works and the peripherals, which are used to communicate with other devices.

In this project we use the ESP32 to communicate with NFC Reader PN532 to read the tag from the students and send the data through the ESP32 Wi-Fi to the fireStore, which we take about it in the next chapter from this documentation and will discuss how esp32 sent the date to the firestore.

Now we will talk about another part from this project the NFC Reader PN532.

2.3 PN532 NFC MODULE

Introduction to NFC:

Before understanding PN532 as a whole, let's learn what is NFC technology and how does it work.

What is NFC? Near Field Communication (NFC) is a technology standard based on Radio Frequency Identification (RFID), transmitting information wirelessly over short distances.

NFC technology makes life easier and more convenient for consumers around the world by making it simpler to make transactions, exchange digital content, and

connect electronic devices with a touch. NFC is compatible with hundreds of millions of contactless cards and readers already deployed worldwide.

How NFC Works?

NFC operates on the principle of inductive coupling, at least for short-range implementations. This essentially involves the reader device generating a magnetic field by passing an electric current through a coil.

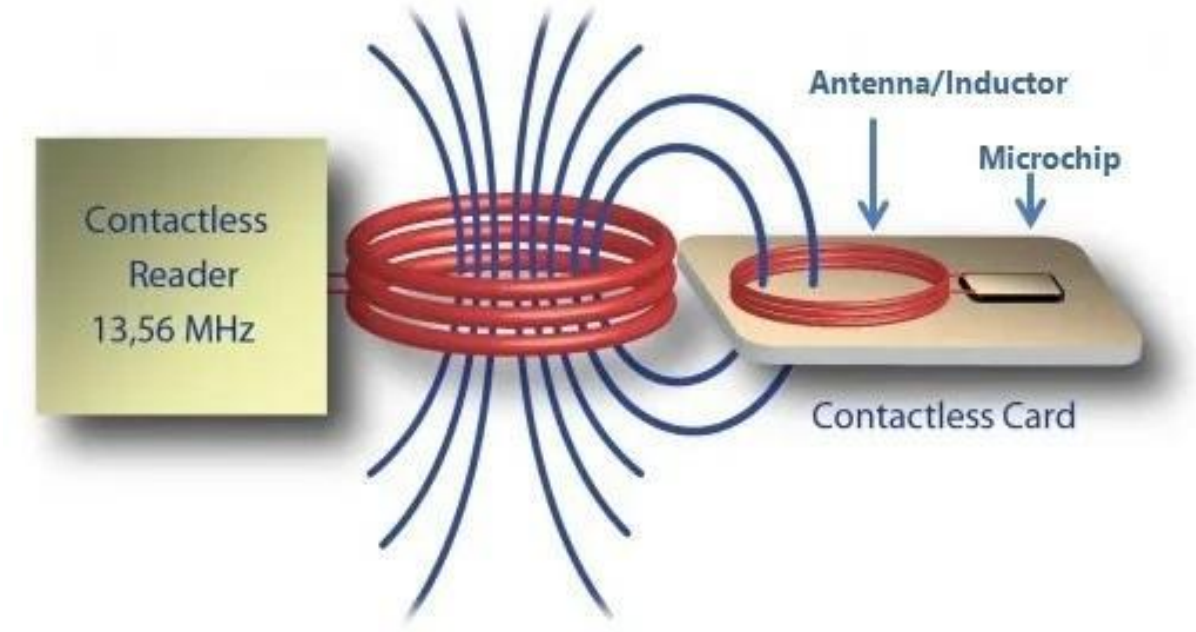


Figure 2.4

When a tag (with its own coil) is brought nearby, the field induces an electric current within the tag — sans any wires or even physical contact. Then, once the initial handshake is complete, any stored data on the tag is wirelessly transmitted to the reader.

Difference Between RFID & NFC:

The key distinction between RFID and NFC lies in their transmission ranges. RFID is often used over longer distances. For example, some regions automatically collect road tolls through RFID. Communication can take place over even longer distances if the RFID tag is equipped with a power source.

Figure 2.5

NFC, however, only has a maximum range of a few centimeters, at most. And in most smartphone-related applications, you'll find that the software will only initiate communication if there's physical contact. NFC devices can act as either a reader or tag. This bidirectional capability allows you to use one piece of hardware, such as your smartphone for all kinds of different applications.

Main applications of NFC:

- Mobile payments: Apple Pay NFC, Google Wallet.
- Tap-to-Pair; Pairing Bluetooth devices just by tapping, instead of manual entering.
- Embedding digital experience in physical products; NFC's small form factor allows for it to be embedded into physical products, enabling a unique experience when interacted.
- Security
- Password replacement
- Product Authentication

NFC Reader PN532 module :

This module is built around NXP PN532, and the maker breaks out almost all the I/O pins of the NXP532 chip on this little module. This compatible module has the following features:

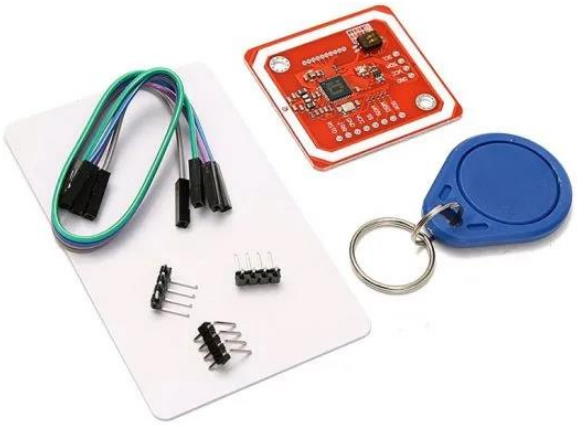


Figure 2.6

- Supports II2, SPI, and high-speed UART (HSU).
- RFID reader/writer mode support for:
- Mi fare 1K, 4K, Ultralight, and DesFire cards.
- ISO/IEC 14443-4 cards such as CD97BX, CD light, DesFire, and P5CN072 (SMX).
- Innovision Jewel cards such as the IRT5001 card.
- FeliCa cards such as RCS_860 and RCS_854.
- Built-in PCB antenna (covered by white paint).
- 5- to 7-cm communication distance.
- On-board level shifter.
- Standard 5-V TTL for I2C/UART and 3.3-V TTL SPI (VCC 3.3 to 5 V).
- Works as an RFID reader/writer.
- Works as a 1443-A card or a virtual card.
- Supports NFC with Android phone.

In the module, the I2C and HSU share the same pins.

The definition of the I2C pins is labeled on top of the breakout board, while the HSU's is labeled at the bottom.

Figure 2.7

Although the HSU mode is configured as the default mode, with a small on-board SMD DIP switch, it becomes easy to change among I2C, SPI, and HSU interfaces. At this part we need to download the PN532 library based on Adafruit_NFCShield_I2C, and in this project we use the SPI peripheral to connect the ESP32 and PN532.

ESP32	PN532
VCC	VCC
GND	GND
18	SCK
19	MISO
23	MOSI
27	SS

2.4 GPS MODULE

Now, we discuss our last used module,The GPS module provides accurate location tracking for the students. It communicates with the ESP32 to transmit real-time location data.

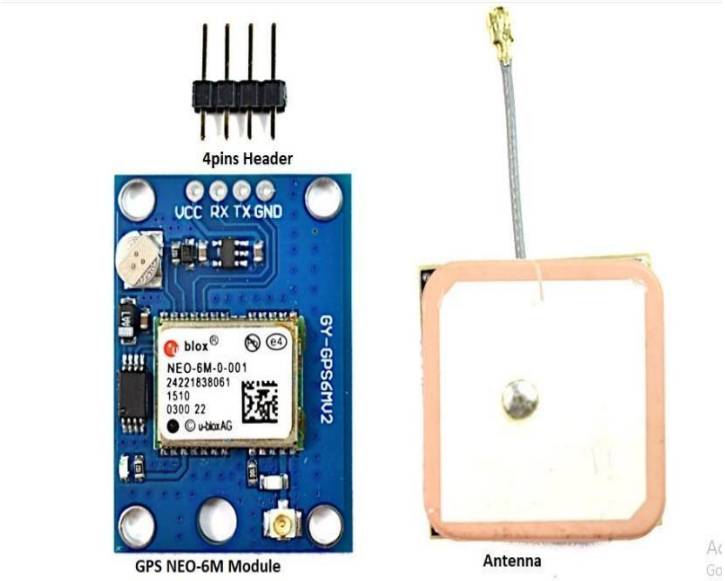


Figure 2.9

GPS receiver specifications:
Basically, GPS (Global Positioning System) receivers work by figuring out how far they are from several satellites. They are pre-programmed to know where the GPS satellites are at any given time and then calculate the location on Earth using the well-known Trilateration process. The more satellites there are above the horizon the more accurately your GPS unit can determine where you are.
This project uses the low power NEO-6M GPS (fig.9) module from u-blox, suitable for battery powered, inexpensive, easy to interface with microcontrollers and insanely popular among hobbyists.

Characteristics:

- Power supply: 2.7V to 3.6V.
- Default baud rate: 9600 bps
- Current consumption: 45mA.
- Up to 22 satellites on 50 channels tracking.
- Up to 5 locations updates a second with 2.5m Horizontal position accuracy.
- Navigation Sensitivity: Up to -161 dBm.
- Navigation Update Rate: 1Hz.
- Comes with an external antenna and built-in EEPROM.
- Interface: RS232 TTL with Serial Baud Rate 4800-230400 (default 9600).
- Operating Temperature: -40°C ~ 85°C.

- Communication Protocol: UBX Binary, RTCM and Standard NMEA output: To know what each data field means in each of these sentences.

NEO-6M GPS Module Pinout:

The NEO-6M GPS module has four pins: GND, TxD, RxD, and VCC. The TxD and RxD pins are used to communicate with the microcontroller.

The NEO-6M GPS module has five major parts on the board, the first major part is the NEO-6M GPS chip in the heart of the PCB.

Next, we have a rechargeable battery and a serial EEPROM module. An EEPROM together with a battery helps retain the clock data, latest position data (GNSS orbit data), and module configuration but it's not meant for permanent data storage.

Without the battery, the GPS always cold starts so the initial GPS lock takes more time. The battery is automatically charged when power is applied and maintains data for up to two weeks without power.

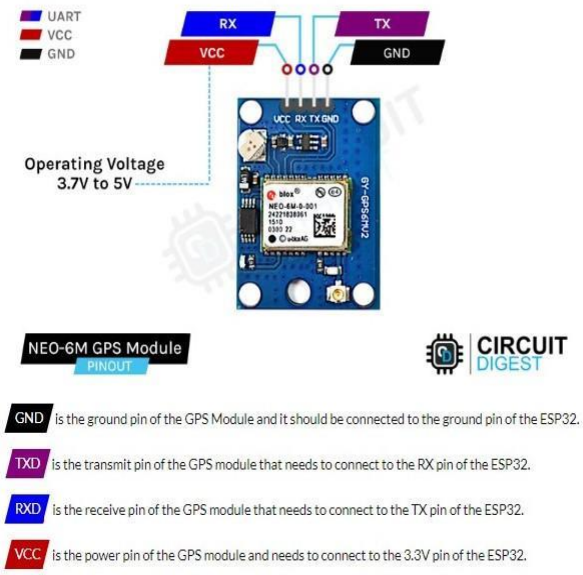


Figure 2.10

Next, we have our LDO, because of the onboard LDO, the module can be powered from a 5V supply. Finally, we have our UFL connector where we need to connect an external antenna for the GPS to properly work.

Circuit Diagram for NEO-6M GSM Module

The circuit for the NEO-6M GPS module is very simple and easy to understand. In this module, all the major work is handled by the GPS module, and we need two or three

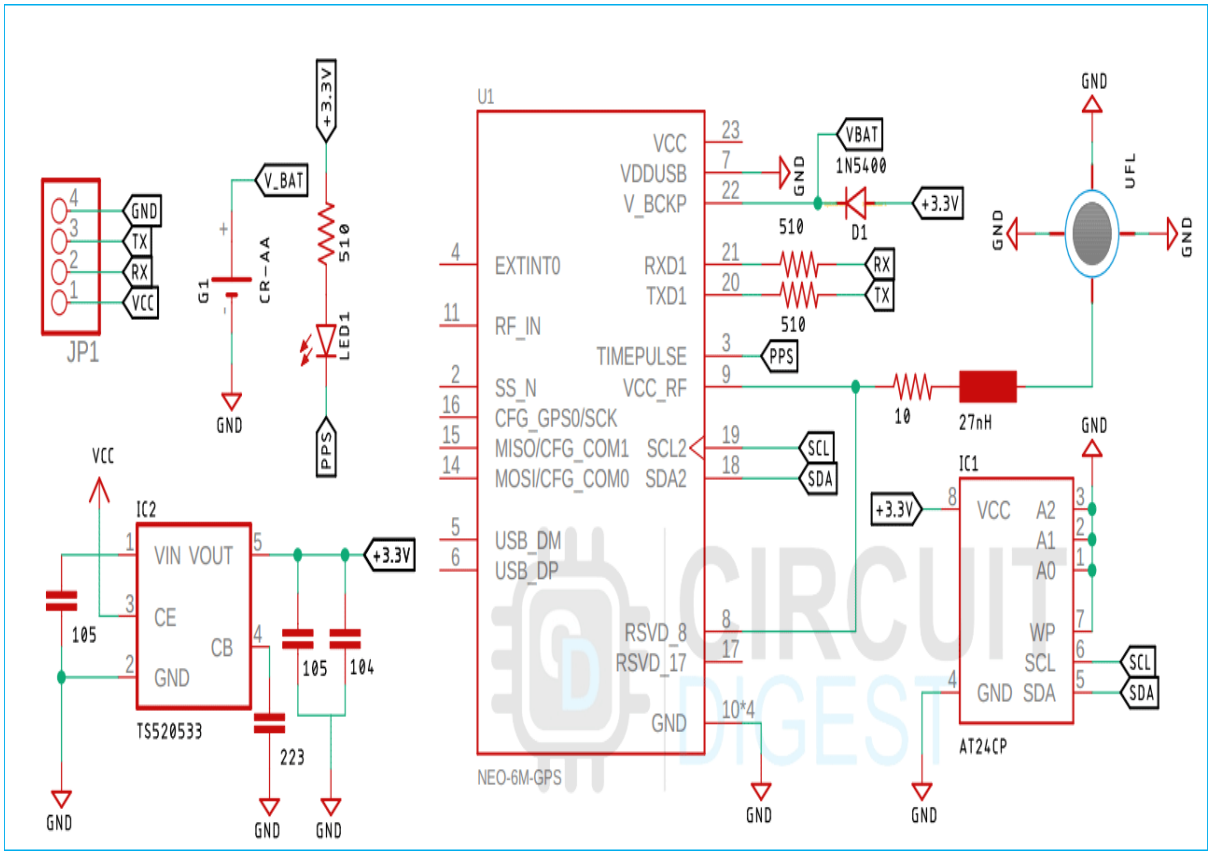


Figure 2.11

more components to make the GPS module work. The complete Schematic of the NEO-6M GPS Module is shown below.

(In the following page ....).

In the schematic, we have a 3.3V voltage regulator that is responsible for converting the input from 5V to 3.3V and we also have an AT24C33 EEPROM and a battery.

EEPROM and Battery together help in retaining battery-backed RAM that can hold clock data and position for a limited period of time. Other than that, we have the UFL

connector in which you need to connect an External Patch Antenna to make the module work.

Hardware Set up:

The GPS module will be powered with a 3.3V power supply. It communicates with the ESP32 via serial communication using the TX and RX pins available on the 4pins header. The following image shows the schematic diagram of the circuit used in this tutorial.

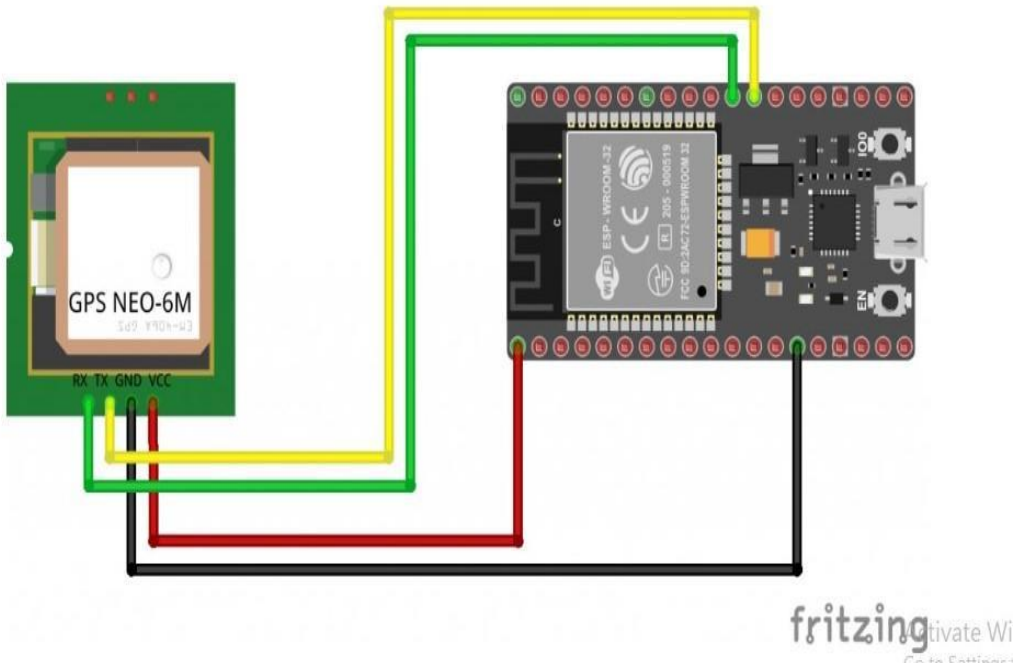


Figure 2.12

- Connect the VCC pin of the GPS module to the ESP32 3.3V pin.
- Connect the GPS ground pin to the ESP32 ground pin.
- Now, connect the RX pin of the GPS module to the TX pin of the ESP32. Similarly, connect the TX pin of the GPS module to the RX pin of the ESP32 (RX and TX pins of the ESP32 are defined in the software).
- Last, connect your ESP32 to the computer through a USB cable. The ESP32 will be powered from the USB 5V.

2.5 SIM800L GSM/GPRS Module

SIM800L GSM/GPRS Module Pinout:

The SIM800L GSM/GPRS module has 12 pins which are NET, VCC, RST, RXD, TXD, GND, SPK-, SPK+, MIC-, MIC+, DTR, RING.

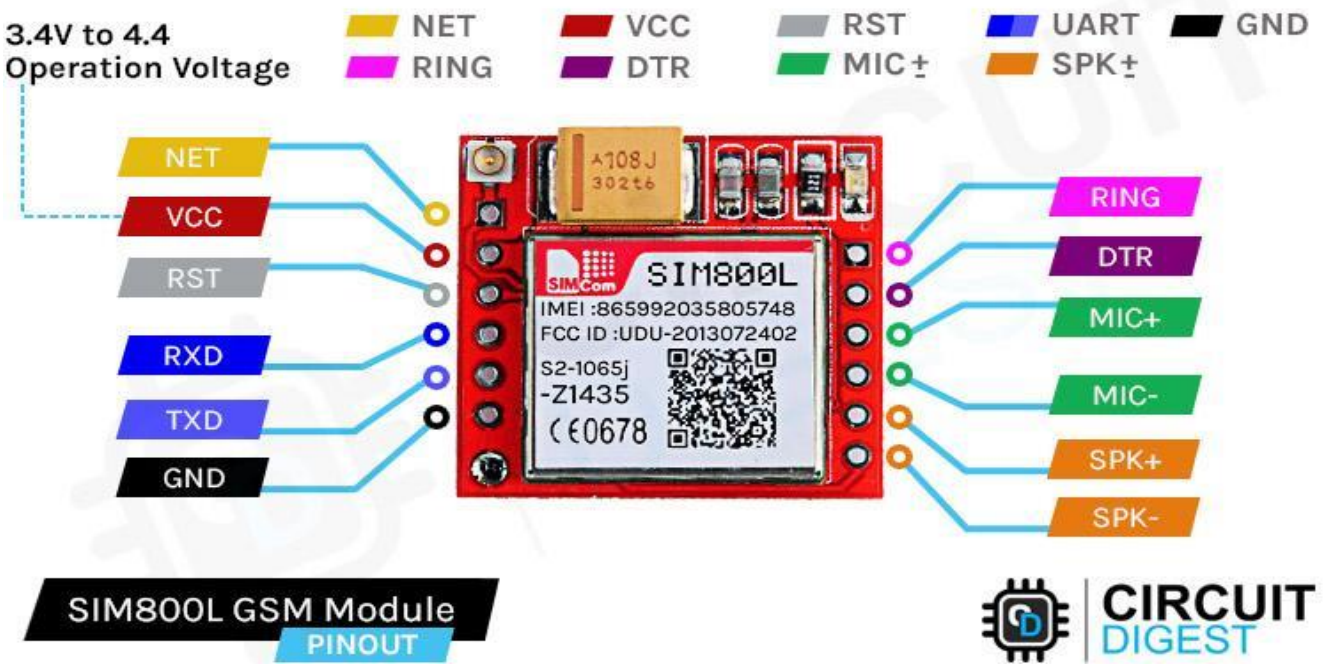


Figure 2.13

A NET is a pin where you can solder the helical antenna that comes with the module.

VCC is the Power supply pin of the module, and it needs to be powered anywhere from 3.4V to 4.4 volts. Connecting this module to a 5V supply will most likely destroy it and if you connect it to 3.3V it won't even run. A lithium battery or a buck converter with 2A current capacity is recommended for this module.

RST is the hard reset pin of the sim800L module. If you are having trouble communicating with this, pull the pin low for 100ms.

RXD is the RX pin for the module used in serial communication.

TXD is the TX pin for this module used in Serial communication.

GND is the Ground pin for this module; connect this pin to the Ground pin of the ESP32.

RING is the ring indicator pin of the module. This pin generally is active high. It will go low for 120ms to indicate incoming calls and can also be configured to pulse when an SMS is received.

DTR this pin can be used to put the module in sleep mode. Pulling the pin high puts the module in sleep mode and disables the serial. Pulling it low will wake the module up.

MIC+- These two pins can be used to connect an external microphone to the module.

SPK+- these two pins can be used to connect an external speaker to the module.

SIM800L GSM/GPRS Module Parts:

The SIM800L module is a compact, versatile, and easy-to-use module for GSM and GPRS. The parts marking of the module is shown below

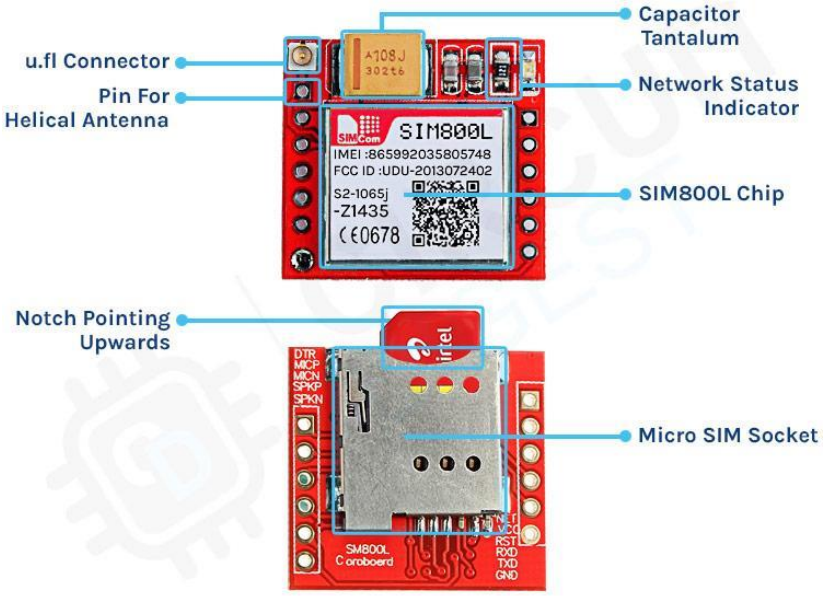


Figure 2.14

If you take a closer look at the SIM800L module there is not much on the PCB. On the front side of the PCB, we have the UFL connector and the SIM800L module itself. We also have some capacitors for decoupling, and we have a 1K current limiting resistor for the LED. Finally, we have a big 100uF,16V tantalum capacitor on board. On the backside of the board, we have the sim holder that is a push-to-lock type connector. This means you just need to insert a SIM card and push it for the card to work.

Overview of the SIM800L GSM/GPRS Module

At the heart of the breakout board, there is the SIM800L GSM/GPRS Module made by SIMCom. As mentioned in the above pinout section, the operating voltage of this device is 3.4V to 4.4V which means you can power this module directly from a lithium polymer battery. Other than that, all the usable pins are broken out to a 0.1" pin pitch that makes this module very breadboard friendly. It also has auto baud rate detection for ease of use. The module needs an external antenna to connect to the network, which is why there are two antenna options available for this board. In the package, you will get a helical antenna that you can directly solder to the NET pin of the module. But if you want to keep the antenna far away from the module board there is an option for connecting the external antenna with the onboard UFL connector. Any sim card with 2G connectivity will work with this module.

As we have mentioned earlier, there is an LED indicator on the top of the SIM800L module. It will blink at various rates to indicate network conditions.

Figure 2.15

- Blink every one second:

If the LED on the module is blinking every second, it indicates that the module is running but it's unable to connect to the cellular network right now.

- Blink every two seconds:

When the onboard LED on the monitor blinks every two seconds this means the GPRS data connection you requested is active and ready to accept requests on demand.

- Blink every three seconds:

When the LED on the module is blinking every three seconds, the module is connected to a network and can send/receive voice and SMS.

Circuit Diagram of the SIM800L GSM/GPRS Module

The circuit diagram of the SIM800L is very simple because the SIM800L IC takes care of all the complicated hardware, and you only need to connect a few wires to get up and running. This schematic will be very helpful to you if you are implementing this circuit in your PCB project.

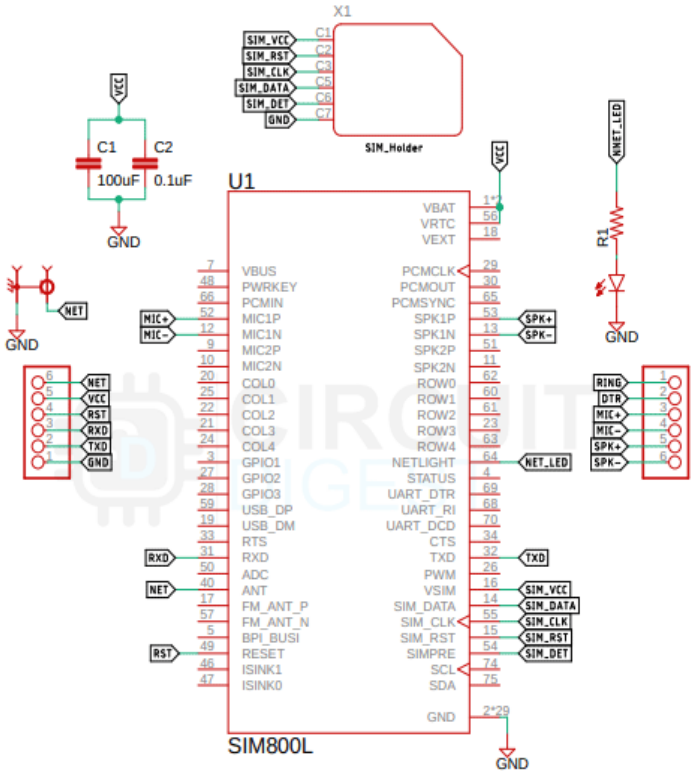


Figure 2.16

In the schematic, the sim card holder is directly connected to the SIM800L module; the power for the sim card is regulated by the SIM800 module only. And you can also see there is a big 100uF tantalum capacitor connected to the module, this is required because the module suddenly draws current spikes that sudden power requirement is fulfilled by the decoupling capacitor. Other than that, we have a tantalum capacitor and a decoupling capacitor along with the indicator LED.

2.6 Libraries used with Esp32.

SoftwareSerial

The #include <SoftwareSerial.h> statement is a preprocessor directive in C++ that instructs the compiler to include the SoftwareSerial library in your code.

The SoftwareSerial library is commonly used in Arduino projects. It provides the ability to create software-based serial communication ports on Arduino boards that do not have hardware support for additional serial ports. With this library, you can create multiple software serial ports to communicate with other devices or modules that use serial communication, such as Bluetooth modules, GPS modules, RFID readers, etc.

Once you include the SoftwareSerial library in your code, you can use its functions and classes to initialize and manage software serial ports, read and write data, and perform other operations related to serial communication.

It's important to note that the SoftwareSerial library is typically used in conjunction with the Serial object, which represents the default hardware serial port available on most Arduino boards (often connected to the USB port for serial communication with a computer.

TinyGPS++

The #include <TinyGPS++.h> statement is a preprocessor directive in C++ that instructs the compiler to include the TinyGPS++ library in your code.

The TinyGPS++ library is a popular library used in Arduino projects for parsing and handling data from GPS modules. It provides a set of functions and classes that make it easier to extract useful information from raw NMEA (National Marine Electronics Association) data received from GPS modules.

Once you include the TinyGPS++ library in your code, you can use its functions and classes to communicate with GPS modules, extract data such as latitude, longitude, altitude, speed, date, time, and more.

TinyGsmClient

The #include <TinyGsmClient.h> statement is a preprocessor directive in C++ that instructs the compiler to include the TinyGsmClient library in your code.

The TinyGsmClient library is a library commonly used in Arduino projects for connecting to the internet using a GSM module or a SIM card. It provides a client implementation that can be used with the Arduino Ethernet library or other libraries that use the Client interface, allowing you to establish a TCP/IP connection to remote servers. Once you include the TinyGsmClient library in your code, you can use its functions and classes to connect to a GSM network, establish a GPRS connection, and communicate with remote servers over TCP/IP.

ArduinoHttpClient

The #include <ArduinoHttpClient.h> statement is a preprocessor directive in C++ that instructs the compiler to include the ArduinoHttpClient library in your code.

The ArduinoHttpClient library is a library commonly used in Arduino projects for making HTTP requests and interacting with web services. It provides a set of functions and classes that allow you to create and send HTTP requests (GET, POST, etc.), handle responses, and perform various HTTP-related operations.

Once you include the ArduinoHttpClient library in your code, you can use its functions and classes to establish connections to remote servers, send HTTP requests, receive and process responses, and handle errors.

ArduinoJson

The #include <ArduinoJson.h> statement is a preprocessor directive in C++ that instructs the compiler to include the ArduinoJson library in your code.

The ArduinoJson library is a powerful library used in Arduino projects for parsing and generating JSON (JavaScript Object Notation) data. It provides a set of functions and classes that make it easier to work with JSON data structures.

Once you include the ArduinoJson library in your code, you can use its functions and classes to parse JSON data received from an API, create JSON objects or arrays, manipulate JSON data, and serialize JSON objects back into a string.

WiFi

The #include <WiFi.h> statement is a preprocessor directive in C++ that instructs the compiler to include the WiFi library in your code.

The WiFi library is a built-in library in the Arduino framework that provides functions and classes to connect and communicate with Wi-Fi networks. It allows your Arduino board to establish a wireless network connection, connect to access points (routers), and perform various network-related operations.

Once you include the WiFi library in your code, you can use its functions and classes to scan for available Wi-Fi networks, connect to a specific network using the network credentials (SSID and password), check the connection status, and perform actions such as sending HTTP requests, retrieving data from web servers, and more.

Firebase_ESP_Client

The #include <Firebase_ESP_Client.h> statement is a preprocessor directive in C++ that instructs the compiler to include the Firebase_ESP_Client library in your code.

The Firebase_ESP_Client library is a library commonly used in Arduino projects for connecting to the Firebase Realtime Database and performing various operations such as reading and writing data, listening for changes, and managing authentication.

Once you include the Firebase_ESP_Client library in your code, you can use its functions and classes to establish a connection with the Firebase Realtime Database, interact with database nodes, handle events and callbacks, and perform authentication if required.

Please note that the Firebase_ESP_Client library is a third-party library that provides a Firebase client implementation for ESP8266 and ESP32 boards.

stdio

The #include <stdio.h> statement is a preprocessor directive in C that instructs the compiler to include the standard input/output library (stdio.h) in your code.

The stdio.h library provides functions and macros for performing input and output operations in the C programming language. It includes functions like printf() for formatted output, scanf() for formatted input, and various file handling functions such as fopen(), fclose(), fread(), and fwrite()

Wire

The #include <Wire.h> statement is a preprocessor directive in Arduino programming that instructs the compiler to include the Wire library in your code.

The Wire library provides functions and classes for I2C (Inter-Integrated Circuit) communication, which is a commonly used serial communication protocol for connecting multiple devices in a master-slave configuration. The I2C protocol allows for easy communication between microcontrollers and various peripheral devices, such as sensors, displays, and other integrated circuits.

Once you include the Wire library in your code, you can use its functions and classes to communicate with I2C devices by sending and receiving data. The library provides methods for configuring the I2C bus, initiating data transmission, reading data from devices, and handling various aspects of the communication protocol.

SPI

The #include <SPI.h> statement is a preprocessor directive in Arduino programming that instructs the compiler to include the SPI (Serial Peripheral Interface) library in your code.

The SPI library provides functions and classes for utilizing the SPI communication protocol on Arduino boards. SPI is a synchronous serial communication protocol commonly used for interconnecting microcontrollers with other devices such as sensors, displays, memory modules, and other integrated circuits.

Once you include the SPI library in your code, you can use its functions and classes to configure the SPI interface, transfer data between the Arduino and SPI devices, and control various aspects of the SPI communication protocol.

Adafruit_PN532

The #include <Adafruit_PN532.h> statement is a preprocessor directive in Arduino programming that instructs the compiler to include the Adafruit PN532 library in your code.The Adafruit PN532 library provides functions and classes for interfacing with the PN532 NFC (Near Field Communication) module. NFC is a short-range wireless communication technology that allows for contactless data exchange between devices, such as smartphones, cards, and tags.

Once you include the Adafruit PN532 library in your code, you can use its functions and classes to communicate with the PN532 module and perform various NFC-related operations, such as reading NFC tags, writing data to NFC tags, and emulating NFC cards.

3.0 SOFTWARE COMPONENTS

3.1 MOBILE DEVELOPMENT

Now, let's talk about the mobile application. We used the Flutter framework and the Dart programming language to create it. We will provide more details about Flutter and Dart in the upcoming **sections. The** mobile application is designed for three types of users: parents, who are the primary users, teachers, who use it to take student attendance, and canteen workers, who use it to serve food to **students. The** application helps parents keep track of their children's attendance by sending notifications when they arrive at and leave school. Additionally, notifications are sent when their child makes a purchase, providing details of the purchase. Parents can recharge their balance using the application, and it also helps them to know the location of their students, set a daily spending limit for their children, and specify their children's allergies to avoid any adverse reactions when purchasing products. We will discuss all these features in more detail in the upcoming sections.

What Is the Flutter Framework?

Flutter is a free and open-source UI framework for creating native mobile applications from Google. Released in 2017, Flutter allows developers to build mobile applications with a single codebase and programming language. This capability makes building both iOS and Android apps simpler and faster. The Flutter framework consists of both a software development kit (SDK) and their widget-based UI library. This library consists of various

Figure 3.1

reusable UI elements, such as sliders, buttons, and text inputs. Developers building mobile applications with the Flutter framework will do so using a programming language called Dart. With a syntax like JavaScript, Dart is a typed object programming language that focuses on front-end development.

3.2 FLUTTER ARCHITECTURE

Flutter is comprised of three architectural layers. Here is a quick breakdown of each one:

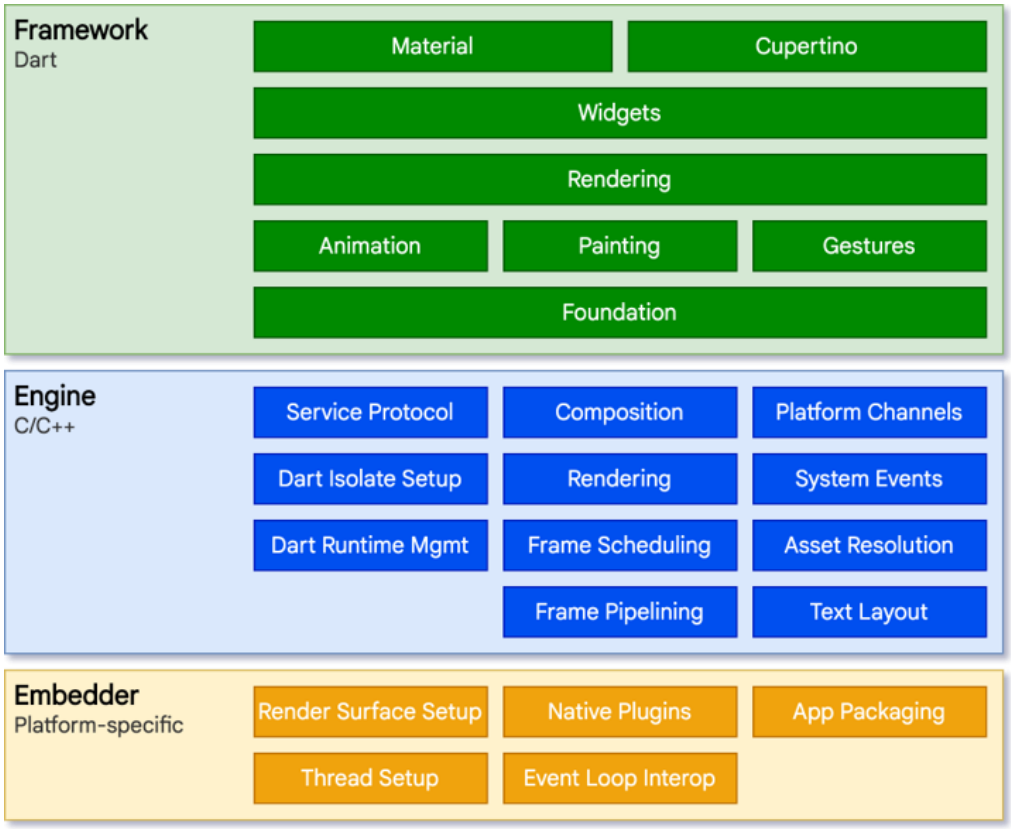


Figure 3.2

1. Framework. This is the more "visible" part of Flutter's technology. As already mentioned, this layer of Flutter's architecture in based on Dart. The rich set of libraries in this layer can be divided into multiple layers, such as the basic foundational classes, the widget layer, and the rendering layer.
2. Engine. This layer is written in C/C++. The engine provides a low-level implementation of the Flutter framework, as well as other core APIs, e.g., graphics, accessibility support, text layout, and plugin architecture.

3. Embedder. The embedded is a platform-specific layer that allows the Flutter app to run on any OS.

Although Flutter is a newer cross-platform framework, more and more companies have chosen Flutter over frameworks such as Xamarin, Cordova, and React Native. Some of the top reasons why development teams choose Flutter include:

- Increased productivity. Using the same codebase for iOS and Android saves both time and resources. Flutter's native widgets also minimize time spent on testing by ensuring there is little to no compatibility issues with different OS versions.
- Easy to learn. Flutter allows developers to build native mobile applications without needing to access OEM widgets or use a lot of code. This, in addition to Flutter's particularly appealing user interface, makes the mobile app creation process much simpler.
- Great performance. Users report that it is difficult to notice the difference between a Flutter app and a native mobile app.
- Cost-effective. Building iOS and Android apps with the same codebase is essentially building two apps for the price of one.
- Available on different IDEs. Developers are free to choose between Android Studio and VS Code to edit their code on Flutter.
- Application rendering. This unique feature allows Flutter to draw and display its own UI components. By not relying on UI components from the target system, Flutter offers an unparalleled level of detail, runtime performance, and control when compared to other frameworks.
- Great documentation & community. Flutter has many great resources to answer your questions, thanks to its ample documentation with easy-to-follow use cases. Flutter users also benefit from community hubs like Flutter Community and Flutter Awesome for exchanging ideas.

3.3 Pros & Cons of flutter framework

Although Flutter is not yet widely adopted, an increasing number of tech companies have started realizing its potential. Some notable Flutter applications include Google Ads, eBay Motors, Philips Hue, and Xianyu from Alibaba.

Let us look at some of the benefits that Flutter offers, as well as some of their drawbacks.

Flutter App Development Pros:

Flutter offers a wide range of benefits for business folks and developers alike. Flutter offers good quality for reasonable costs, as well as great usability and speed.

- Development goes faster when the same code is used for both iOS and Android applications. Flutter's singular codebase speeds up time-to-market while cutting mobile app development costs significantly.
- Flutter app performance is equivalent to that of native real-time applications. Unlike other frameworks, Flutter apps do not need a bridge to interact with native components. Since these bridges typically cause performance issues, this gives Flutter a decided advantage.
- Flutter's "hot reload" feature allows developers to change the code on emulators, simulators, or real devices, and see results in real-time. The changed code is then reloaded immediately while the app is running. No restart required. Hot reloads make building UIs, adding features, and bug fixing simpler than ever.

Flutter App Development Cons:

While Flutter app development presents a lot of positives, it is also important to consider the lesser aspects of the framework:

- Flutter is relatively new. While Flutter offers many plugins and UI components, frameworks like Xamarin and React Native offer a much larger selection. While Flutter will not be the new framework on the block forever, this is the situation now.
- Dart is not very popular. While it is a great programming language, developers are often much more likely to consider languages such as Java or Kotlin.

- Some components are only available for iOS or Android, but not both. These types of components more often support Android since Flutter comes from Google, and Android developers are typically more interested in Flutter than iOS developers.

3.4 FLUTTER DART

Figure 3.3

What is Dart?

Google had its first ever release of Flutter 1.0 last December, after having it in beta mode for over 18 months. Dart is the programming language used to code Flutter apps. Dart is another product by Google and released version 2.1, before Flutter, in November. As it is starting out, the Flutter community is not as extensive as React Native, Ionic, or Xamarin.

A while back, I discovered a liking for JavaScript. I was ecstatic to be working on a React Native mobile app for my internship. I enjoy coding hybrid mobile apps too, so wanted to give Flutter a try, as I had done Xamarin sometime last year.

At my first glance of Flutter (and Dart), I felt befuddled and couldn't seem to understand anything. They even had a section on their docs for developers moving from React Native. So, I took to digging deeper into all things Dart.

Dart looks a bit like C and is an object-oriented programming language. So, if you prefer the C languages or Java, Dart is the one for you, and you'll likely be proficient in it.

Dart is not only used for mobile app development but is a programming language. Approved as a standard by Ecma (ECMA-408), it's used to build just about anything on the web, servers, desktop and of course, mobile applications (Yes, the same people who standardized our favorites ES5 and ES6.)
Dart, when used in web applications, is trans piled to JavaScript, so it runs on all web browsers. The Dart installation comes with a VM as well to run the .dart files from a command-line interface. The Dart files used in Flutter apps are compiled and packaged into a binary file (.apk or. ipa) and uploaded it to app stores.

What does code in Dart look like?
Like most ALGOL languages (like C# or Java):

1. The entry point of a Dart class is the main () method. This method acts as a starting point for Flutter apps as well.
2. The default value of most data types is null.
3. Dart classes only support single inheritance. There can be only one superclass for a particular class, but it can have many implementations of Interfaces.
4. The flow control of certain statements, like if conditions, loops (for, while and do-while), switch-case, break and continue statements are the same.
5. Abstraction works in a similar manner, allowing abstract classes and interfaces.

Unlike them (and sometimes a bit like JavaScript):

1. Dart has type inference. The data type of a variable need not be explicitly declared, as Dart will "infer" what it is. In Java, a variable need to have its type explicitly given during declaration. For example, String something; But in Dart, the keyword is used instead like so, var something; The code treats the variable according to whatever it contains, be it a number, string, bool or object.
2. All data types are objects, including numbers. So, if left uninitialized, their default value is not a 0 but is instead null.

3. A return type of a method is not required in the method signature.
4. The type num declares any numeric element, both real and integer.
5. The super () method call is only at the end of a subclass's constructor.
6. The keyword new used before the constructor for object creation is optional.
7. Method signatures can include a default value to the parameters passed. So, if one is not included in the method call, the method uses the default values instead.
8. It has a new inbuilt data type called Runes, that deals with UTF-32 code points in a string. For a simple example, see emojis and similar icons.

And all these differences are just a few in the many that you can find in the Dart Language tour, which you can check out here.

Dart also has inbuilt libraries installed in the Dart SDK, the most used being:

1. dart: core for core functionality; it is imported in all dart files.
2. Dart: sync for asynchronous programming.
3. dart: math for mathematical functions and constants.
4. dart: convert for converting between different data representations, like JSON to UTF-8.

You can find more information on Dart libraries here.

Using Dart in Flutter:

Flutter has more app-specific libraries, more often on user interface elements like:

1. Widget: common app elements, like Text or List View.
2. Material: containing elements following Material design, like FloatingActionButton.
3. Cupertino: containing elements following current iOS designs, like CupertinoButton.

3.5 FLUTTER STATE MANGEMENT

When working on a Flutter app, you might encounter the need to split a large UI component into several smaller ones to improve the readability of the code. With multiple components, it's crucial to implement effective communication between them. All UI components should be always aware of the state of the app. This is called state management.

In Flutter, you can manage the state of your app just by using setState. But while setState can be your best friend, it's not a good idea to depend on it solely. There are many other factors you should also consider while developing a Flutter app, such as architecture, scalability, readability, complexity, etc. Staying on top of everything requires an effective state management technique.

There are myriad state management solutions available for Flutter, including Provider, InheritedWidget and InheritedModel, Redux, BLoC, GetIt, MobX, Riverpod, etc.

We used the BloC design pattern for state management in our application, specifically, we used Cubit. You will see how we are able to discuss the logic separately from the UI, and this is due to the use of Cubit.

What is BLoC?

Business logic components (BLoC) allow you to separate the business logic from the UI. Writing code in BLoC makes it easier to write and reuse tests.

What is Cubit?

Cubit is a simpler version of the BLoC pattern. It eliminates the need to write events. Cubit exposes direct functions, which can result in appropriate states. Writing a Cubit instead of BLoC also reduces boilerplate code, making the code easier to read.

Now, let's discuss our database. As mentioned in the hardware section, we are using Cloud Firestore as our database. In this section, we will focus on how to connect the mobile application to our Firebase project to access the data sent by devices and store its own data. Additionally, we will discuss how data is saved in Firestore. The mobile application interacts with the Cloud Firestore database, where devices store information and the app read and responds to it.

3.6 FLUTTER PACKAGES

Here are some of the packages involved in FLUTTER:

1. **Sqflite:**

SQLite plugin for Flutter. Supports iOS, Android and MacOS.

- Support transactions and batches
- Automatic version management during open
- Helpers for insert/query/update/delete **queries.**
- DB operation executed in a background thread on iOS and **Android.**

2. **Intl:**

- Provides internationalization and localization facilities, including message translation, plurals and genders, date/number formatting and parsing, and bidirectional text.
- The most important library is intl. It defines the Intl class, with the default locale and methods for accessing most of the internationalization mechanisms. This library also defines the DateFormat, NumberFormat, and BidiFormatter classes.

3. **bloc:**

- The goal of this package is to make it easy to implement the BLoC Design Pattern (Business Logic Component).
- This design pattern helps to separate *presentation* from *business logic*.
- Following the BLoC pattern facilitates testability and reusability. This package abstracts reactive aspects of the pattern allowing developers to focus on writing the business logic.
- A Cubit is class which extends BlocBase and can be extended to manage any type of state. Cubit requires an initial state which will be the state before emitting has been called.
- The current state of a cubit can be accessed via the state getter and the state of the cubit can be updated by calling emit with a new state.

- State changes in cubit begin with predefined function calls which can use the emit method to output new states. onChange is called right before a state change occurs and contains the current and next state.

4. **flutter_bloc:**
   - Widgets that make it easy to integrate blocs and cubits into Flutter. Built to work with package:bloc.
   - *Note: All widgets exported by the flutter_bloc package integrates with both Cubit and Bloc instances.*
5. **hexcolor:**
   - Brings HexColor for your dart and flutter projects, convert material colors to hex colors.
6. **conditional_builder_null_safety:**
   - This package's purpose is to show UI elements according for conditional type and supports null safety.
   - Show UI elements according for conditional type and supports null safety.
7. **shared_preferences:**
   - Flutter plugin for reading and writing simple key-value pairs. Wraps NSUserDefaults on iOS and SharedPreferences on Android.
   - Wraps platform-specific persistent storage for simple data (NSUserDefaults on iOS and macOS, SharedPreferences on Android, etc.). Data may be persisted to disk asynchronously, and there is no guarantee that writes will be persisted to disk after returning, so this plugin must not be used for storing critical data.
8. **fluttertoast:**
   - Toast Library for Flutter, easily create toast messages in single line of code.
9. **Image_picker:**
   - Flutter plugin for selecting images from the Android and iOS image library and taking new pictures with the camera.
10. **file_picker:**

- A package that allows you to use a native file explorer to pick single or multiple absolute file paths, with extension filtering support.

11. **map_launcher:**
    - A simple package that uses url_launcher to launch the maps app with the proper scheme on all platforms.
    - On iOS, map links as specified by Apple are launched. On Android, the geo intent is used as documented here. For web and other platforms, the plugin will simply launch Google Maps.
12. **permission_handler:**
    - Permission plugin for Flutter. This plugin provides a cross-platform (iOS, Android) API to request and check permissions.
    - On most operating systems, permissions aren't just granted to apps at installation time. Rather, developers must ask the user for permissions while the app is running.
    - This plugin provides a cross-platform (iOS, Android) API to request permissions and check their status. You can also open the device's app settings so users can grant a permission. On Android, you can show a rationale for requesting a permission.
13. **path_provider:**
    - Flutter plugin for getting commonly used locations on host platform file systems, such as the temp and app data directories.
    - A Flutter plugin for finding commonly used locations on the filesystem. Supports Android, iOS, Linux, macOS and Windows. Not all methods are supported on all platforms.
14. **excel:**
    - A flutter and dart library for reading, creating, editing, and updating excel sheets with compatible both on client and server side.
    - a flutter and dart library for reading, creating, and updating excel-sheets for XLSX files.
    - This library is MIT licensed So, it's free to use anytime, anywhere without any consent, because we believe in Open Source work.

15. **http:**
    - A composable, Future-based library for making HTTP requests.
    - This package contains a set of high-level functions and classes that make it easy to consume HTTP resources. It's multi-platform, and supports mobile, desktop, and browsers.

16. **flutter_typeahead:**
    - A highly customizable typehead (autocomplete) text input field for Flutter.
    - A TypeAhead (autocomplete) widget for Flutter, where you can show suggestions to users as they type.
17. **flutter_native_splash:**
    - Customize Flutter's default white native splash screen with background color and splash image. Supports dark mode, full screen, and more.
18. **open_file:**
    - A plug-in that can call native APP to open files with string result in flutter, support iOS (DocumentInteraction) / android(intent) / PC (ffi) / web(dart:html).
19. **connectivity_plus:**
    - This plugin allows Flutter apps to discover network connectivity and configure themselves accordingly. It can distinguish between cellular vs Wi-Fi connection.
20. **cupertino_icons:**
    - This is an asset repo containing the default set of icon assets used by Flutter's Cupertino widgets.
21. **multi_select:**
    - A simple multiselect dropdown. It provides a concise way to create a Multi Select ComboBox/SelectBox.
22. **dropdown_search:**
    - Simple and robust Searchable Dropdown with item search feature, making it possible to use an offline item list or filtering URL for easy customization.

23. **google_maps_flutter:**
    - A Flutter plugin for integrating Google Maps in iOS and Android applications.
24. **flutter_credit_card:**
    - A Credit Card widget package, support entering card details, card flip animation.
    - A Flutter package allows you to easily implement the Credit card's UI easily with the Card detection.
25. **percent_indicator:**
    - A library that allows you to display progress widgets based on percentage, can be Circular or Linear, you can also customize it to your needs.

3.7 CONNECTING FIREBASE TO A MOBILE APPLICATION

Here are the basic steps to connect Firebase to a mobile application:

Note: these steps are applied to the same Firebase project we created in the hardware part

1. Select "Android" as the app platform.
2. We will be asked for the app's package name, which can be found in different places, including the app level build.gradle file.

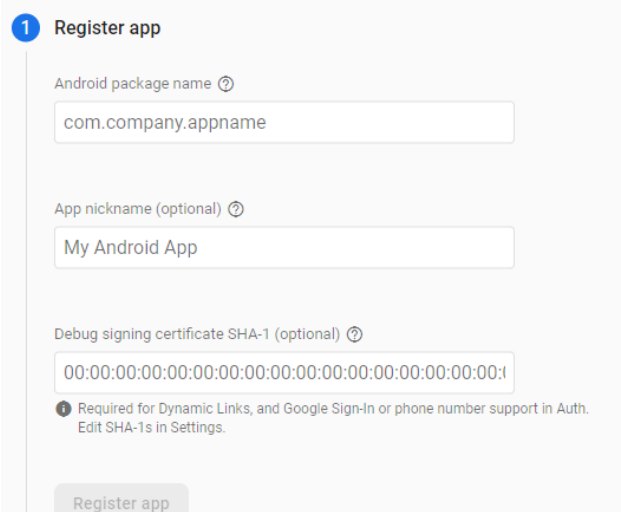


Figure 3.4

It is the ‘applicationId’ field. Copy and paste it to Android package name field.

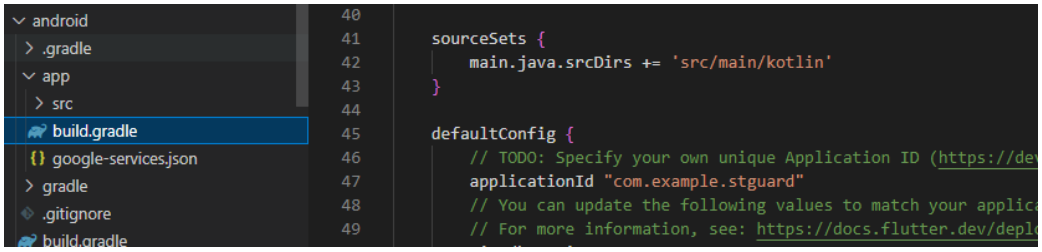


Figure 3.5

3. Optionally, you can add a nickname for the app and a debug signing certificate SHA-1 hash for authentication purposes. This is not required for development builds, but it is necessary if you plan to publish the app to the Google Play Store.
4. Click on "Register app" to continue and then download the google-services.json configuration file that we will need to add to the project.
5. Place the google-services.json file in the android/app directory of your project.

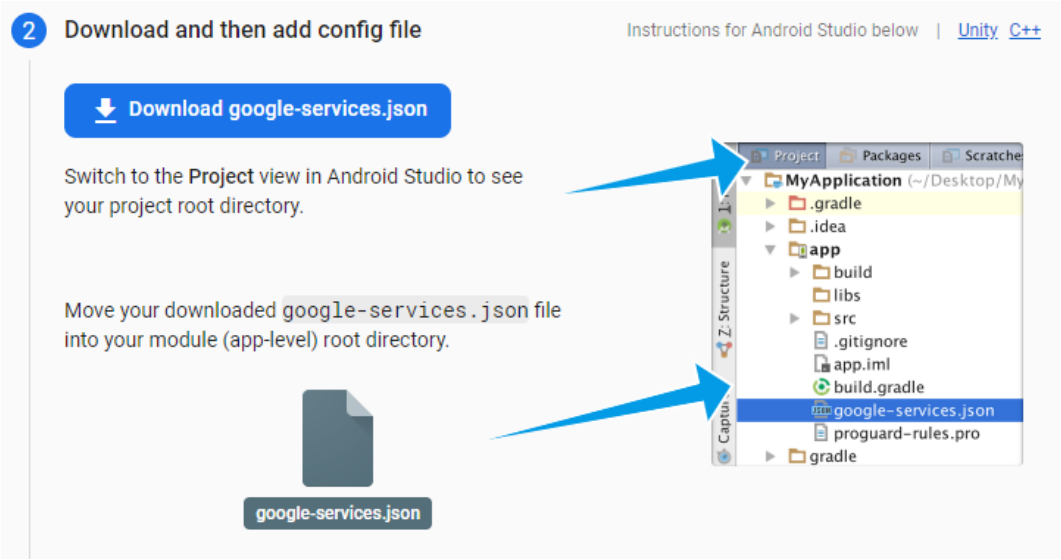


Figure 3.6

6. Then, we add Firebase SDK
   1. To make the google-services.json config values accessible to Firebase SDKs, you need the Google services Gradle plugin.

Add the plugin as a buildscript dependency to your project-level build.gradle file:

Root-level (project-level) Gradle file (<project>/build.gradle):

```
buildscript {
  repositories {
    // Make sure that you have the following two repositories
    google()  // Google's Maven repository
    mavenCentral()  // Maven Central repository
  }
  dependencies {
    ...
    // Add the dependency for the Google services Gradle plugin
    classpath 'com.google.gms:google-services:4.3.15'
  }
}

allprojects {
  ...
  repositories {
    // Make sure that you have the following two repositories
    google()  // Google's Maven repository
    mavenCentral()  // Maven Central repository
  }
}
```

Figure 3.7

2. Then, in your module (app-level) build.gradle file, add both the google-services plugin and any Firebase SDKs that you want to use in your app:

- (•) Kotlin
- ( ) Java

Module (app-level) Gradle file (<project>/<app-module>/build.gradle):

```
plugins {
  id 'com.android.application'
  // Add the Google services Gradle plugin
  id 'com.google.gms.google-services'
  ...
}

dependencies {
  // Import the Firebase BoM
  implementation platform('com.google.firebase:firebase-bom:31.2.1')

  // TODO: Add the dependencies for Firebase products you want to use
  // When using the BoM, don't specify versions in Firebase dependencies
  implementation 'com.google.firebase:firebase-analytics-ktx'

  // Add the dependencies for any other desired Firebase products
  // https://firebase.google.com/docs/android/setup#available-libraries
}
```

Figure 3.8

3. After adding the plugin and the desired SDKs, sync your Android project with Gradle files.
4. This is the end. Press the 'next' button and you all set.

Now, we need to add Firebase dependencies to the 'pubspec.yaml' file in your Flutter project:

1. Open the pubspec.yaml file and scroll down to the dependencies section.
2. Add the dependency you want to use under the dependencies section. The format should be dependency_name: version_number. For example, to add the Cloud Firestore package, you would add the following line: cloud_firestore: ^4.4.0
3. Save the pubspec.yaml file and run flutter pub get in your terminal to install the new dependency.

After we talked about how we can connect our App to Firebase, let's talk about how data is saved in Firestore in the next chapters.

4.0 CLOUD FIRESTORE

Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud. Like Firebase Realtime Database, it keeps your data in sync across client apps through realtime listeners and offers offline support for mobile and web so you can build responsive apps that work regardless of network latency or Internet connectivity.
Cloud Firestore also offers seamless integration with other Firebase and Google Cloud products, including Cloud Functions.

4.1 Key capabilities

1. Flexibility, The Cloud Firestore data model supports flexible, hierarchical data structures. Store your data in documents, organized into collections. Documents can contain complex nested objects in addition to subcollections.
2. Expressive querying, In Cloud Firestore, you can use queries to retrieve specific documents or to retrieve all the documents in a collection that match your query parameters. Your queries can include multiple, chained filters and combine filtering and sorting. They're also indexed by default, so query performance is proportional to the size of your result set, not your data set.
3. Realtime updates, Like Realtime Database, Cloud Firestore uses data synchronization to update data on any connected device. However, it's also designed to make simple, one-time fetch queries efficiently.
4. Offline support, Cloud Firestore caches data that your app is actively using, so the app can write, read, listen to, and query data even if the device is offline. When the device comes back online, Cloud Firestore synchronizes any local changes back to Cloud Firestore.
5. Designed to scale, Cloud Firestore brings you the best of Google Cloud's powerful infrastructure: automatic multi-region data replication, strong consistency guarantees, atomic batch operations, and real transaction support. We've designed Cloud Firestore to handle the toughest database workloads from the world's biggest apps.

4.2 How does it work?

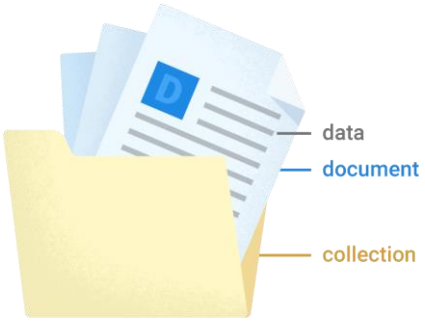


Figure 4.1

Cloud Firestore is a cloud-hosted, NoSQL database that your Apple, Android, and web apps can access directly via native SDKs. Cloud Firestore is also available in native Node.js, Java, Python, Unity, C++ and Go SDKs, in addition to REST and RPC APIs.

Following Cloud Fire store's NoSQL data model, you store data in documents that contain fields mapping to values. These documents are stored in collections, which are containers for your documents that you can use to organize your data and build queries. Documents support many different data types, from simple strings and numbers, to complex, nested objects. You can also create subcollections within documents and build hierarchical data structures that scale as your database grows. The Cloud Firestore data model supports whatever data structure works best for your app.

Additionally, querying in Cloud Firestore is expressive, efficient, and flexible. Create shallow queries to retrieve data at the document level without needing to retrieve the entire collection, or any nested subcollections. Add sorting, filtering, and limits to your queries or cursors to paginate your results. To keep data in your apps current, without retrieving your entire database each time an update happens, add real-time listeners. Adding real-time listeners to your app notifies you with a data snapshot whenever the data your client apps are listening to changes, retrieving only the new changes.

Protect access to your data in Cloud Firestore with Firebase Authentication and Cloud Firestore Security Rules for Android, Apple platforms, and JavaScript, or Identity and Access Management (IAM) for server-side languages.

Choose a database: Cloud Firestore or Realtime Database

Firebase offers two cloud-based, client-accessible database solutions that support real-time data syncing:

- Cloud Firestore is Firebase's newest database for mobile app development. It builds on the successes of the Realtime Database with a new, more intuitive data model.

Cloud Firestore also features richer, faster queries and scales further than the Realtime Database.

- Realtime Database is Firebase's original database. It's an efficient, low-latency solution for mobile apps that require synced states across clients in real-time.

Let's get started with Cloud Firestore.

Create a Cloud Firestore database:

To get started with Firebase, click on Go to Console in the upper right corner.

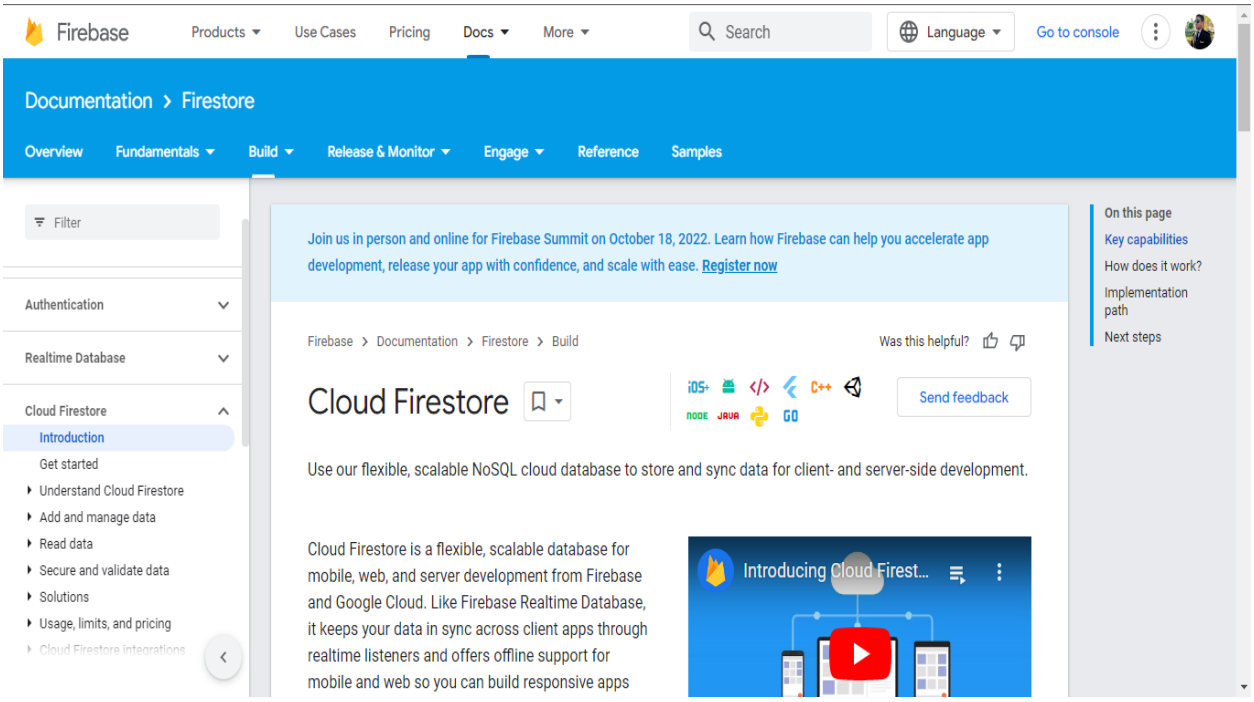


Figure 4.2

Then, click on Create a project.

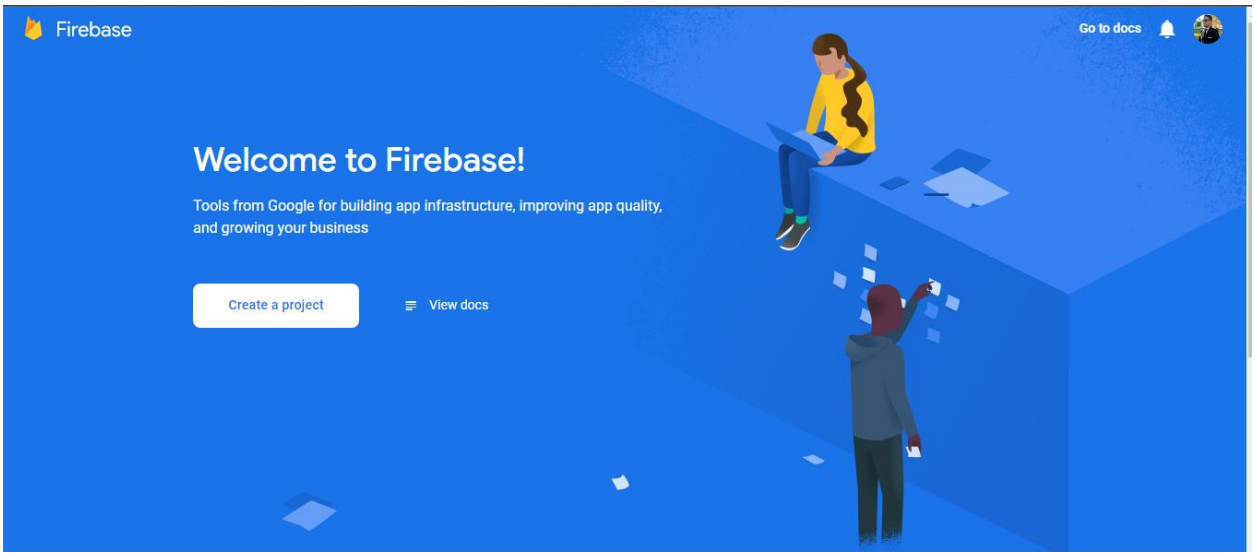


Figure 4.3

Continue filling in your information to get your Firebase project created.

After your project is created, click on **Build** on the left then go to **Authentication** to choose which authentication method to use.

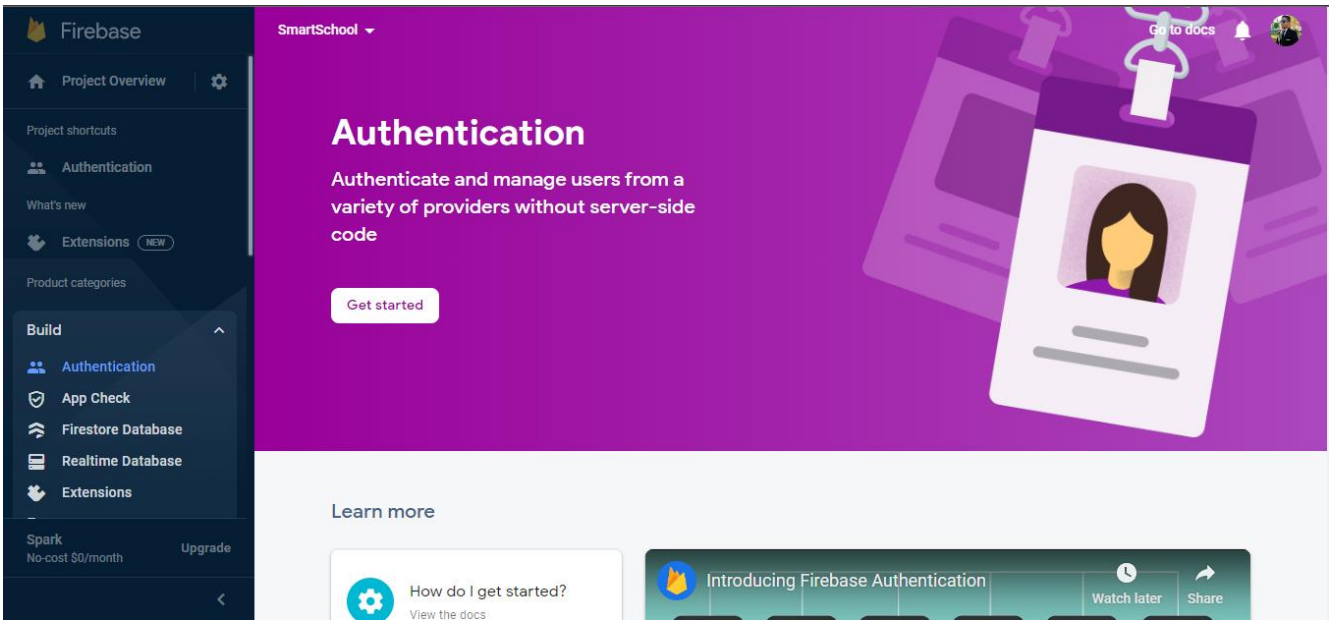
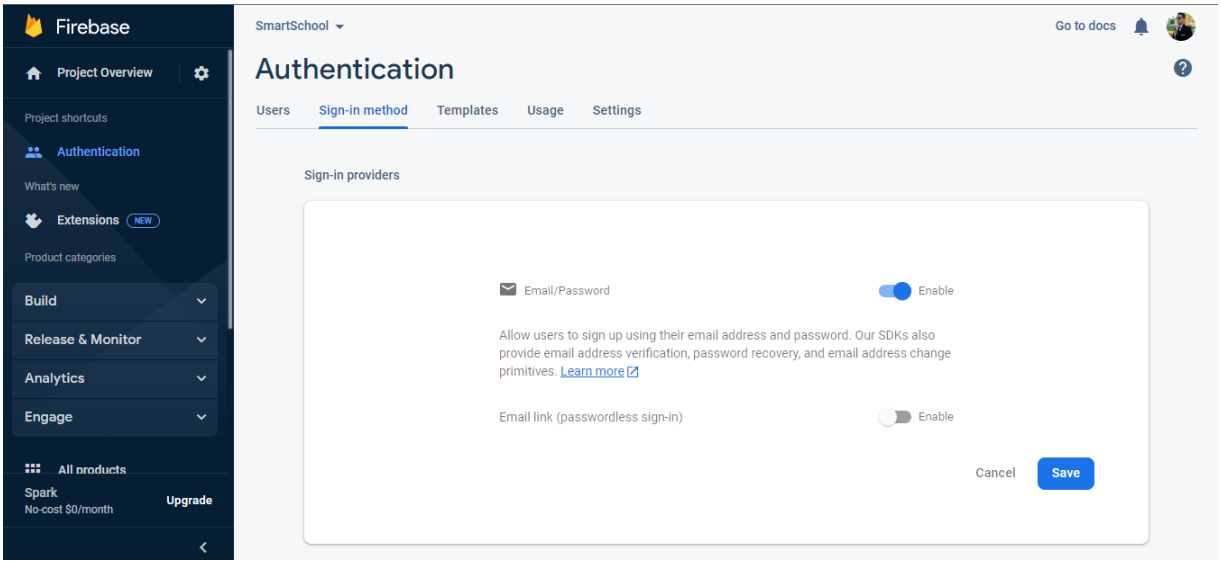


Figure 4.4

We will choose the **email/password** method. Click on it and enable it. Then, press save.



In

Figure 4.5

ESP32, it's not easy to get Email and password directly from users, so we are going to create a specific user in advance. Select Users from the menu above and register one new user through the Add user button. After we add a user, we are going to set up the Firestore DB. From Build on the left, select Firestore Database. Click on Create database.

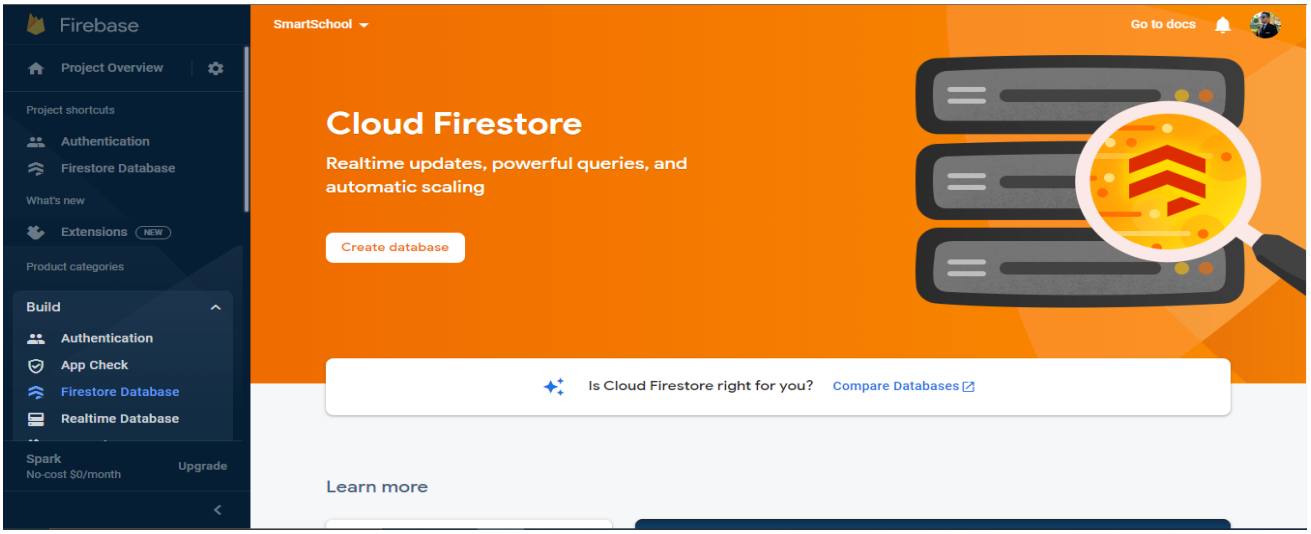


Figure 4.6

When you click on **Create database**, it will ask you to select secure rules for the database.

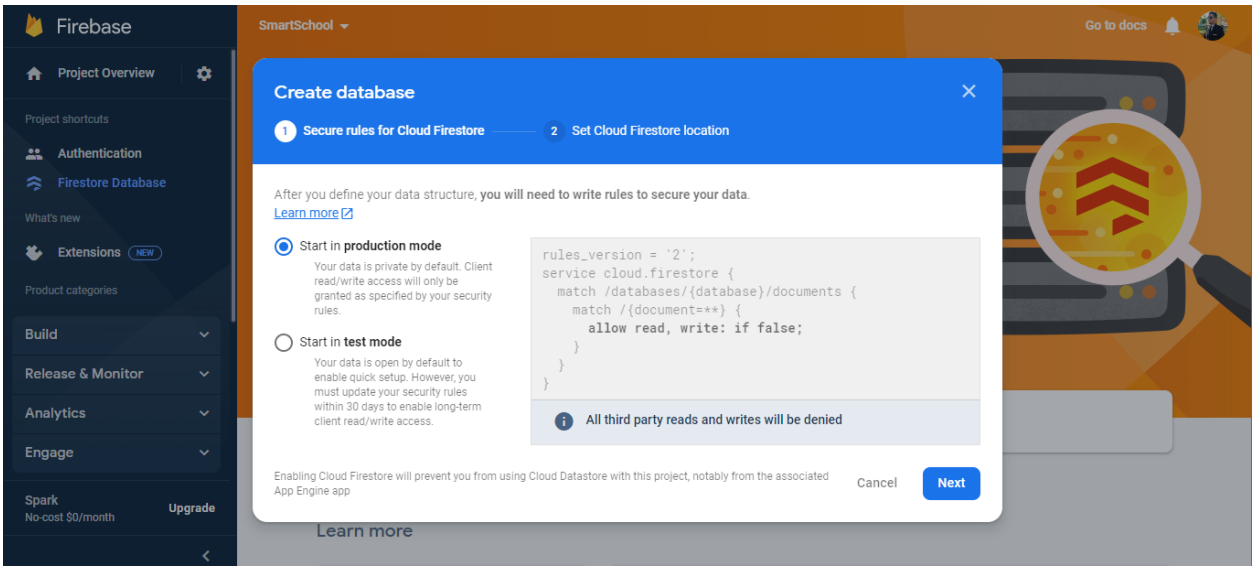


Figure 4.7

No matter what we choose now, we can change it later. So, we'll select the default choice and then press **Next**. Finally, select the Cloud Firestore location.

I used the link below to know the closest location to Egypt. We google it to find the closed location to Egypt.

After choosing the location, a Cloud database has been created.

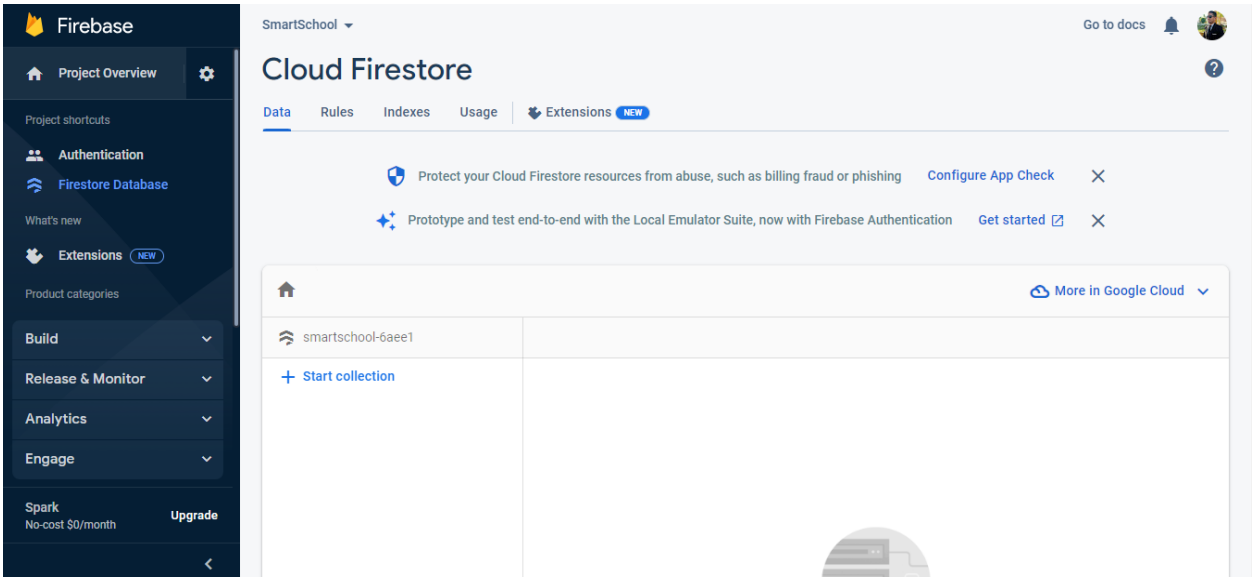


Figure 4.8

A final step is to set up the rules for database access. Select '**Rules**' from the menu above. Currently, no one can read or write because the if condition is false.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if false;
    }
  }
}
```

Figure 4.9

We will modify the rule so that it can be read or written if it's an authorized request. This allows the registered users in this project to read and write this database.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

Figure 4.10

This is the end of the basic Firebase setup.

4.3 FIREBASE USED SERVICES IN MOBILE APPLICATION

Authentication:

- Ensures the identity of the user using the mobile application by entering an e-mail address and password.
- The user first registers an e-mail address and password if he does not have one, then logs in with the e-mail address and password.
- After using the mobile application, the user can sign out from the app.

Firestore Database.

- provides several features that make it a powerful and flexible database for building applications:
    1. Real-time updates: Firestore supports real-time updates, which means that any changes made to the database are instantly synchronized across all connected clients.
    2. Scalability: Firestore is designed to handle large volumes of data and can automatically scale to meet the needs of your application.
    3. Security: Firestore provides a robust security model that allows you to control access to your data using authentication, authorization, and rules.
    4. Offline support: Firestore provides built-in support for offline data access, which means that your application can continue to function even when the user is offline.

Integration with other Firebase services:

Firestore integrates seamlessly with other Firebase services, such as authentication, cloud functions, and cloud messaging, making it easy to build powerful and scalable applications.

- Overall, Firestore is a powerful and flexible database that provides a simple and intuitive API for storing and synchronizing data in real time. It is an ideal choice for building web and mobile applications with complex data requirements.

Storage:

is a cloud-based storage service provided by Google as part of the Firebase platform. It allows developers to store and serve user-generated content such as images, videos, audio files, and documents. Firebase Storage is designed to be secure, reliable, and scalable, making it an ideal choice for storing and sharing files for web and mobile applications and provides several key features:

Secure and reliable storage:

Firebase Storage provides secure and reliable storage options for developers, with support for multiple file types and sizes.

Integration with other Firebase services:

Firebase Storage is integrated with other Firebase services such as Authentication, Cloud Functions, and Realtime Database, making it easy to build powerful and robust applications.

Simple and flexible API:

Firebase Storage provides a simple and flexible API for uploading, downloading, and managing files, with support for advanced features such as resumable uploads and downloads.

Global CDN:

Firebase Storage uses a global Content Delivery Network (CDN) to ensure fast and reliable delivery of files to users worldwide.

Cost-effective pricing:

Firebase Storage offers cost-effective pricing plans based on usage, with a free quota available for developers to get started.

Firebase Storage is easy to set up and use, with a simple web interface and client libraries for popular programming languages such as JavaScript, iOS, and Android. Developers can use Firebase Storage to store and serve files for a wide range of applications, including social media platforms, e-commerce sites, and content management systems.

Messaging:

Firebase Cloud Messaging (FCM) is a messaging solution provided by Google as part of the Firebase platform. It enables developers to send push

notifications and messages to their users on Android, iOS, and the web. Firebase Messaging is designed to be a reliable and scalable solution for sending messages to millions of devices with minimal effort.

Firebase Messaging provides several key features:

1. Cross-platform messaging: Firebase Messaging supports messaging on Android, iOS, and the web, allowing developers to reach their users across multiple platforms.
2. Rich notifications: Firebase Messaging supports rich, personalized notifications that can include images, videos, and action buttons.
3. Audience segmentation: Firebase Messaging allows developers to target specific groups of users with personalized messages based on user behavior, location, and other criteria.
4. Delivery reliability: Firebase Messaging uses a reliable and scalable infrastructure to ensure that messages are delivered quickly and reliably to all devices.
5. Analytics: Firebase Messaging provides analytics and insights into message delivery, engagement, and conversion rates, allowing developers to optimize their messaging campaigns.

Firebase Messaging is easy to integrate with other Firebase services such as Authentication, Realtime Database, and Cloud Functions, allowing developers to build powerful messaging applications with minimal effort. It also provides a simple and flexible API for sending messages programmatically, making it easy to automate messaging workflows and integrate with third-party services.

4.4 Fire base packages

1) **firebase_core:**

- A Flutter plugin to use the Firebase Core API, which enables connecting to multiple Firebase apps.

2) **firebase_auth:**

- Flutter plugin for Firebase Auth, enabling Android and iOS authentication using passwords, phone numbers and identity providers like Google, Facebook and Twitter.

3) **cloud_firestore:**

- Flutter plugin for Cloud Firestore, a cloud-hosted, noSQL database with live synchronization and offline support on Android and iOS.

4) **firebase_messaging:**

- Flutter plugin for Firebase Cloud Messaging, a cross-platform messaging solution that lets you reliably deliver messages on Android and iOS.

5) **firebase_storage:**

- Flutter plugin for Firebase Cloud Storage, a powerful, simple, and cost-effective object storage service for Android and iOS.

5.0 SYSTEM ARCHETICTURE

5.1 Main system components

The Smart School System follows a client-server architecture, where the hardware components and the mobile application act as clients, and the backend server manages the communication and data storage.

Hardware Components:

- ESP32: The ESP32 serves as the central control unit, responsible for processing and managing the interactions between the different hardware components.
- NFC Chip: The NFC chip embedded in the student's bracelet enables contactless communication for payment transactions and data exchange with the NFC reader.
- GPS Module: The GPS module tracks the location of the students and communicates the data to the ESP32.
- NFC Reader: The NFC reader reads the data stored on the NFC bracelet, such as student information and payment details.

Mobile Application:

The mobile application is the user interface for parents, canteen workers, and teachers. It allows parents to track their child's location, manage payment settings, and provide allergy information. Canteen workers can manage inventory, track sales, and process payments, while teachers can take attendance and add notes about students.

Backend Server:

The backend server plays a crucial role in the system architecture. It handles the communication between the mobile application, hardware components, and the payment gateway. It also manages the storage and retrieval of data in the Cloud Firestore database.

Cloud Firestore Database:

Cloud Firestore is a cloud-based NoSQL document database provided by Google. It stores all the data related to the system, such as student information, payment details, allergy information, canteen inventory, and teacher notes. The database enables real-time synchronization, ensuring that all clients have access to the latest data.

Payment Gateway:

The payment gateway is integrated with the backend server and allows secure and seamless payment transactions. It connects to the parents' credit card information, enabling students to make payments for snacks through the NFC chip.

5.2 Main Devices

Bracelet Device:

Connections of u-blox NEO-6M GPS Module and SIM800Lwith esp 32:

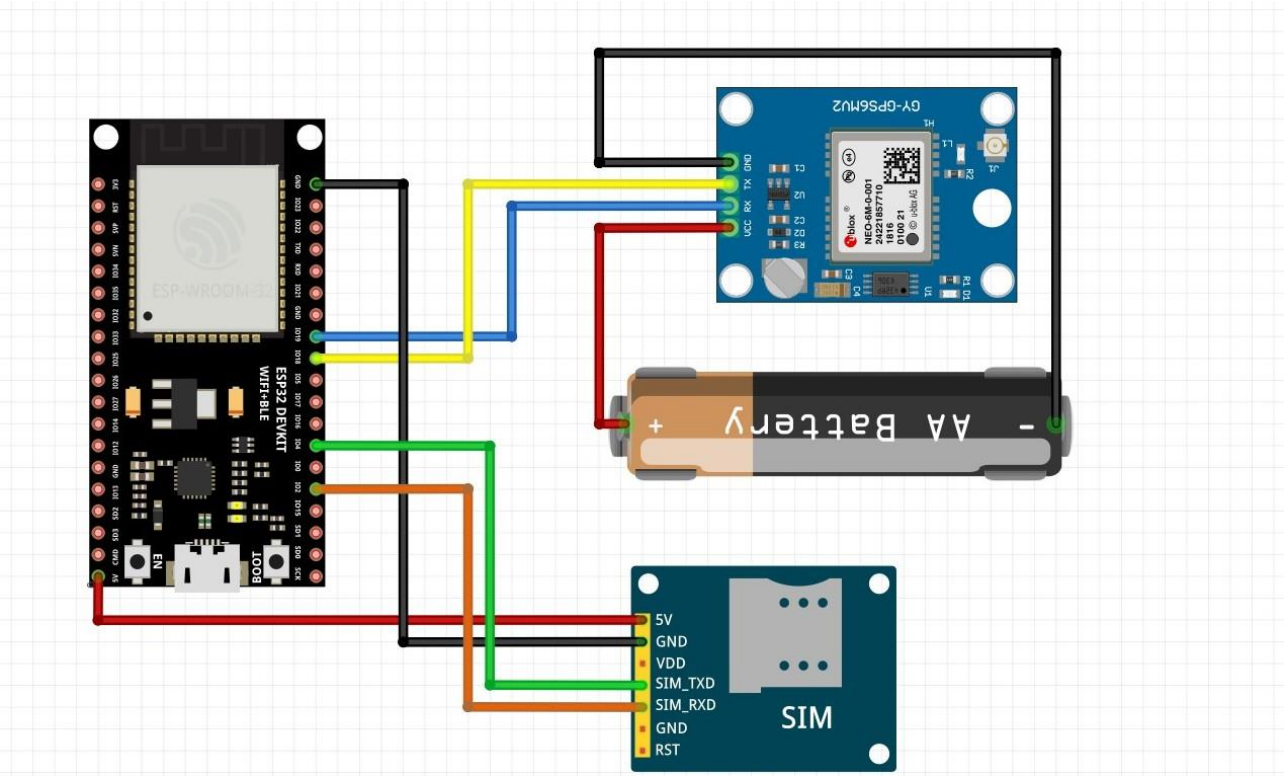


Figure 5.1

NEO-6M GPS Module.

- GND is connected to the ground of battery.
- TX is connected to pin no 18 on esp32.
- RX is connected to pin no 19 on esp32.
- Vcc is connected to the +ve of the battery.

SIM800Lwith esp32.-

- 5v is connected to the 5v pin on the esp32.
- GND is connected to the GND pin on the esp32.
- SIM-TXD is connected to pin no 4 on the esp32.
- SIM-TXD is connected to pin no 2 on the esp32.

In the smart school system project, the u-blox NEO-6M GPS module and SIM800L module are used in conjunction with the ESP32 microcontroller to enable GPS tracking and communication functionalities. Here's an explanation of how these components connect and perform in the project:

1. u-blox NEO-6M GPS Module:
    - The u-blox NEO-6M GPS module is responsible for acquiring GPS satellite signals and determining the precise location coordinates of the NFC bracelet worn by the student.
    - The GPS module communicates with the ESP32 microcontroller through a serial communication protocol, typically using UART (Universal Asynchronous Receiver-Transmitter) interface.
    - The GPS module is connected to specific pins on the ESP32, such as TX (transmit) and RX (receive) pins, to establish a serial communication link.
    - The ESP32 reads the GPS data, including latitude and longitude coordinates, from the NEO-6M module through the serial interface.
    - The obtained GPS location data is then used to update the student's location in real-time and send it to the parents' application.
2. SIM800L Module:
    - The SIM800L module is a GSM/GPRS module that facilitates communication with the parents' application via cellular networks.

- It allows the ESP32 to establish a cellular connection and send/receive data over the internet.
- The SIM800L module connects to the ESP32 through serial communication, typically using UART.
- The ESP32 communicates with the SIM800L module using AT commands, which are specific instructions sent over the serial interface to control the module's functionality.
- The SIM800L module requires a SIM card to establish a cellular connection and access the internet.

3. Connection with ESP32:
    - The UART TX pin of the GPS module is connected to a designated UART RX pin on the ESP32, and the UART RX pin of the GPS module is connected to the corresponding UART TX pin on the ESP32.
    - This connection allows the ESP32 to receive GPS data from the NEO-6M module.
    - The UART TX and RX pins of the SIM800L module are connected to specific UART pins on the ESP32 in a similar manner, enabling communication between the ESP32 and the SIM800L module.
4. Functionality:
    - The ESP32 reads GPS data from the NEO-6M module and combines it with other data, such as the student's entry/exit status.
    - It then sends the GPS location and relevant information to the parent application through the SIM800L module using cellular communication.
    - The SIM800L module establishes a cellular connection and communicates with the parent application's server, enabling the transmission of data.
    - The parent application receives the GPS location updates and processes them to provide real-time location information to the parents.

By connecting the u-blox NEO-6M GPS module and SIM800L module with the ESP32, the project achieves GPS tracking functionality and enables communication between the NFC bracelet and the parent application via cellular networks. This integration allows parents to receive accurate location updates of their child and ensures seamless communication within the smart school system.

Gate device:

Connections of NFC,LED and Buzzer(GATE)

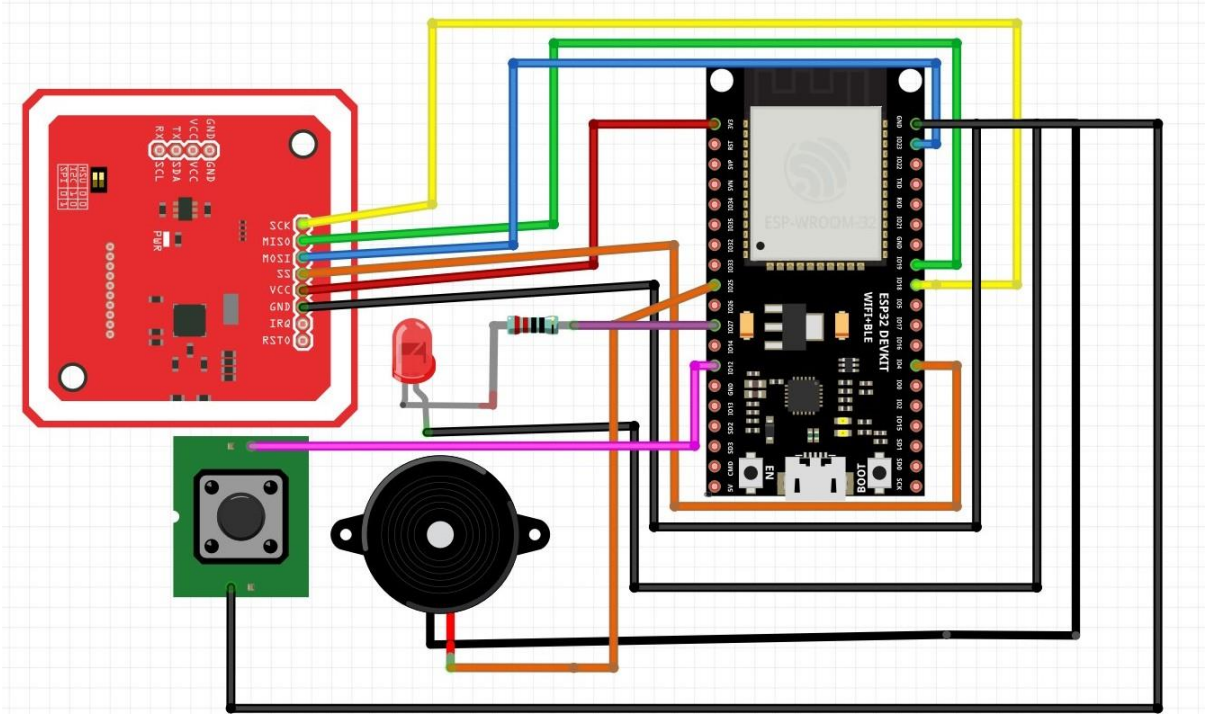


Figure 5.2

NFC:

- The SCK pin on NFC is connected to pin 18 on esp32.
- The MISO pin on NFC is connected to pin 19 on esp32.
- pin MOSI on NFC is connected to pin 18 on esp32.
- -The SS pin on NFC is connected to pin 4 on esp32.
- The VCC pin on NFC is connected to 3v3 pin on esp32.
- The GND pin on NFC is connected to GND pin on esp32.

buzzer:

- The positive terminal of the buzzer is connected to pin 25.
- The negative terminal of the buzzer is connected to GND pin.

LED:

- The positive terminal is in series with resistor and connected to pin 27 on esp32.
- The negative terminal is connected to GND pin on esp32.

push button:

- The positive terminal is connected to pin 12.
- The negative terminal is connected to GND.

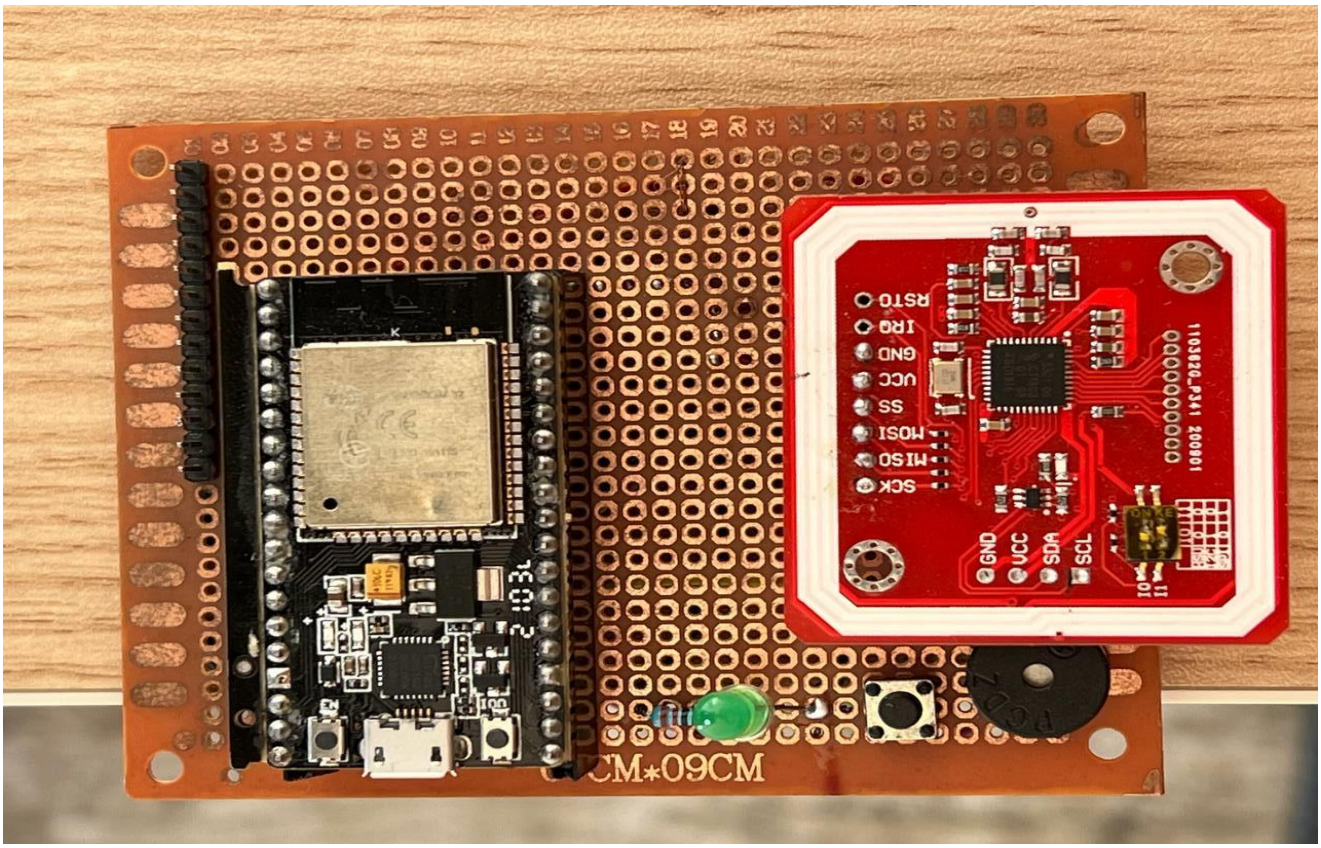


Figure 5.3

In the first gate device of the smart school system, the flow of data involves capturing the student's entry or exit status, generating an alive location update, and sending it to the parents. Here's an explanation of the data flow in the first gate:

1. Student Interaction:
    - When a student approaches the first gate, they pass the nfc bracelet over the nfc reader on the gate.
    - The school security presses the push button to indicate whether they are entering or leaving the school premises.
2. Input Capture:
    - The push button press is detected by the gate device, which serves as an input signal.
    - The device captures this input to determine the student's entry or exit status.
3. Parent Notification:
    - The gate device sends the alive location update to the parent application through a wireless communication channel (e.g., Bluetooth or Wi-Fi).
    - The parent application receives the alive location update, which includes the student's location and entry/exit status.
4. Buzzer Activation:
    - The gate device activates a buzzer to generate a sound or alert.
    - The buzzer notifies the student that their NFC bracelet has been successfully detected.

The flow of data in the first gate involves capturing the student's entry/exit status, retrieving the GPS location from the NFC bracelet, generating an alive location update, sending it to the parent application, and activating a buzzer for the student's awareness. This data flow ensures that parents receive real-time information about their child's location and provides a notification mechanism to confirm the successful detection of the NFC bracelet.

Canteen device:

Connections of NFC and Buzzer(Canteen):

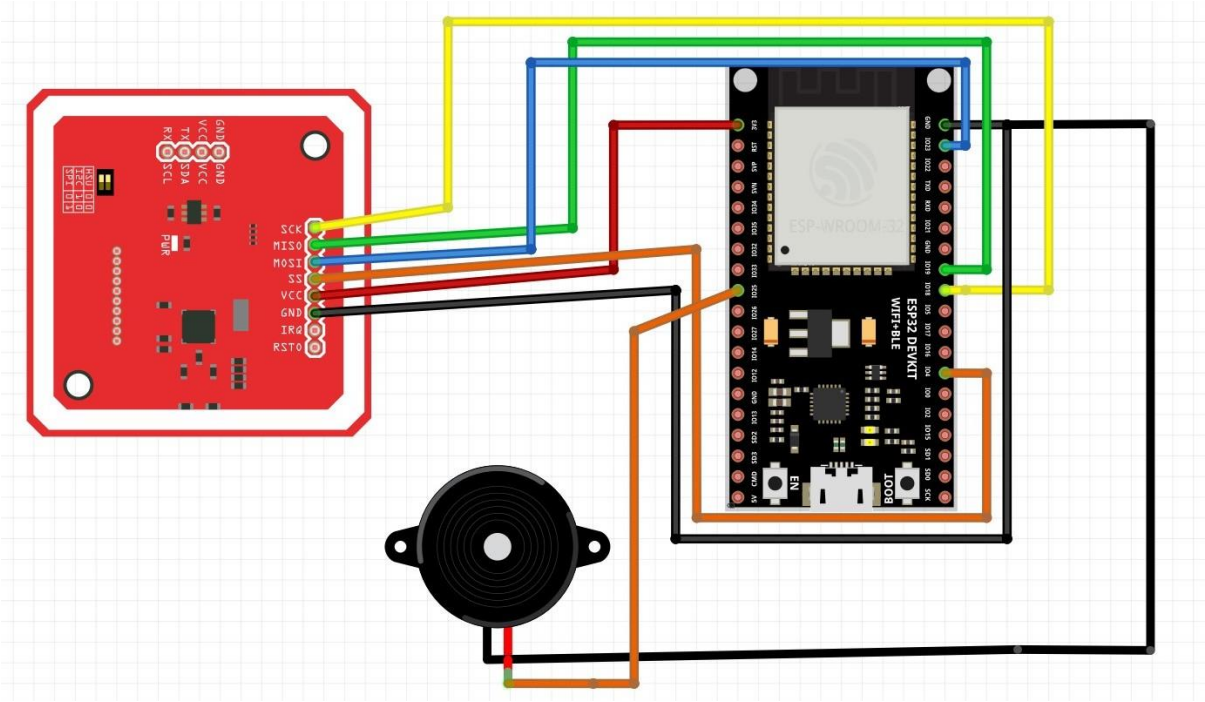


Figure 5.4

NFC:

- The SCK pin on NFC is connected to pin 18 on esp32.
- The MISO pin on NFC is connected to pin 19 on esp32.
- pin MOSI on NFC is connected to pin 18 on esp32.
- The SS pin on NFC is connected to pin 4 on esp32.
- The VCC pin on NFC is connected to 3v3 pin on esp32.
- The GND pin on NFC is connected to GND pin on esp32.

Buzzer:

- The positive terminal of the buzzer is connected to pin 25.
- The negative terminal of the buzzer is connected to GND pin.

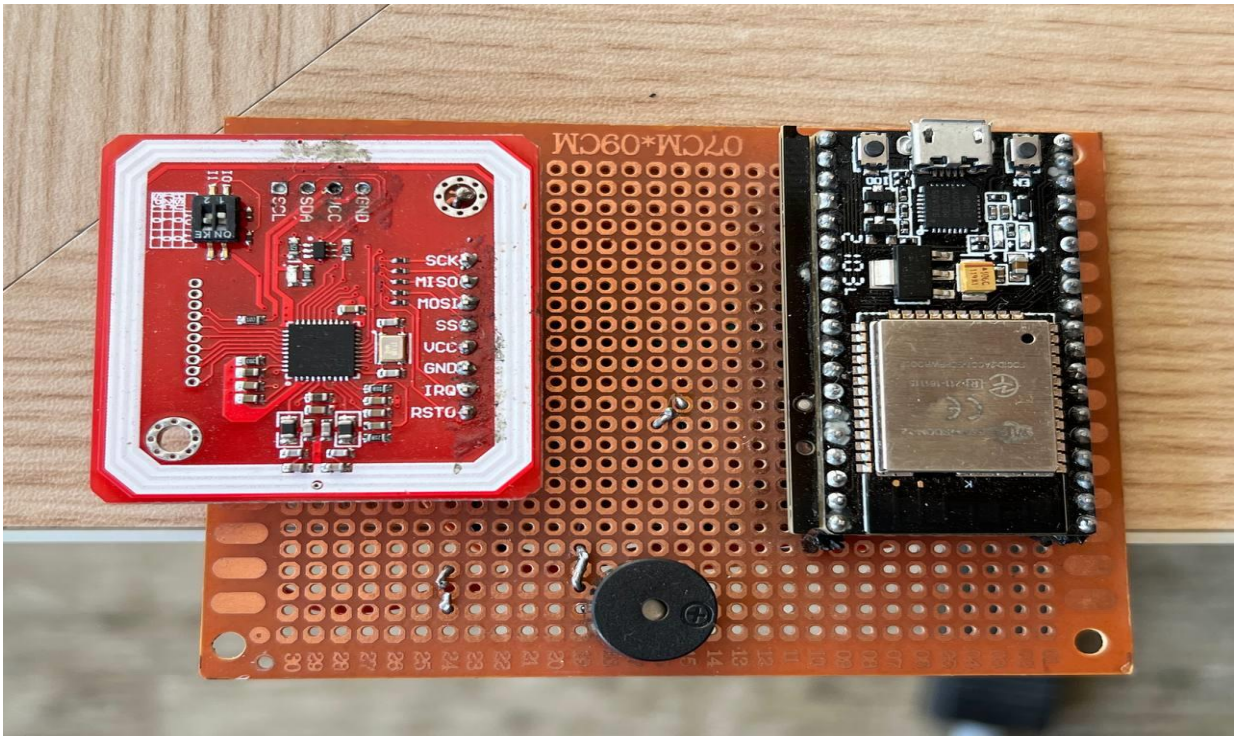


Figure 5.5

The NFC chip on the bracelet is used for contactless payments at the school canteen.
Parents can charge their students NFC chips through the parent application.
When a student wishes to make a purchase at the canteen, they can simply tap their bracelet on an NFC reader.
The NFC reader communicates with the NFC chip on the bracelet to initiate the payment process.
The payment information is securely transmitted to the canteen's payment system.
Allergy Detection and Payment Decline:
Parents can use the parent application to indicate if their child has any allergies to specific food ingredients.
When a student attempts to make a purchase, the canteen worker checks the ordered items against the student's allergy information.
If the ordered item contains allergenic ingredients, the payment is automatically declined to prevent the student from consuming potentially harmful food.

NFC bracelet:

Figure 5.6

The NFC (Near Field Communication) bracelet is a key component of the smart school system project. It combines hardware and software to enable various functionalities within the system. Here's an explanation of how the NFC bracelet works:
Communication with ESP32.
The ESP32 microcontroller serves as the main processing unit for the bracelet, handling data communication and executing software instructions.
The NFC chip allows the bracelet to communicate with other NFC-enabled devices, such as the canteen payment system, parent application and school gate.
The GPS module provides location tracking functionality, allowing parents to monitor their child's whereabouts. (prototype)

Communication with Parent Application:
The NFC bracelet is connected to the parent application via a wireless connection.
Through the parent application, parents can receive real-time updates about their child's location using the GPS functionality of the bracelet.
Parents can also configure the application to receive notifications if their child enters or leaves specific areas, enhancing safety and security.

Canteen Payment System Integration:

The NFC chip on the bracelet is used for contactless payments at the school canteen. Parents can link their credit card information to the NFC chip through the parent application. When a student wishes to make a purchase at the canteen, they can simply tap their bracelet on an NFC reader. The NFC reader communicates with the NFC chip on the bracelet to initiate the payment process. The payment information is securely transmitted to the canteen's payment system, deducting the appropriate amount from the parent's linked credit card.

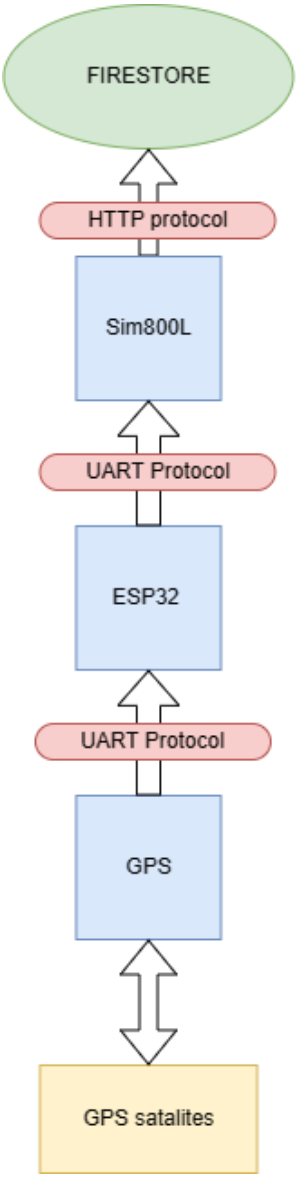


Figure 5.7

Allergy Detection and Payment Decline:

Parents can use the parent application to indicate if their child has any allergies to specific food ingredients.

When a student attempts to make a purchase, the canteen worker checks the ordered items against the student's allergy information. If the ordered item contains allergenic ingredients, the payment is automatically declined to prevent the student from consuming potentially harmful food.

Cloud Firestore Data Storage:

The data collected from the NFC bracelet, such as student location, payment transactions, and allergy information, is stored in a Cloud Firestore database.

Cloud Firestore provides a scalable and secure solution for storing and retrieving data in real-time.

The database allows for efficient retrieval and updating of student information by the parent application, canteen worker section, and teacher section.

Overall, the NFC bracelet plays a crucial role in the smart school system, providing features such as location tracking, contactless payments, allergy detection, and secure data storage. It’s hardware components and integration with the parent application,

canteen payment system, and Cloud Firestore database enable seamless communication and functionality within the system.

GPS MODULE :

In the opposite diagram it illustrates the data flow from the gps module.

As it receives the location data from the GPS satellites and transfers it to ESP32 microcontroller though the UART protocol.

Then the ESP32 passes the Sim800L module also by UART protocol.

Then Sim800L module which is connected to the fire store transfers the collected data from the GPS by HTTP protocol.

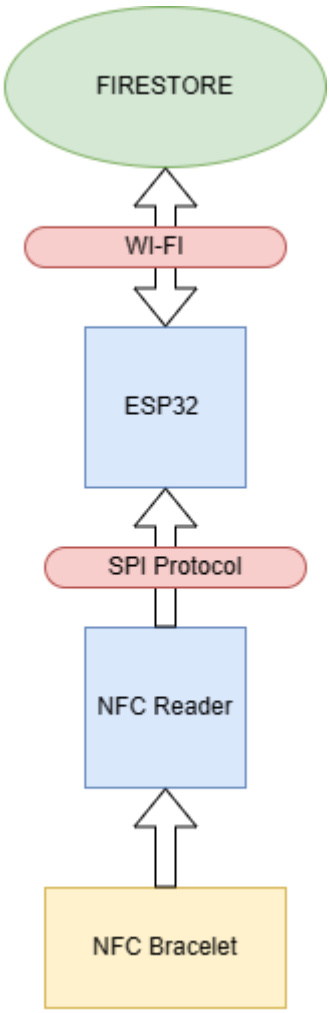


Figure 5.8

NFC module :

In the opposite figure it illustrate the data flow from the NFC reader.

The sudent pass the NFC bracelet through the gate , NFC reader reads the data on the ship then transfer it to the integrated ESP32 microcontroller through SPI protocol .

Then the collected data sent to the FIrestore by the microcontroller through WI-FI.

5.3 System diagrams

5.3.1 USE CASE DIAGRAM

First, we talk about use case diagram, in the opposite diagram(fig.30) explains how the main actors in our system perform.

A use case diagram is a visual representation of the interactions between actors (users or external systems) and the system itself. In the context of the smart school system project, the use case diagram helps us understand the various functionalities and roles involved. Let's explore the main use cases:

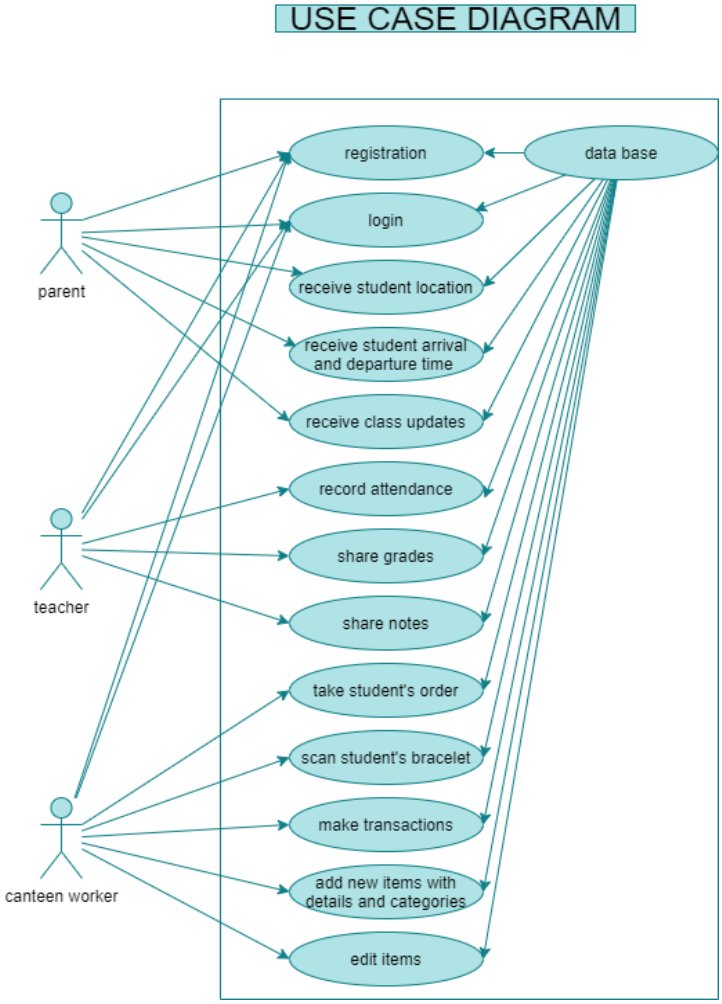


Figure 5.9

1. **Parent**:
    - Registering his children information on the system to initiate the login to the system.
    - View Student Location: The parent can view the current location of their child using the mobile application.
    - Receive their child arrival and departure times and also receive any class notes or updates.
2. **Canteen Worker:**
    - Manage Snacks: The canteen worker is responsible for managing the available snacks. They can add, update, or remove snack items from the system.

- Process Payments: When a student purchases a snack using the NFC bracelet, the canteen worker uses the system to process the payment and complete the transaction.

3. Teacher:
    - Take Attendance: The teacher can use the system to mark the attendance of students in their class. This information is then accessible to parents through the application.
    - Add Notes to Parents: Teachers can add important notes or updates about a student's academic progress or behavior, which parents can view through the application.
4. System:
    - Store Data in Cloud Firestore: The system uses the Cloud Firestore database to store and retrieve data securely. This allows for easy access and retrieval of information by authorized users.
    - Connect NFC Bracelet and NFC Reader: The NFC bracelet worn by students is connected to the NFC reader to facilitate payments for snacks.
    - Track Student Location: The system tracks and updates the student's location using the GPS module in the NFC bracelet.

The use case diagram provides an overview of the interactions between the different actors and the system, highlighting the key functionalities and responsibilities of each party involved in the smart school system project.

5.3.2 DATA FLOW DIAGRAM LEVEL 0

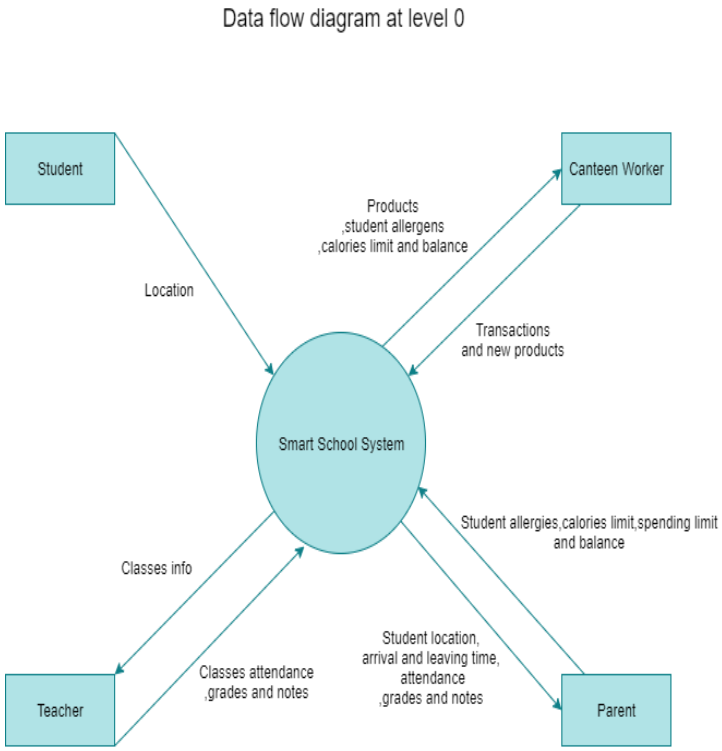


Figure 5.10

The data flow diagram (DFD) illustrates the flow of data within the smart school system project. It represents how information is input, processed, and output by the system. Here's an explanation of the key components of the DFD:

1. Student Data Input:
    - NFC Bracelet and GPS: The student wears an NFC bracelet with a GPS module. The GPS module tracks the student's location, and the NFC chip stores the student's identification and payment information.
2. Parent interface:
    - Location Request: Parents can request the current location of their child through the mobile app.
    - Allergy Information: Parents can input their child's allergy information through the app.
    - Calorie limit: parents can set a limit for their child calories intake.
    - Payment limit control: When a student attempts to make a snack purchase, the payment is approved if there are no allergic ingredients present and the child spending limit is not exceeded.
3. Canteen Worker Interface:
    - Snack Details: The canteen worker updates and maintains the snack details, including the ingredients, prices, and availability.
    - Payment Processing: The canteen worker processes the payment for snack purchases made by students using the NFC chip.
4. Teacher Interface:

- Attendance Recording: Teachers can input the attendance of students through the application interface.
- Parent Communication: Teachers can add notes and communicate with parents through the application interface.

5.3.3 PARENT ACTIVITY DIAGRAM

The parent activity diagram illustrates the sequence of activities and actions that a parent can perform within the smart school system. It outlines the different paths and decisions that a parent can take while interacting with the system. Here's an explanation of the key activities in the parent activity diagram:

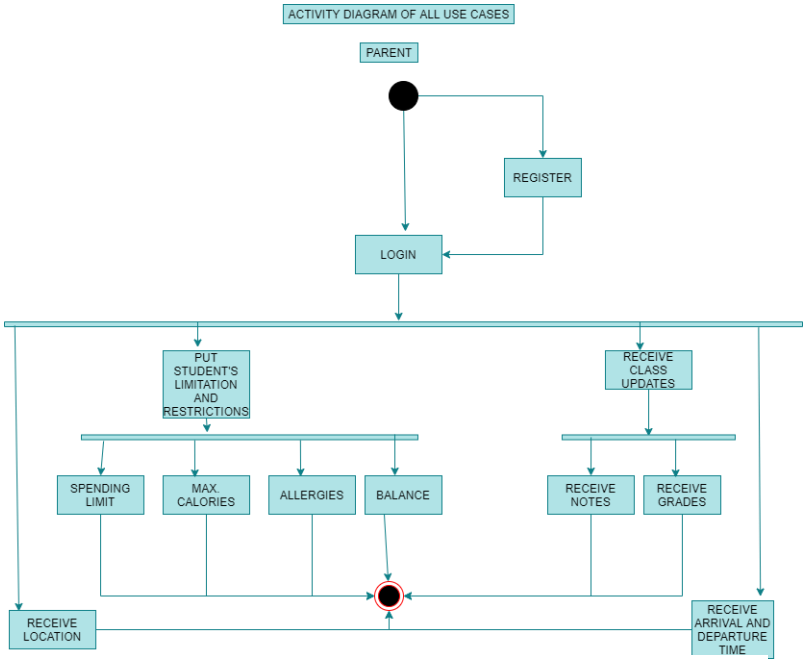


Figure 5.11

1. **Start**:
    - The parent launches the mobile application and logs in.
2. **View Child's Location:**
    - The parent can choose to view the current location of their child.
    - If the parent selects the "View Location" option, the system retrieves the GPS data from the NFC bracelet and displays the child's location on the map.
    - The parent can either continue viewing the location or return to the main menu.
3. **Manage Allergy Information:**
    - The parent can manage allergy information for their child.
    - If the parent selects the "Manage Allergies" option, they can input or update allergy details for their child.
    - The system stores this information for future reference.
4. **View Payment History:**
    - The parent can access the payment history of their child's snack purchases.

- If the parent selects the "View Payment History" option, the system retrieves and displays transaction records.
- parents can view details such as the date, time, and amount spent on snacks.

5. **Receive Class Updates:**
    - The parent can receive updates and notifications from the child's teacher regarding class-related information.
    - If the parent selects the "Receive Class Updates" option, the system displays the messages or notifications sent by the teacher.
    - The parent can read the updates and choose to reply or take further action if necessary.
6. **View Arrival and Departure Time:**
    - The parent can view the scheduled arrival and departure time of their child.
    - If the parent selects the "View Arrival/Departure Time" option, the system retrieves and displays the scheduled times for the child's school arrival and departure.
    - parents can plan their schedule accordingly based on this information.
7. **Set Spending Limit and Calorie Intake Limit:**
    - parents can set spending and calorie intake limitations for their child's snack purchases.
    - If the parent selects the "Set Limits" option, they can specify the maximum amount their child can spend on snacks and the maximum calorie intake for each transaction.
    - The system enforces these limits, ensuring that the child cannot exceed the defined thresholds.
8. **End:**
    - The parent chooses to log out of the application, ending their session.

5.3.4 TEACHER ACTIVITY DIAGRAM

The teacher activity diagram illustrates the sequence of activities and actions that a teacher can perform within the system, it outlines different paths and decisions that a teacher can take while interacting with the system. Here`s and explanation of the key activities in the teacher diagram:

1. Start:
    - The teacher logs in to the system using their credentials.
2. Take Attendance:
    - The teacher can take attendance for their class.
    - If the teacher selects the "Take Attendance" option, the system displays a list of students enrolled in the class.
    - The teacher marks the attendance status (e.g., present, absent) for each student.
    - The system updates the attendance records accordingly.

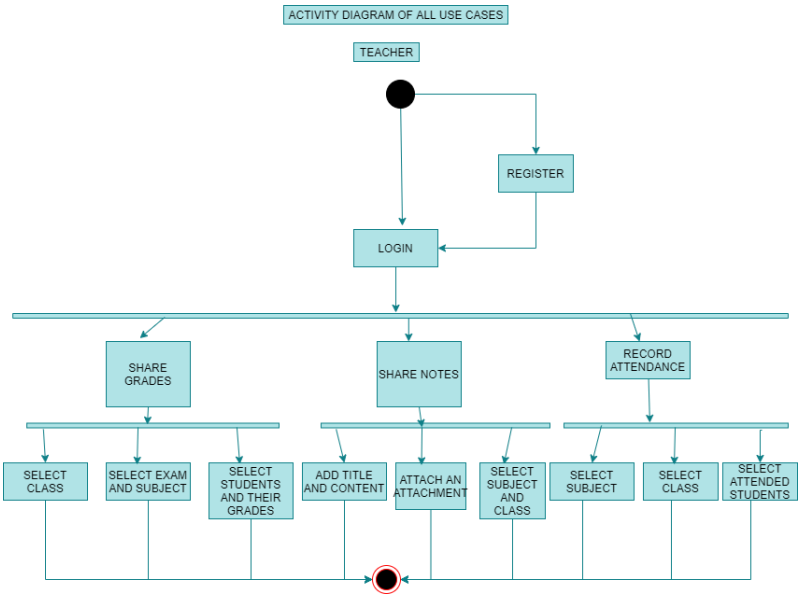


Figure 5.12

3. Add Notes to Parents:
    - The teacher can add notes or communicate with parents regarding their child's performance, behavior, or any other relevant information.
    - If the teacher selects the "Add Notes to Parents" option, the system provides a text input field or a messaging interface.
    - The teacher writes the note or message, selects the recipient parents, and sends the communication.

- The system stores the notes and notifies the parents accordingly.

4. Share Grades with Parents:
    - The teacher can share student grades and progress with parents.
    - If the teacher selects the "Share Grades" option, the system provides access to the grade book or assessment records.
    - The teacher selects the relevant assessment or grading period and chooses the parents to share the grades with.
    - The system sends grade notifications or provides secure access to view the grades through the parent interface.
5. Upload Attachments:
    - The teacher can upload and share attachments or documents with parents or students.
    - If the teacher selects the "Upload Attachments" option, the system provides a file upload interface.
    - The teacher selects the file(s) to upload, specifies the intended recipients, and uploads the attachments.
    - The system stores the attachments and notifies the recipients about the availability of the files.
6. Generate Reports:
    - The teacher can generate reports related to student performance, attendance, or any other required metrics.
    - If the teacher selects the "Generate Reports" option, the system provides a set of report options.
    - The teacher selects the desired report type and parameters.
    - The system processes the data and generates the report, which can be saved, printed, or shared.
7. End:
    - The teacher chooses to log out of the system, ending their session.

5.3.5 CANTEEN WORKER ACTIVITY DIAGRAM

The canteen worker activity diagram illustrates the sequences of activities and actions that he can take within the system. It outlines the different paths and decisions that are available to take while interacting with the system. Here`s an explanation of the key activities in the diagram:

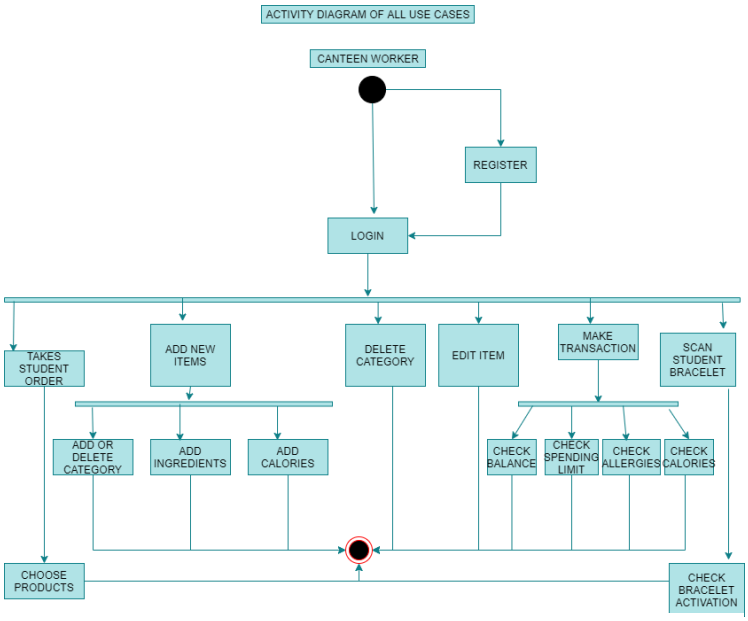


Figure 5.13

1. **Start:**
   - The canteen worker logs in to the system using their credentials.
2. **View Orders**:
   - The canteen worker can view the list of pending orders placed by students.
   - If the canteen worker selects the "View Orders" option, the system displays the orders with details such as the student's name, order items, and payment status.
3. **Check Balance, Allergies, and Calorie Limit:**
   - Before processing the payment, the canteen worker checks the student's account balance, allergies, and calorie limit.
   - If the student has sufficient balance and no allergies or calorie limitations for the selected snacks, the process continues. Otherwise, appropriate actions are taken (e.g., notifying the student, suggesting alternative snacks, or declining the order).
4. **Prepare Snacks**:
   - The canteen worker prepares the snacks or food items based on the received orders.

- If the canteen worker selects the "Prepare Snacks" option, they access the inventory or menu management system.
- The canteen worker checks the availability of ingredients and ensures that all necessary items are present.
- The snacks are then prepared according to the specified orders, following the recipes and portion sizes.

5. Mark Order as Ready:
    - Once the snacks are prepared, the canteen worker marks the order as ready for pickup.
    - If the canteen worker selects the "Mark Order as Ready" option, the system updates the order status to indicate that it is ready for pickup by the student.
6. Process Payment:
    - After confirming the balance, allergies, and calorie limit, the canteen worker proceeds to process the payment.
    - If the canteen worker selects the "Process Payment" option, the system provides a payment interface.
    - The canteen worker requests payment from the student or scans their NFC bracelet to initiate the transaction.
    - The system verifies the payment information, deducts the amount from the student's connected parent's credit card, and updates the payment status.
7. End:
    - The canteen worker chooses to log out of the system, ending their session.

5.3.6 ERD Entity Diagram

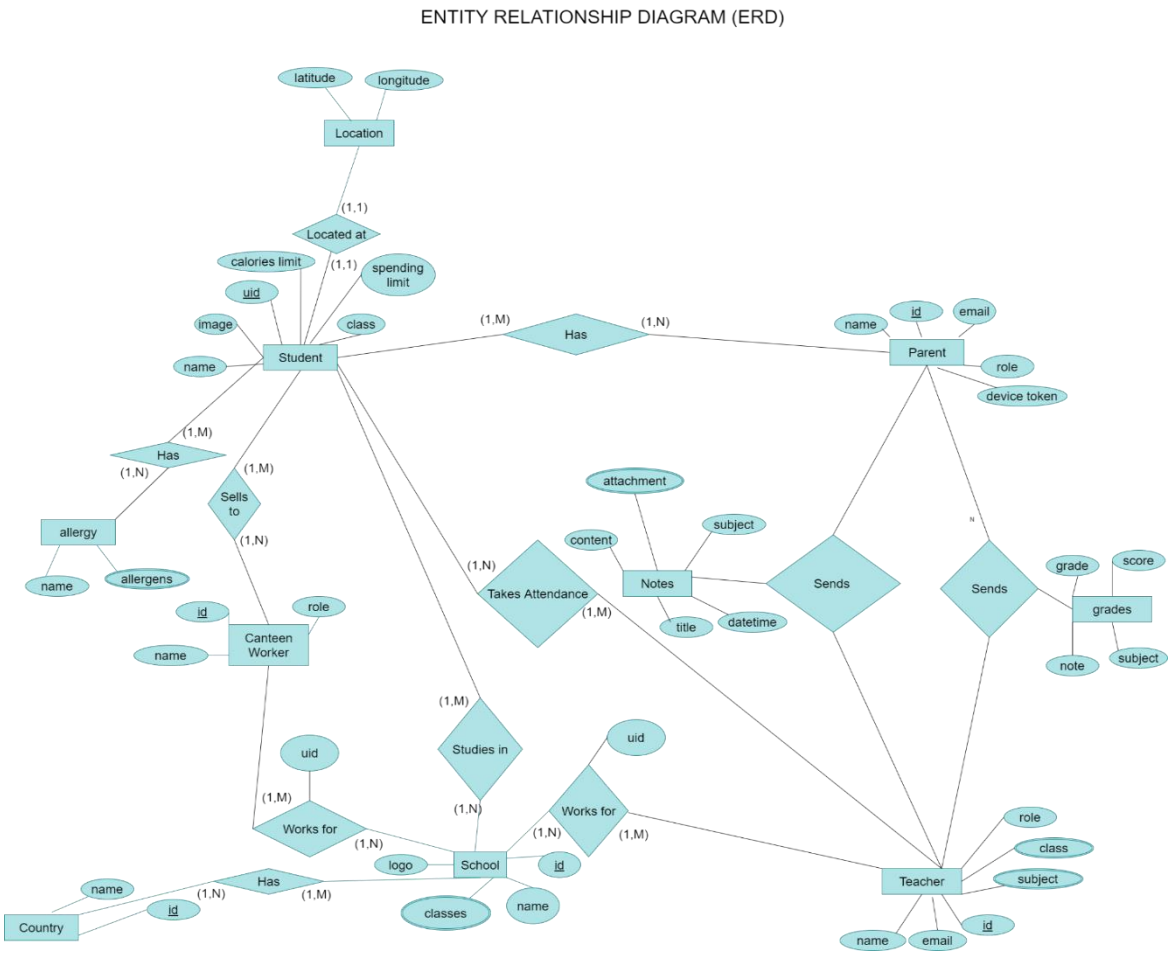


Figure 5.14

- An ERD, or Entity-Relationship Diagram, is a visual representation of entities and their relationships within a database. It is a modeling tool used to design and document relational databases.
- In an ERD, entities are represented by rectangles, and relationships between entities are represented by lines connecting the rectangles. Each entity has attributes that describe the characteristics or properties of the entity, represented by ovals connected to the entity rectangle.
- The three main components of an ERD are entities, attributes, and relationships. Entities represent objects or concepts within the database, attributes describe the

properties of entities, and relationships represent the connections or associations between entities.

- There are several types of relationships that can be represented in an ERD, including:
    1. One-to-One (1:1): In this relationship, each instance of one entity is associated with exactly one instance of another entity.
    2. One-to-Many (1:M): In this relationship, each instance of one entity is associated with one or more instances of another entity.
    3. Many-to-One (M:1): In this relationship, many instances of one entity are associated with exactly one instance of another entity.
    4. Many-to-Many (M:M): In this relationship, many instances of one entity are associated with many instances of another entity.
- ERDs are commonly used in the database design process to help developers and stakeholders understand the structure of the database. They can also be used to validate and optimize the database design, and to communicate the design to other members of the development team.

6.0 Mobile Application

6.1 PARENT INTERFACE

Firstly, the **welcome screen**:

The welcome screen is the first screen the user faces if he hasn't logged in to the app, after that If the parent has registered to the app the parent enters his email address and password and chooses the parent role.

There is an option to show the password after typing it as dots. Just in case the parent forgot his password, he could press (Forget password?) text button to navigate to the **FORGET PASSWORD SCREEN** which is the one on the right. Then entering his email address press reset then the password will be reset.

After resetting the password, the parent can login. In case the parent doesn't have an account, then he will have to press (Register Now) that will navigate to the registration screen that will be discussed in the next slide.

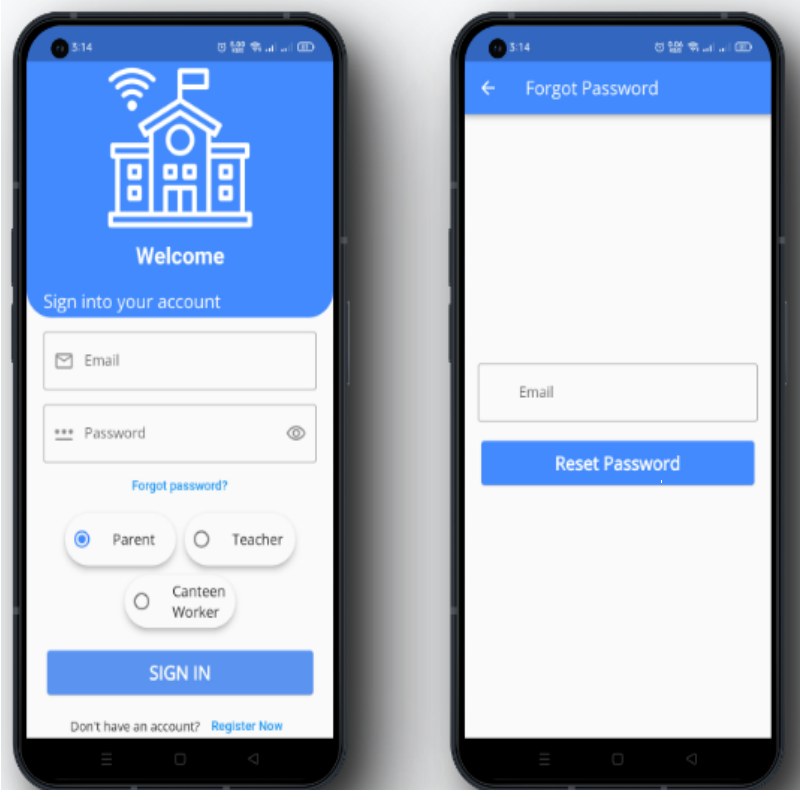


Figure 6.1

Registration:

The registration screen is the screen at which the parent registers an email and password on the app, also he enters his name as related data to his account.

There is an option to show the password after typing it as dots. Moreover, the parent should select the parent role to make the email as a parent role.

After that the parent presses the (SIGN UP) button, it will be making sure that the email is valid.

Checking the validation of the email is by checking if the entered email is used by another user or not and if not registration of the email will be successful.

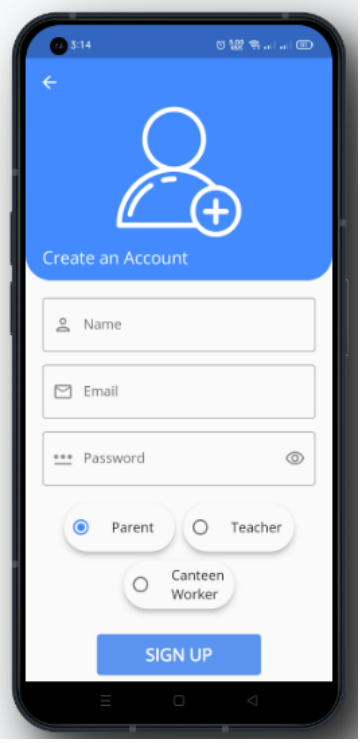


Figure 6.2

Home screen:

Home screen is the first screen the parent face after login, At the top left there is a navigation drawer icon that we will discuss later the use of it. Also, at the top right there is a notification icon that we will discuss later the use of it.

We can see that the email has got a balance of (1,843.00) that we will discuss later how to charge your balance.

We also can find (Add Family Member) that shows that the parent hasn't linked his child to the app and in next slide we will discuss what happen after pressing it.

Finally, there is no activity found picture that indicates that because of not connecting a child there is no activity is shown.

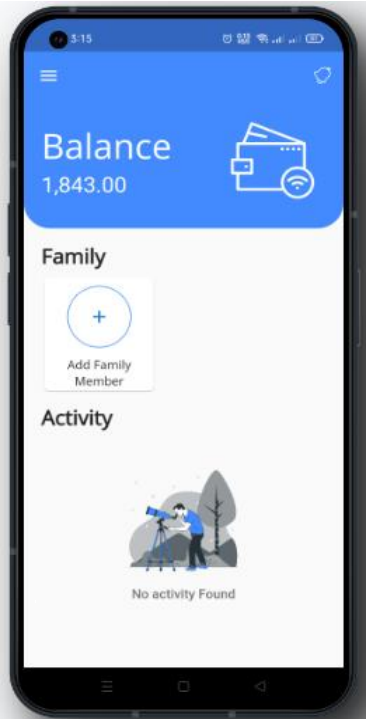


Figure 6.3

Adding family member (student) to parent account:

After the parent presses the (Add Family Member) button the screen at the left appears, this screen shows the countries that the school is found in.

Then Parent should pick up the country at which his child lives in and press it, after that it will navigate to the screen on the right.

The screen on the right lists the schools' names that can be found in the selected country, Parents should select the school that their child belongs to and select it.

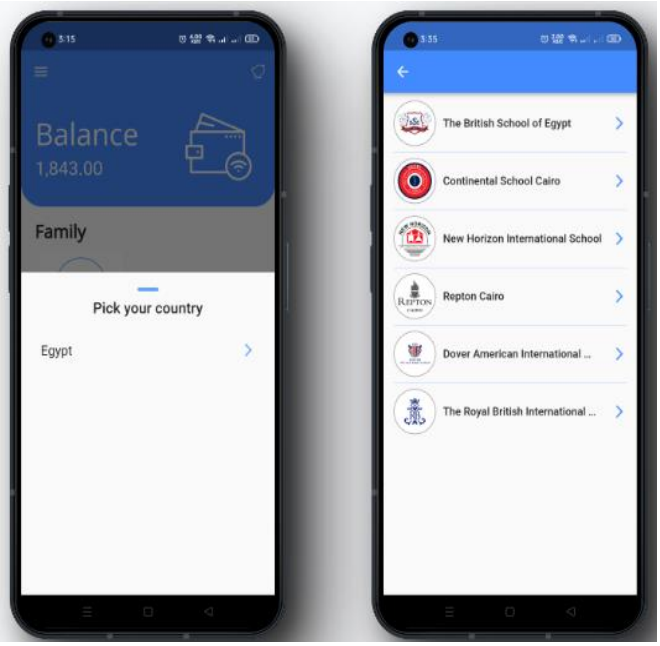


Figure 6.4

Find your family member screen.

After selecting the school, it will navigate to the Find your family member screen.

At this screen the parent should enter the ID of his child. This ID is sent to the parents by school. The Parent should do these steps again if he has more than on child.

After entering the ID and pressing confirm we can see that a family member is added with his name and image, this happens after doing the steps in the previous slide.

If the student ID is connected to another parent there will be a message that says that that this student ID is already connected to another parent.

If the student ID is entered and confirmed successfully that will navigate to the home screen.

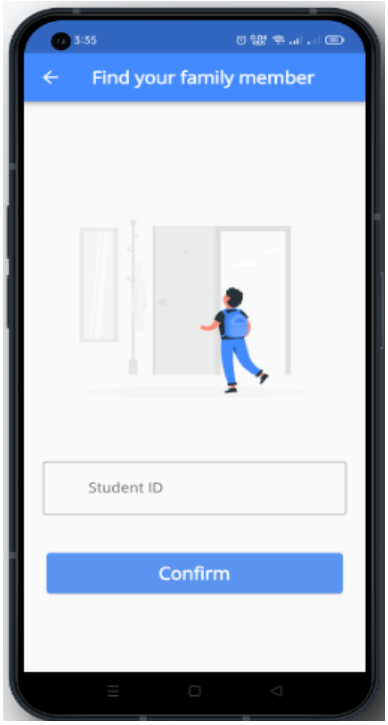


Figure 6.5

Charging the account screens

By pressing the navigation menu, the menu in the screen on the left will appear to the parent. That shows his profile including his name and his account and the settings.

The settings include adding family members, recharge, clearing history and sign out from the application. By pressing (recharge) this will navigate to the screen on the right.

For parents to charge his account he should enter the credit card number, expiration date, the card holder name, and the recharge amount. Then by pressing recharge his account will be recharged.

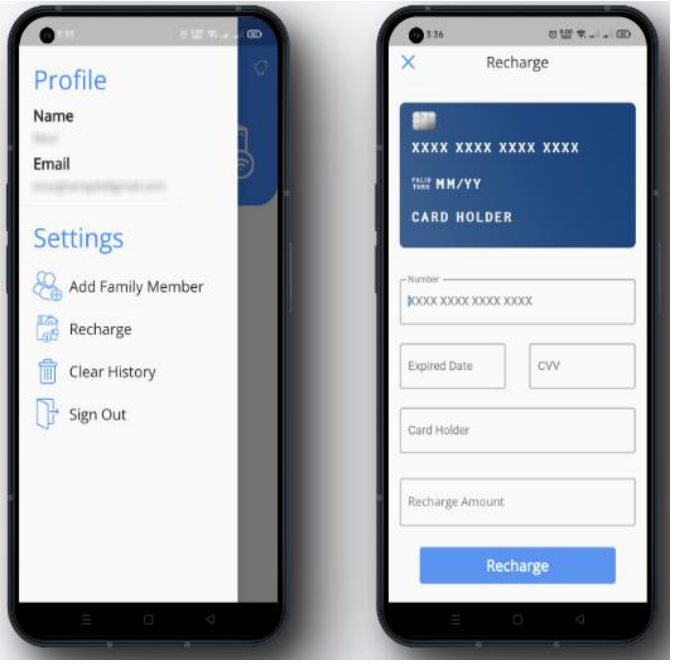


Figure 6.6

Recharging transaction screen and blank notification screen

After the steps in the previous slide and recharge amount of 100 and pressing recharge that will navigate to the screen on the left that indicates that the transaction is processed successfully. Then parents should press okay to get back to the home screen.

By pressing the notification button on the top right, that will navigate to the notification screen which is the one on the right. In this case there are no notifications available.

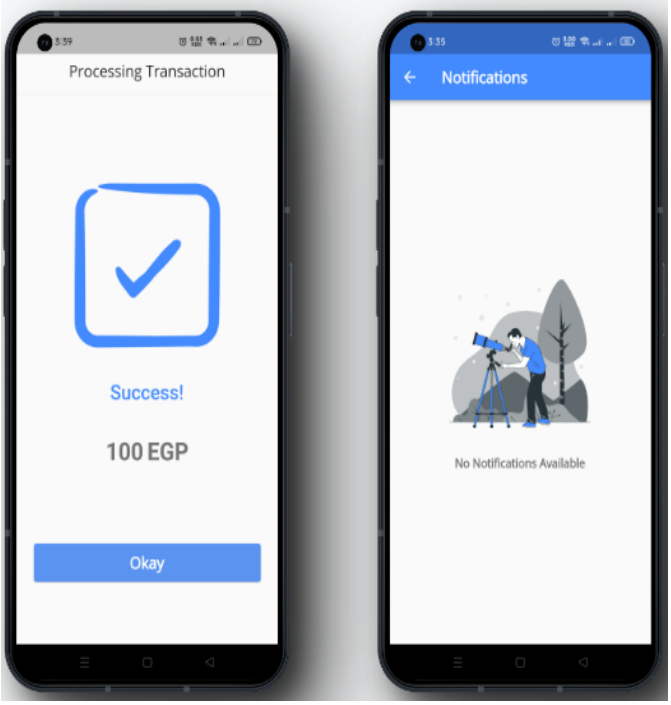


Figure 6.7

Home screen with activities and notification screen with notifications

The screen on the left is a home screen with some activities done by the student.

The first activity is the departure of the student from the school, The second activity is that the student purchased products from the school canteen with total price 16, and the third activity is the arrival of the student to the school. By pressing on the notification icon on the top right that will navigate to the notifications screen on the right, In the notification screen there are different types of notifications:

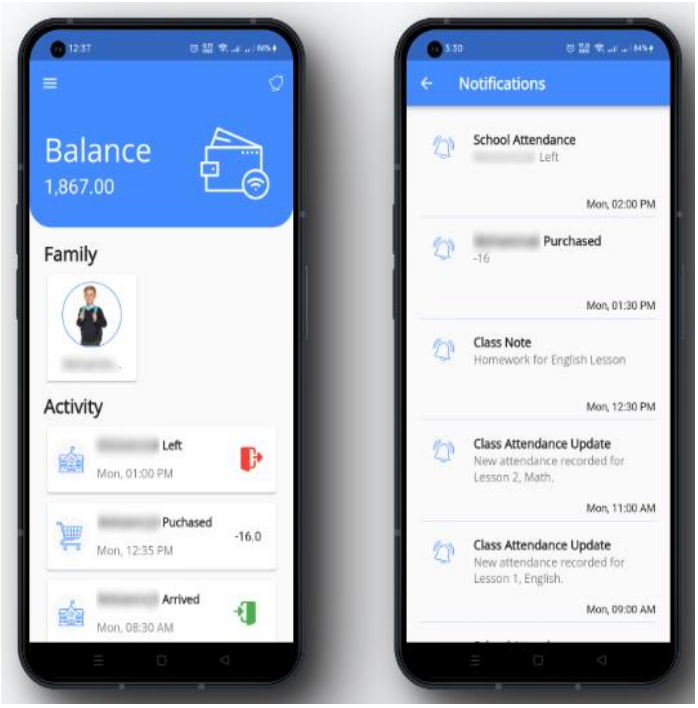


Figure 6.8

1ST

Notification mentions the departure of the student from the school.

2ND

notification is that the student purchased products from the school canteen with total price 16.

3RD

notification indicates that there is a class note uploaded by the English teacher and it is a homework.

4TH

notification informs that the student has attended a math class which was about lesson 2.

5TH

notification informs that the student has attended an English class which was about lesson 1.

Transaction details screen

In the notifications screen, by pressing on the purchasing notification that will navigate to the transaction details screen, In the transaction details screen, everything about the transaction becomes clear. First, the total price of the transaction. Second, the date and time of the transaction.
Moreover, the products that had been bought. Mentioning the name of the product, the price of each product and how many items are available of each product.

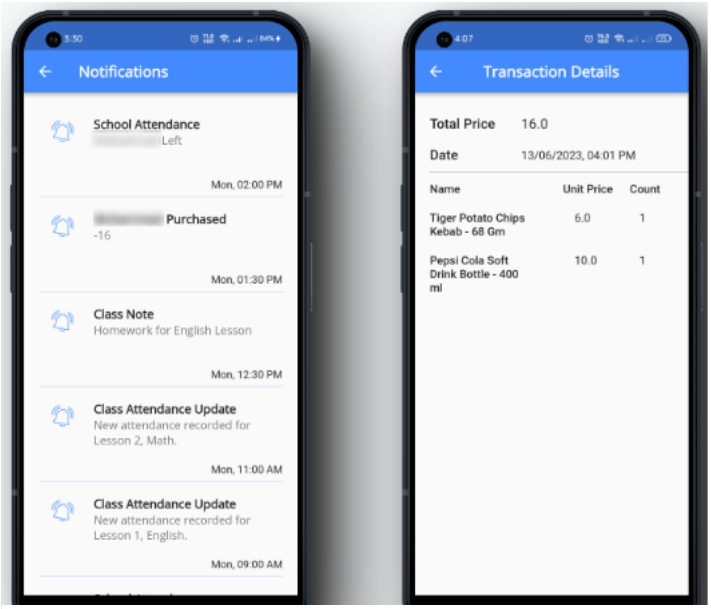


Figure 6.9

Student community screen

By pressing on the family member item that will navigate to the student community which is the screen on the left.
The student community screen contains two options to select which are the settings and the class updates.
by pressing on settings that will navigate to the settings screens while pressing on class updates will navigate to the class updates screen that will be discussed later.

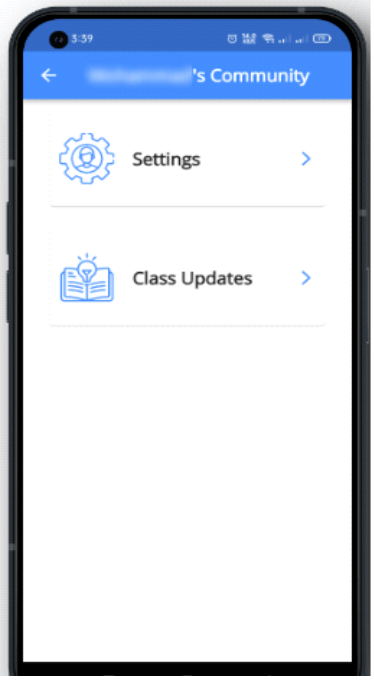


Figure 6.10

Students settings screen

By pressing on the settings option that will navigate to the settings screen the one on the left, we will discuss everything in the setting screen later, but we will start with the digital ID number activation.

By default, the digital ID number is activated which means the bracelet of the student can make transactions and every other operation, by pressing on the switch button that will be deactivated which means the bracelet cannot make transactions or any other operation like recording arrival and departure from the school.

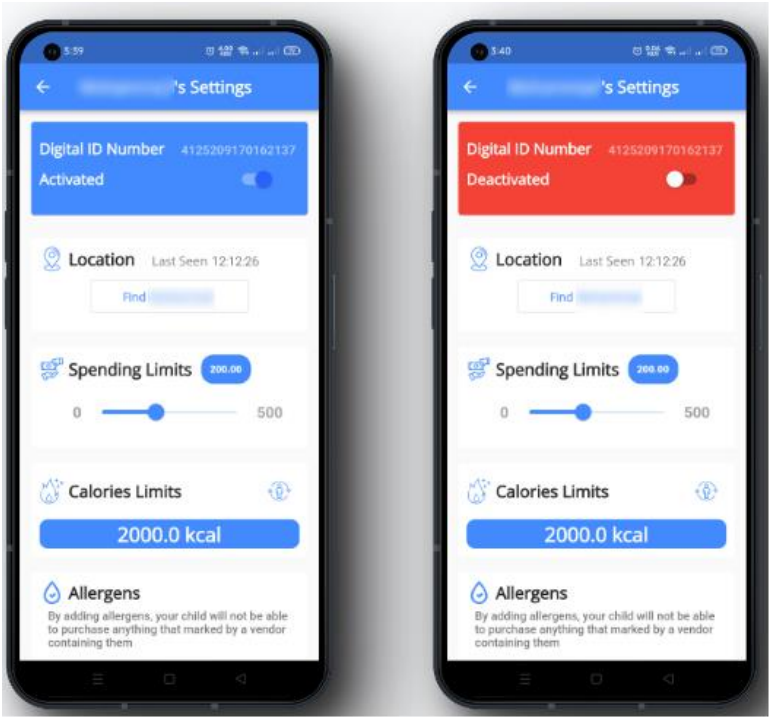


Figure 6.11

The deactivation is often used when the student bracelet is lost.

deactivation screen

This screen shows deactivation of a digital ID number and informs the parent that by deactivation he won't be able to spend or load up the account as mentioned in the screen.

Whereby pressing (YES, I'M SURE) the bracelet will no longer be usable as if the student tries to scan it at the gate it won't send a notification to the parent and if he tries to buy something from the canteen it won't make the transaction.

So, the parent must make sure that he wants to deactivate the bracelet before pressing (YES, I'M SURE).

The case in which it is useful to deactivate the bracelet is when the bracelet is lost.

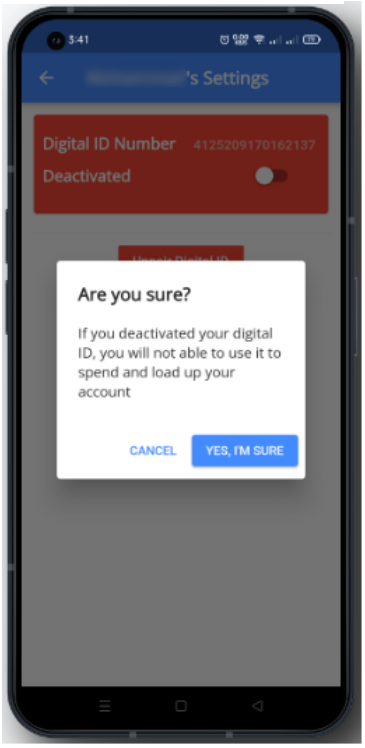


Figure 6.12

Student settings and maps screens

The screen on the left is the student setting screen, The screen shows 'Location', 'spending limit', 'calories limit', 'allergens'.

By pressing on (find student) button that will navigate to the maps screen, in maps screen the location of the student is found on maps, The location of the child can be found anytime he is wearing the bracelet.

Also, the parent can put a limit for the child's spending, which makes something like pocket money for the child that cannot spend more than that limit.

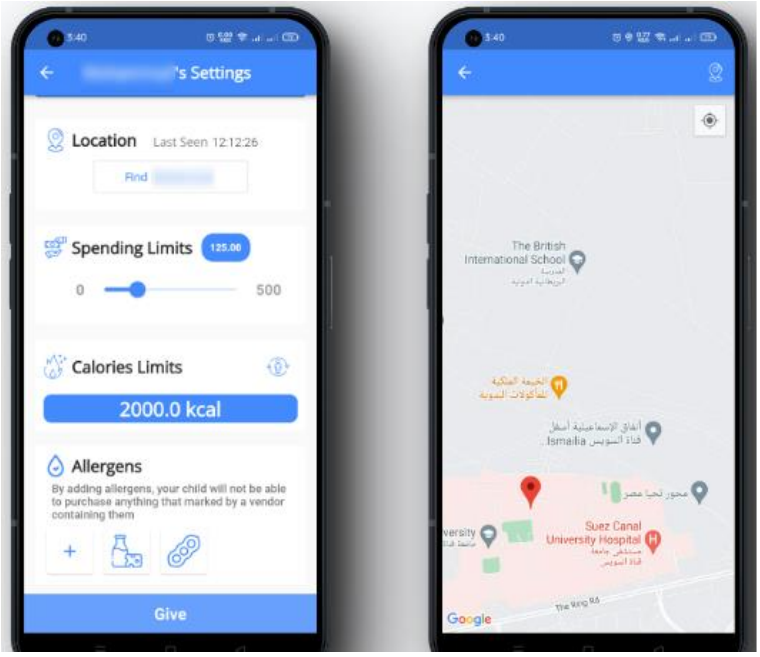


Figure 6.13

Update calories limit screen and allergens screen

By pressing on the button under 'calories limit' word in the student setting screen that will navigate to the update calories limit screen the screen on the left.

In the update calories limit screen, the parents should set a realistic calorie limit for his child based on his age, gender, height, weight, and activity level as mentioned in the screen, it is also recommended to consult a doctor, The

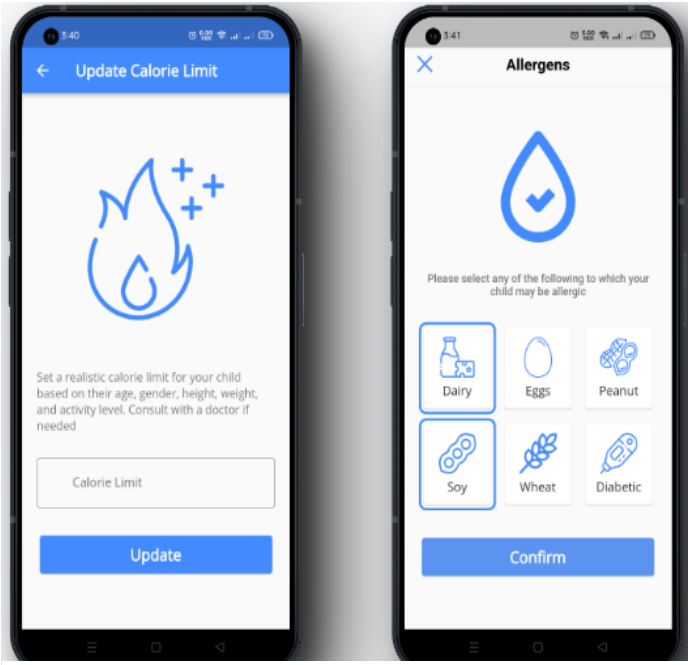


Figure 6.14

parent should enter the calories limit in the field.
After that the parent presses update and then the child won't be able to buy products that their calories exceed this calories limit.
By pressing the button under 'allergens' word in the student setting screen that will navigate to the allergens screen the screen on the right, In the student setting screen, it mentioned that by adding allergens the child will not be able to purchase anything that marked by the vendor containing them. In the allergens screen the parent has any of the listed allergens that could be allergic to the child as mentioned in the screen, after selecting allergens the parent selects confirm button.

Attendance history screen

In the home screen the one on the left, by clicking on the student leaving item or on the student arrival item that will navigate you to the attendance history screen the one on the right.
In the attendance history screen every event of the student as arrival or departure is recorded, there are three columns that describe each event. the first column is the day of the event and its date, The second column is the time of the event, The third column is the type of event such as departure or arrival.

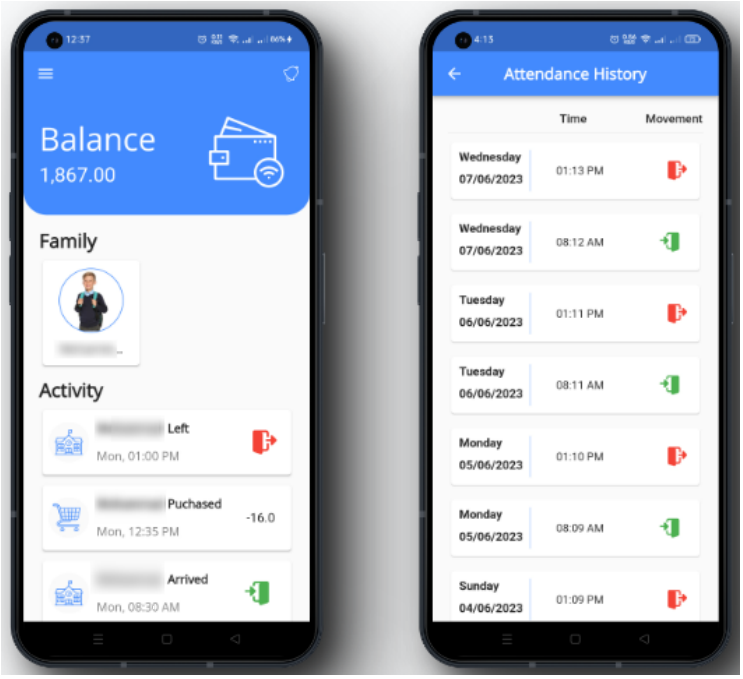


Figure 6.15

The event with red color indicates a departure while the one with the green color indicates the arrival.

Class updates screen

In the student's community screen when pressing on the class updates option that will navigate to the class update screen.

In the class update screen, there are three aspects about the student class, the first is class notes whereby pressing it will navigate to class notes screen that lists the notes of the class, the second is the class attendance whereby pressing it will navigate to class attendance screen that lists the lessons that the student has attended, the third is the class grades whereby pressing it will navigate to class grades screen that lists the grades of every exam and subject.

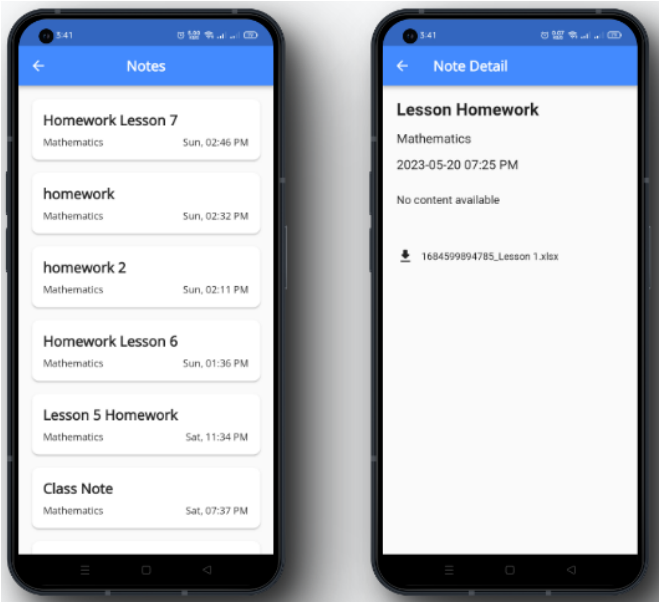


Figure 6.16

It also shows the recent event in every aspect for example, the recent class note that had been sent is [homework lesson 7], the recent class attendance that had been recorded is [lesson 8] and the recent grade that had been announced is [Midterm].

The notes screen and note detail screen.

In the class updates screen, by pressing the class notes option that will navigate to the notes screen.

In the notes screen the activities uploaded by the teacher of the class are listed, the title of the note is mentioned, under the title the subject is mentioned, beside the subject the date and time of sending the note is mentioned. By pressing on the note that will navigate to the note details screen.

In the note detail screen, the title of the note is mentioned. The subject, the date and time of sending the note are mentioned. There is a file attached to the note which can be downloaded.

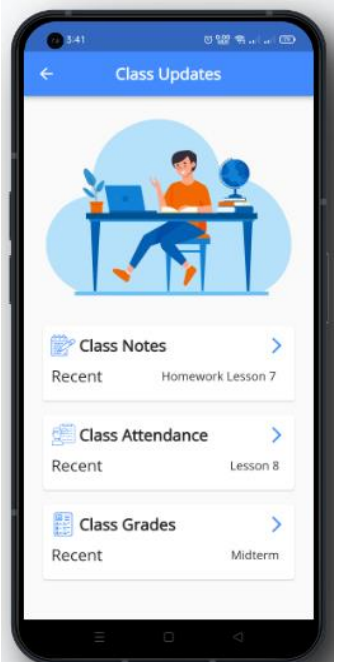


Figure 6.17

Class attendance screen and grades screen

In the class update screen by pressing on the class attendance that will navigate to the class attendance screen, in the class attendance screen the attendance of each class is recorded in a list.

Each attendance cell contains the lesson at the top left. under the lesson, the subject is mentioned. Also, the day and the date are mentioned, there are a right icon or wrong icon are in each cell, the right icon indicates that the student had attended the lesson, the wrong icon indicates that the student hadn't attended the lesson.

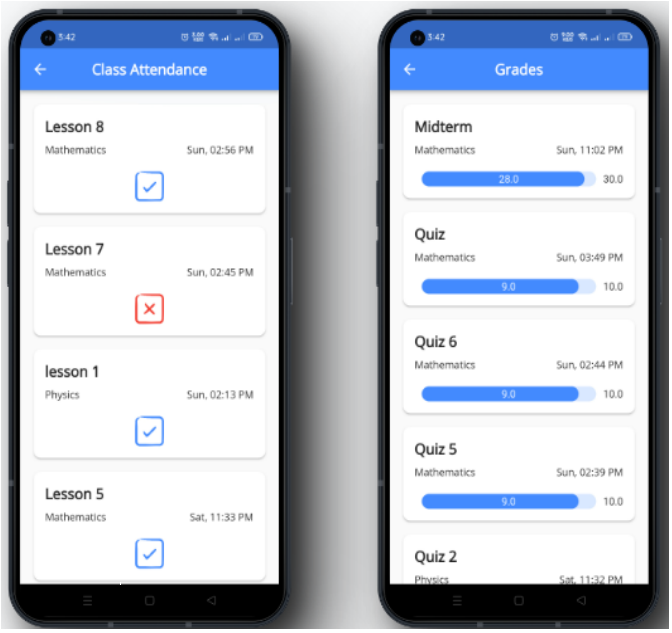


Figure 6.18

In the class update screen by pressing on the class grades that will navigate to the class grades screen, in the class grades screen the grades of exams are listed. in each cell, at the top is the exam type, under the exam type is the subject. Beside the subject is the day and time of the exam, in the bottom there is the grade of the student and the maximum score beside it.

6.2 TEACHER INTERFACE

Teacher overview:

This section discusses the role of teachers in our app and how they interact with their student's attendance, grades and assignments. Our vision is to make it easier and time saving for teachers to do the routine work in school and display that data in a fine way that parents can interact with it easily. Each teacher has his unique access to the material that he teaches giving him the capability to not only take the students attendance more easily but also to share the quizzes and exams grades with the student's parents in a friendly visionary table which saves them a lot of time and effort. Teachers could also send their notes to their students such as homework and assignments.

Joining our community:

Our app gives teachers the ability to join our community and choose between lots of options such as the country that they are in and the schools that they teach in. Teachers may work in one or more schools, and our job is to give them an easy, safe, and fast maneuver between the schools that they work in.

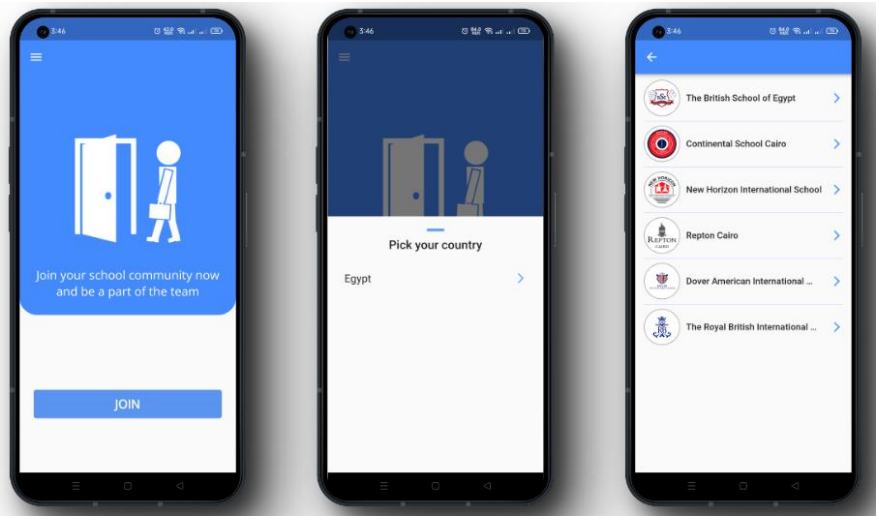


Figure 6.19

Teacher ID:

Each teacher has his unique in that gives him safe access to his work, this id is given by the system administrator and cannot be shared with any unauthorized individual.

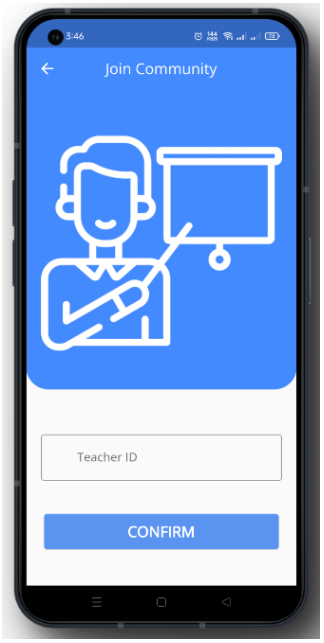


Figure 6.20

Main Menu:

The main screen (as shown in fig 3) is the main menu which gives the teacher several options to choose from. The teacher can share the grades of exams or quizzes, he can also record the attendance of students in class, moreover he can share notes such as assignments or homework for his students and finally he can check all the records that he had done using the application. Each option will be further discussed.

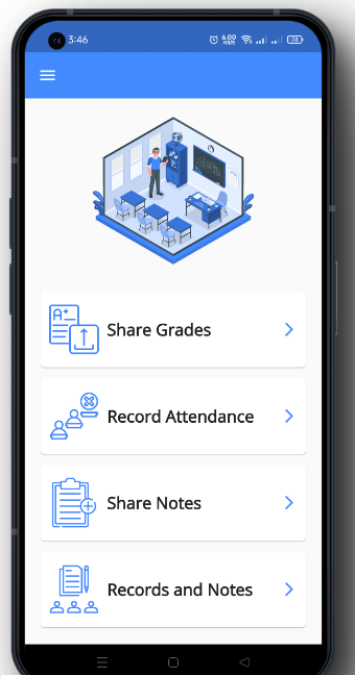


Figure 6.21

Sharing Grades:

On this section, the teacher submits the grades of each student in any type of exam such as quizzes, midterms or final exams. (As shown in fig 4) each teacher must choose the subject that he's teaching and the class that those grades belong to. He must also write the grades in an excel sheet form that is represented in 3 tables, one for student's id, one for student's name and the last one for his grades. We offer our teachers a ready template if they don't have one. They also need to enter the type of exam that they want to record its grades and the maximum achievable grade that each test should have and then attach that excel file that has all the student's data, and the app will reformat it.

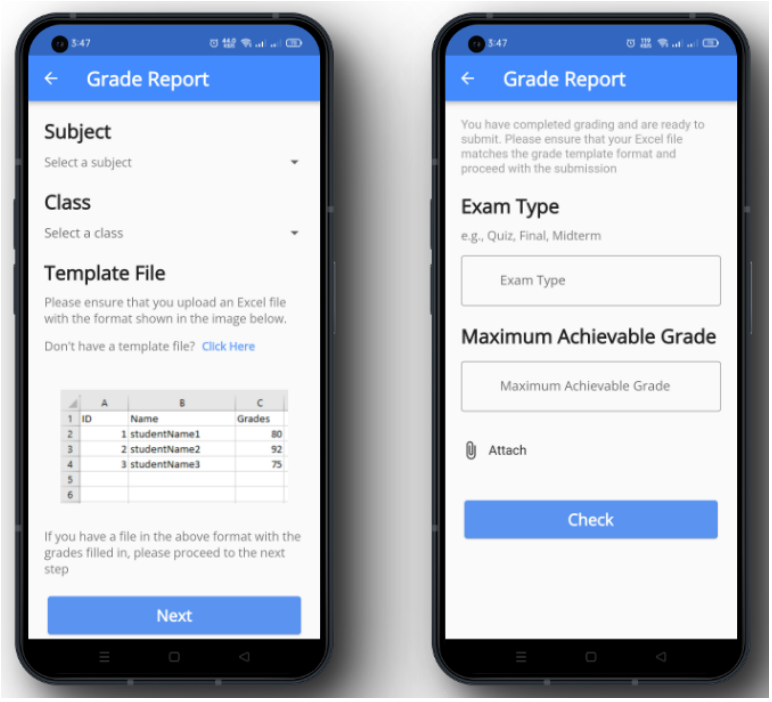


Figure 6.22

Editing Grades:

Here shows the format that the application extracted from the sheet provided by the teacher. The teacher has full access to edit any grade before uploading it, for example if is there is any incorrect syntax or wrong inputs, he can edit it. And then finally he uploads the grades.

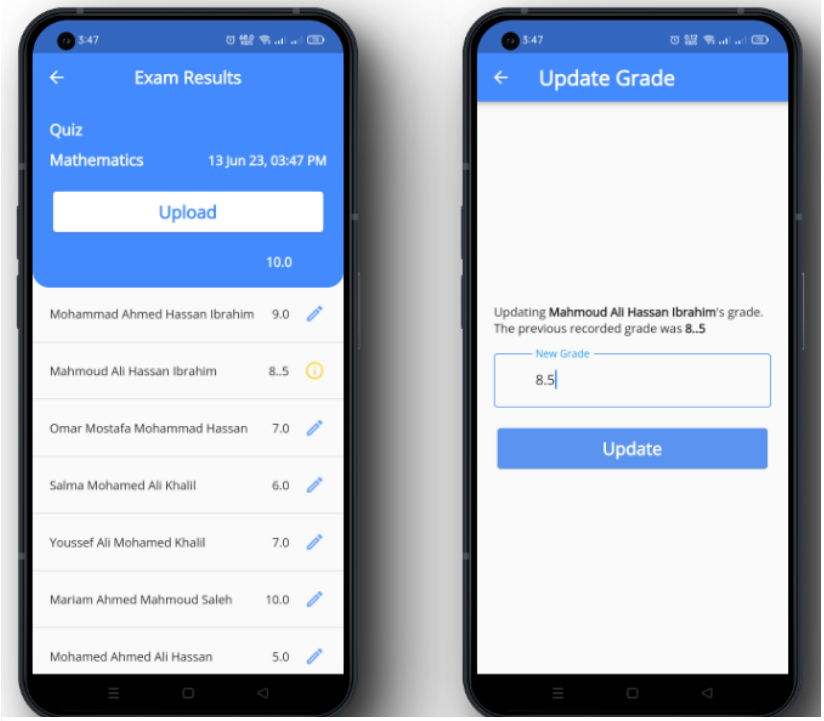


Figure 6.23

Taking Attendance:

Taking the Attendance is the one of the main purposes this app was made. Where each parent can track his son and see whether he attended or not for his son's safety. Teachers create a name for the lesson they are about to explain for example (lecture 1), then choose the subject that they are teaching and the class name so that that a file is created with this data. Each teacher manually takes the attendance for each student. The outline student border goes blue if the teacher ticks on true and goes red if the teacher ticks on false.

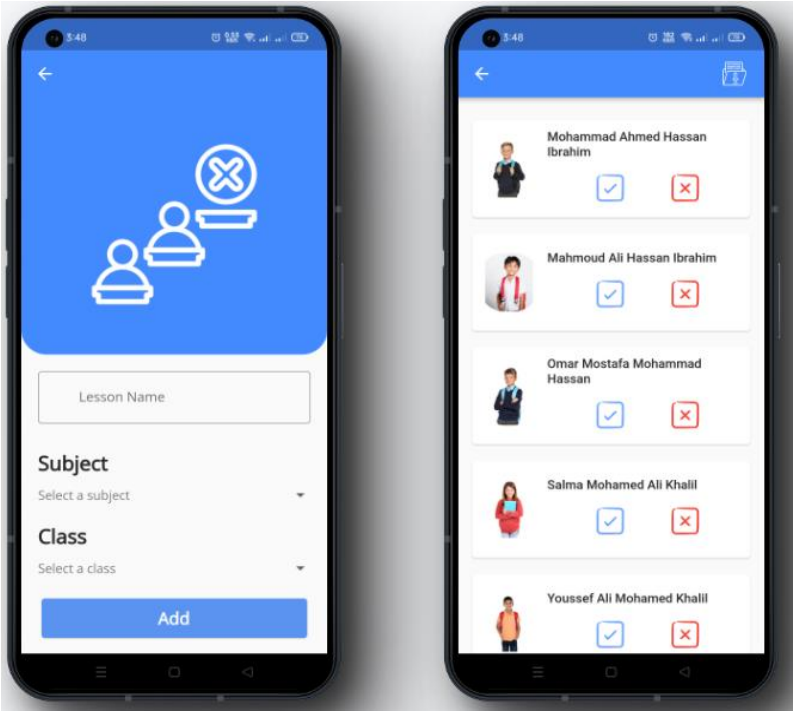


Figure 6.24

Sharing Notes

This section is confined to sending notes for students with their homework or assignments that must be done at home. The teacher must type in the title of the assignment that he needs to make and the content of the assignment, he could also attach a file of the assignment. The app gives him permission to send it to all the students in the class or specific ones in specific class. Notifications are sent to the student's parents of the work that needs to be done.

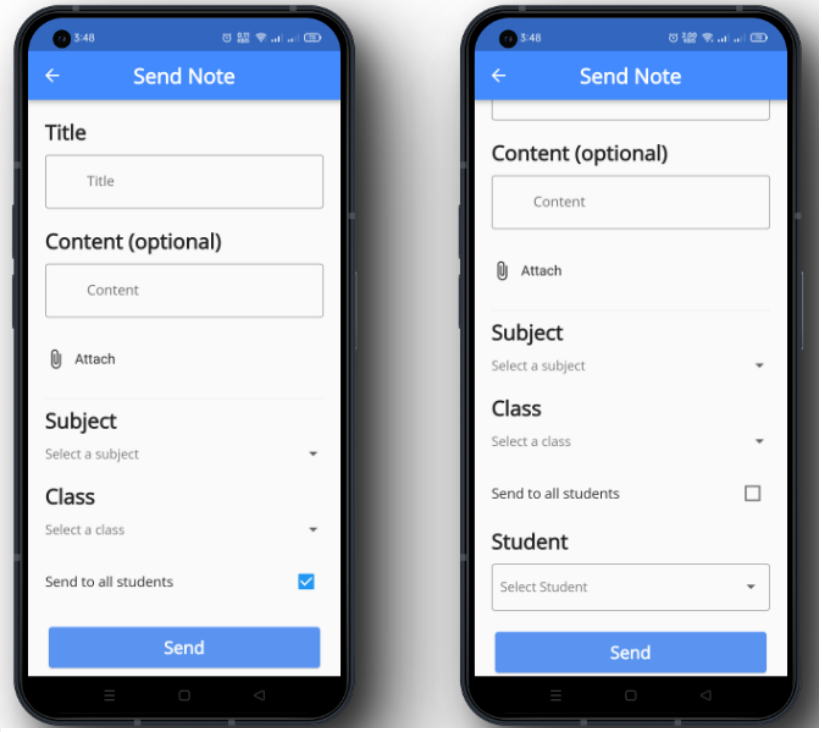


Figure 6.25

Records and Notes:

This section has all the data that the teacher inputs across all the fields of the application (notes, grades and attendance). It is formatted by the date of the day. You can choose the date of the day on which you want your data to be listed.

.

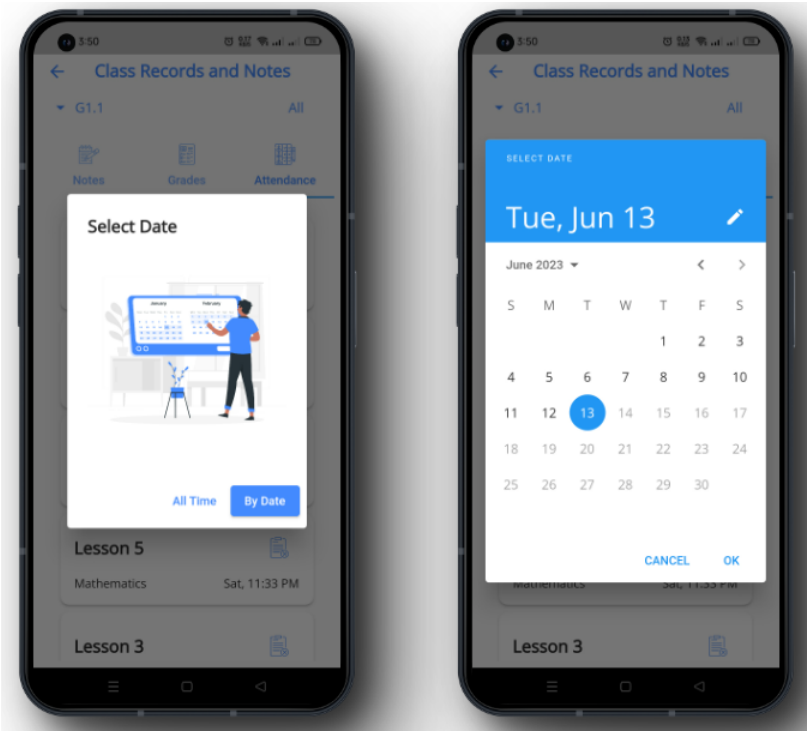


Figure 6.26

Notes Records:

Data will now be listed to all (all days the teacher inserted data). Notes that the teacher added in previous sections are listed in this section in front of him where he can open and see it with all the data that he added. For example, the name of the class that he mentioned before, and any content added in that assignment.

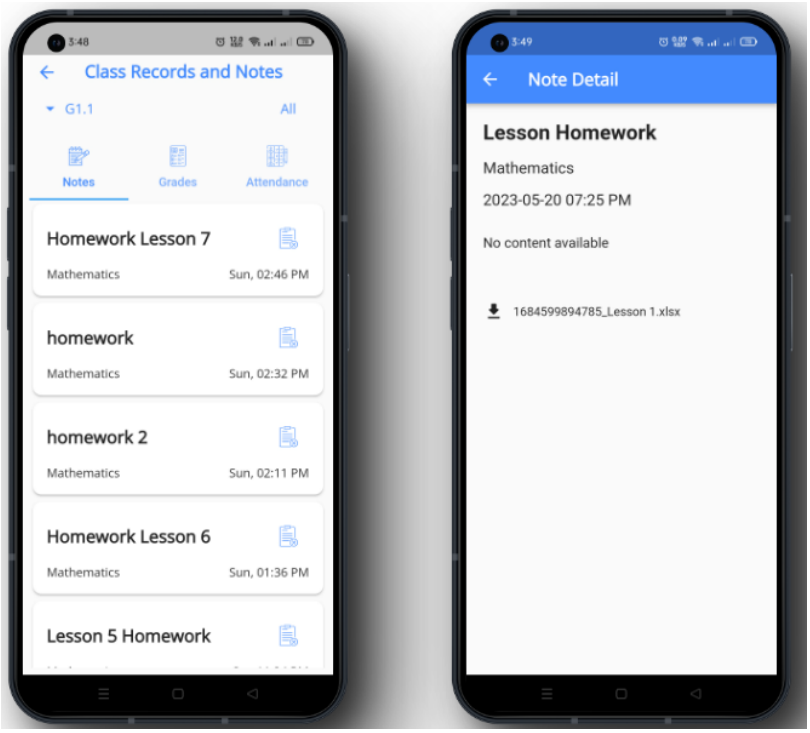


Figure 6.27

Attendance Records:

The attendance of students that was taken throw all days is now displayed as a list where the teacher can open each file and revise it.

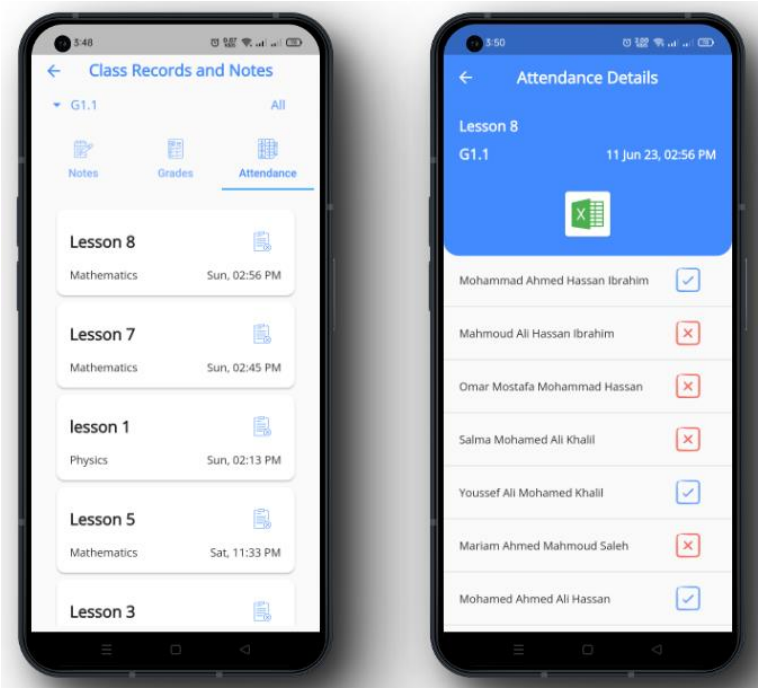


Figure 6.28

Saving Attendance:

Teachers can save their attendance file as excel sheets to make it well presented later and easy to collect information from. They can choose the location where they want to save their excel file.

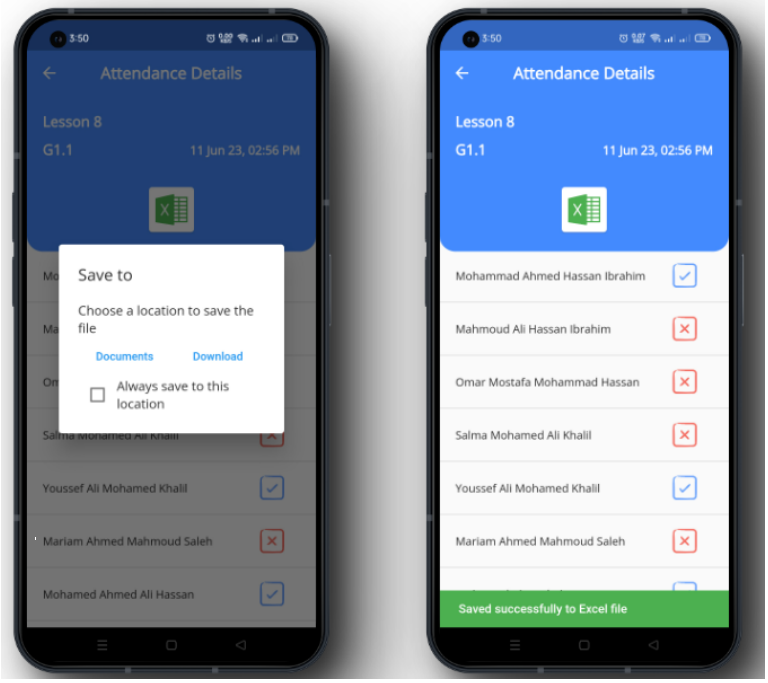


Figure 6.29

6.3 Allergens Research

After conducting a search on Google, we discovered that there are various types of allergies that can be triggered by different products. Some common types of allergies include:

Food Allergies:

Food allergies can be caused by specific ingredients or substances present in certain foods. Common food allergens include peanuts, tree nuts, dairy, eggs, wheat, soy.

Egg Allergy:

How to read a label for a peanut-free diet:

Albumin, Egg whites, Egg yolk, Dried egg, Egg powder, Egg solids, Egg substitutes, Eggnog, Fat substitutes made with egg, Globulin, Livetin, Lysozyme (used in Europe), Mayonnaise, Meringue, Ovalbumin, Ovomucin, Ovomucoid, Ovo vitellin, Surimi.

Other possible sources of eggs or egg products

Lecithin, Macaroni, Marzipan, Marshmallows, Nougat, Pasta, A shiny glaze or yellow baked good may mean egg, Fat substitutes made from either egg or milk protein, Egg whites and eggshells may be used as clarifying agents in soup stocks, consommés, bouillons, and coffees.

Peanut Allergy:

reading a label for a peanut-free diet:

Don't give your child foods that contain any of the following ingredients:

1.Arachis hypogea. This is the scientific name for the peanut plant.

2.Artificial nuts, Beer nuts, Cold-pressed, expressed, or expelled peanut oil, Goobers, Ground nuts, Mixed nuts, Monkey nuts, Nut pieces, Nutmeat, Peanut butter, Peanut flour, Peanut protein hydrolysate.

Foods that may contain peanut protein:

1. African, Chinese, Indonesian, Vietnamese, Thai, and other ethnic dishes

2. Baked goods, Candy, Cereals, Chili and spaghetti sauce, Chocolate in candy or candy bars, Crackers, Egg rolls, Enchilada sauce, Flavoring (natural and artificial), Hydrolyzed plant protein, Hydrolyzed vegetable protein, Ice cream and frozen yogurts, Marzipan, Mexican dishes, Mole sauce, Nougat.

Milk Allergy:

reading a label for a milk-free diet:

Don't have foods that contain any of the following ingredients:

Artificial butter flavor, Butter, butter fat, Buttermilk, Casein, Caseinates (ammonium, calcium, magnesium, potassium, sodium), Cheese, cottage cheese, curds, Cream, Custard, pudding, Ghee, Half-and-half (a blend of whole milk and light cream), Hydrolysates (casein, milk protein, protein, whey, whey protein), Lactalbumin, lactalbumin phosphate, Lactoglobulin, Lactose, Lactoferrin, Milk (derivative, protein, solids, malted, condensed, evaporated, dry, whole, low-fat, nonfat, skim), Nougat, Pudding, Rennet casein, Sour cream, Sour cream solids, Whey (delectated, demineralized, protein concentrate), Yogurt

Other possible sources of milk or milk products:

1. Brown sugar flavoring, Caramel candies, Caramel flavoring, Chocolate, High-protein flour, Lunch meats, hot dogs, sausages, Margarine, Natural flavoring, Milk-based fat substitutes

Wheat Allergy:

reading a label for a wheat-free diet:

Stay away from foods that contain any of the following ingredients:

Bran, Bread crumbs, Bulgur, Cereal extract, Couscous, Cracker meal, Durum, Durum flours (a type of wheat), Einkorn (a type of wheat), Emmer (a type of farro), Enriched flour, Farina, Flour (all purpose, bread, cake, instant, pastry, self-rising, soft wheat, steel ground, whole wheat), Gluten, Graham flour, High gluten flour, High protein flour, Hydrolyzed wheat protein, Matzoh, matzoh meal, Pasta, Seitan, Semolina (a type of wheat), Spelt (a type of wheat), Sprouted wheat ,Triticale (a type of wheat) ,Vital gluten ,Wheat bran ,Wheat durum ,Wheat germ, Wheat gluten, Wheat grass, Wheat malt, Wheat protein isolate, Wheat sprouts, Wheat starch, Whole wheat berries, Whole wheat flour.

Other possible sources of wheat or wheat products:

1. Ingredients that may mean the presence of wheat protein include the following:

2.Gelatinized starch, Glucose, Gum, Hydrolyzed vegetable protein, Kamet, Modified food starch, Modified starch, Natural flavoring, Oats, Soy sauce, Starch, Surimi, Textured vegetable protein, Vegetable starch

Soy Allergy:

reading a label for a soy-free diet:

For any FDA-regulated food, the word "soy" must appear somewhere on the label. This can be in the ingredient list. Or there may be a special allergen label such as "contains soy." Always read the entire ingredient label. Don't give your child foods with any of these ingredients:

Edamame, Hydrolyzed soy protein, Miso, Natto, Shoyu sauce, Shoyu, Soy albumin, Soy cheese, Soy fiber, Soy flour, Soy grits, Soy ice cream, Soy nuts, Soy milk, Soy sprouts, Soy protein concentrate, Soy protein isolate, Soy sauce, Soy sprouts, Soy yogurt, Soya, Soybean (curd, granules), Tamari, Tempeh, Textured vegetable protein (TVP), Tofu.

Foods that don't contain soy could be contaminated during manufacturing. Advisory statements are not regulated by the FDA. They are voluntary. These include labels such as "processed in a facility that also processed soy." Or "made on shared equipment." Ask your child's healthcare provider if your child can eat food with these labels. Or if your child should stay away from them.

Other possible sources of soy or soy products:

Asian food, Flavorings, Vitamin E, which contains soybean oil, Hydrolyzed plant protein, Hydrolyzed vegetable protein, Natural flavorings, Vegetable broth, Vegetable gum, Vegetable starch.

Handling the allergies in database:

To ensure effective allergy prevention and communication, the information regarding allergies is added to the database. This enables parents to take necessary precautions and helps the canteen worker identify problematic products for students. Here's how the process works:

1. **Parent's Role:**
    - Parents provide accurate and up-to-date information about their child's allergies.

- They input this information into the database, including specific allergens and associated reactions.

2. Database Management:
    - The database securely stores the allergy information for each student.
    - It maintains a comprehensive record of allergens that can cause problems for individual students.
3. Allergy Notification:
    - When a student attempts to purchase a product from the canteen, the database is accessed.
    - The database compares the product's ingredients with the student's allergy information.
4. Canteen Worker's Alert:
    - If the product contains allergens that can cause problems for the student, the canteen worker receives a notification.
    - The worker is immediately made aware that the product should not be sold to the student due to potential allergic reactions.

By integrating allergy information into the database, parents are empowered to protect their children from allergens, while the canteen worker is equipped to make informed decisions regarding product availability. This collaborative approach ensures that students can make safe choices when purchasing food or other products from the canteen, reducing the risk of allergic reactions.

Handling the allergies in parent application:

These screens aim to provide guidance on how parents can effectively handle allergies affecting their children. By following the steps outlined below, parents can ensure the well-being of their son or daughter and keep them safe from potential allergens.

1. Initial Setup:
    - Parents need to register their child's allergies in the database system.
    - The database should include specific details about the allergens and the corresponding symptoms or reactions they cause.
2. Parent's Role:

- Parents must carefully select the appropriate allergies that their child has from the provided options.
- Once the allergies are chosen, any related problems or concerns regarding the child's health should be promptly updated in the database.

3. **Canteen Worker's Role:**
    - The database system automatically notifies the canteen worker about the allergies and associated reactions that the child experiences.
    - When a parent updates the database with new information, the canteen worker is immediately alerted.
4. **Product Availability:**
    - Based on the updated database, the canteen worker can identify which products might cause problems for the child due to their allergies.
    - The worker can then ensure that those specific products are not available for purchase by the child.

By implementing this system, parents can effectively communicate their child's allergies to the canteen staff, enabling them to provide a safe and allergen-free environment for the child. This process emphasizes the importance of timely updates and information sharing to mitigate any potential health risks or allergic reactions.

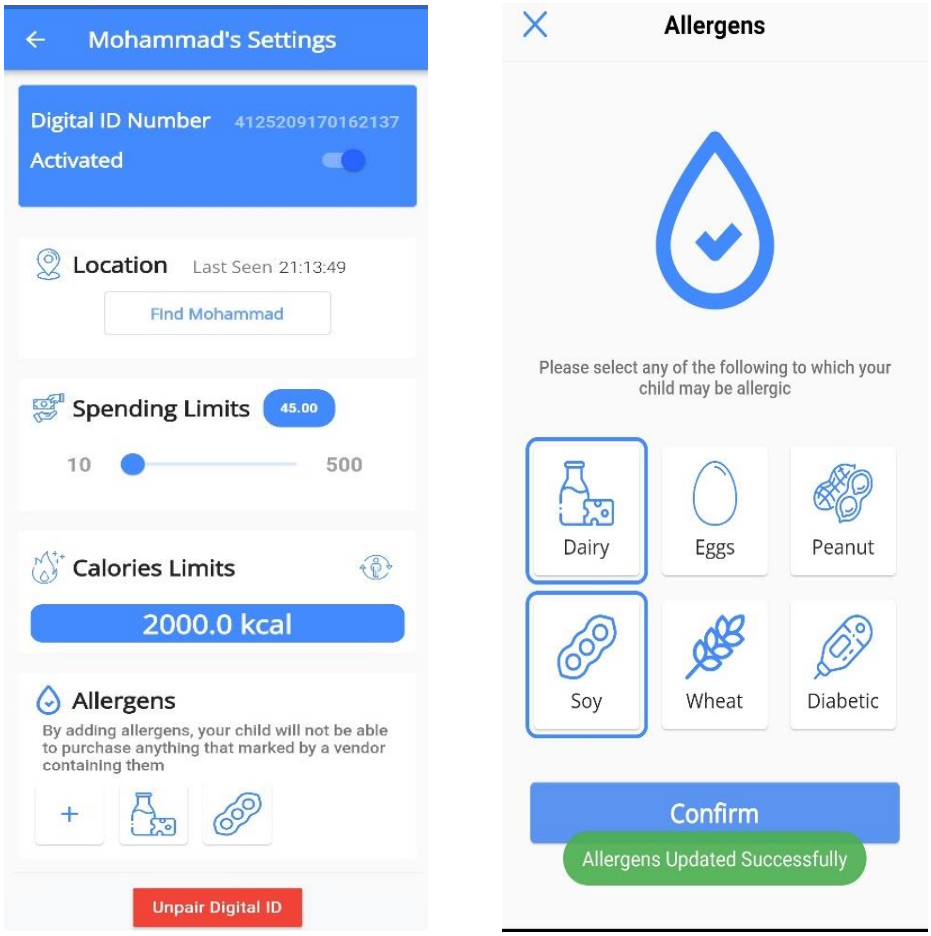


Figure 6.30

Handling the allergies in canteen application:

These screens elaborate on how the canteen worker can effectively handle allergies when adding products to the menu. By following the outlined steps, the worker can ensure the safety of students with known allergies.

Product Entry:

When adding a new product to the canteen menu, the canteen worker must include the ingredients used to make the product.

The worker needs to input this ingredient information accurately into the database.

Allergen Identification:

The database system analyzes the ingredients provided by the canteen worker. It identifies any allergens that may be present in the product based on the known allergens associated with the ingredients.

Alerting the Worker:

If the product contains allergens that may cause problems for a student, the database system notifies the canteen worker. The worker receives an immediate alert, indicating that the product shouldn't be sold to a particular student with relevant allergies.

Preventing Transactions:

When a student attempts to purchase a product, the canteen worker scans the database for the student's allergy information. If the product contains allergens that can potentially harm the student, the transaction is declined or prevented.

This approach ensures that the canteen worker remains vigilant in providing a safe environment for students with allergies. By accurately inputting ingredient information and relying on the database's analysis, potential allergic reactions can be avoided. This process emphasizes the importance of thorough ingredient labeling and effective communication between the canteen worker, the database system, and the students' allergy information.

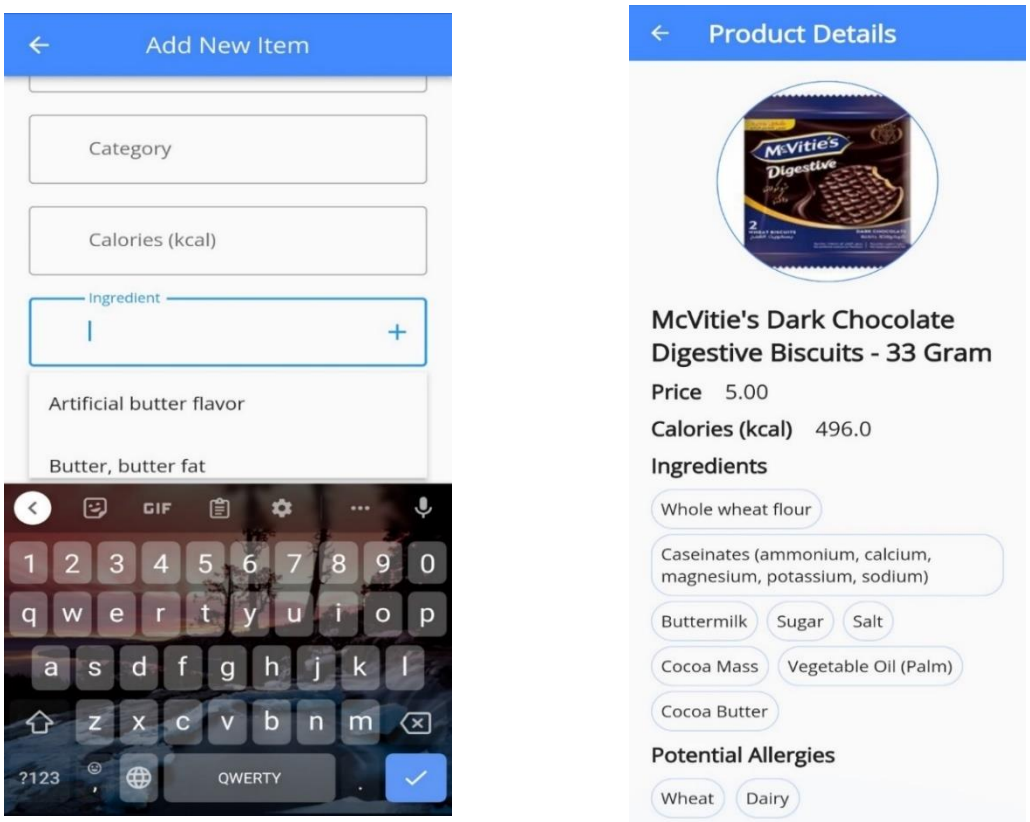


Figure 6.31

These screens demonstrate the scenario where a student attempts to purchase a product from the canteen but encounters a failed transaction due to allergies. The following description explains the process in detail.

Student Selection:

The student chooses a product from the available options in the canteen.

The selection triggers the transaction process.

Allergy Check:

The canteen system checks the student's allergy information stored in the database.

It retrieves the specific allergens associated with the student.

Product Analysis:

The canteen system analyzes the ingredients of the selected product.

It matches the product's ingredients with the student's known allergens.

Transaction Validation:

If any of the product's ingredients contain allergens that could cause an allergic reaction in the student, the transaction fails.

The canteen system displays an error message indicating that the product cannot be purchased due to potential allergies.

Alternatives or Notifications:

Upon a failed transaction, the canteen system may suggest an alternative.

products that do not contain allergens harmful to the student.

Additionally, if the canteen worker is present, they can be alerted about the failed transaction and assist the student in finding a suitable alternative.

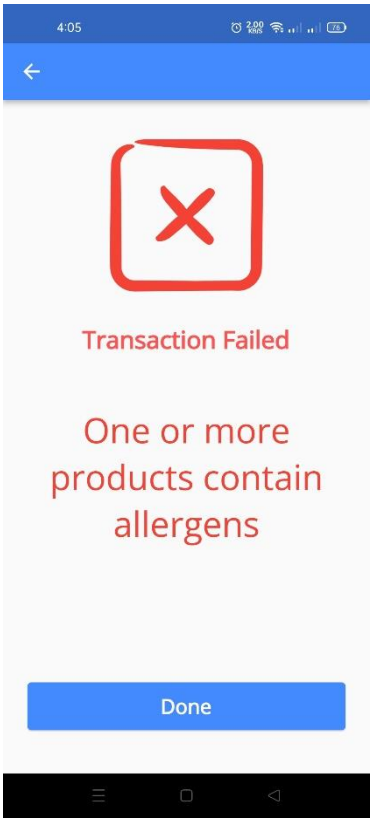


Figure 6.32

By implementing this process, the canteen aims to prioritize student safety by preventing the purchase of products that could potentially cause allergic reactions. It ensures that students with allergies are protected and guided towards safer options within the canteen's offerings.

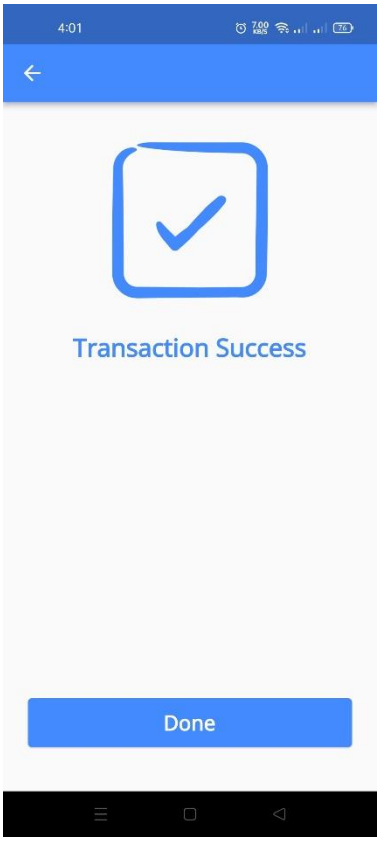


Figure 6. 33

6.4 CANTEEN WORKER INTERFACE

Canteen overview:

This section discusses the role of canteen workers in our app and how they interact with the student's purchases and manages the items that they buy with the help of a calculated calories system and allergies constraints. Parents control what their children eat and charge their balance with the amount of money they think is enough. They also state their children's allergies to help them stay away from food that may be bad for them, they also control the maximum limit of calories that each student can take daily. This helps parents take control of what their children eat.

Joining Our Community:

Our app gives canteen the ability to join our community and choose between variety of options such as the country that they are in and the wide variety of schools that are maintained by our application.

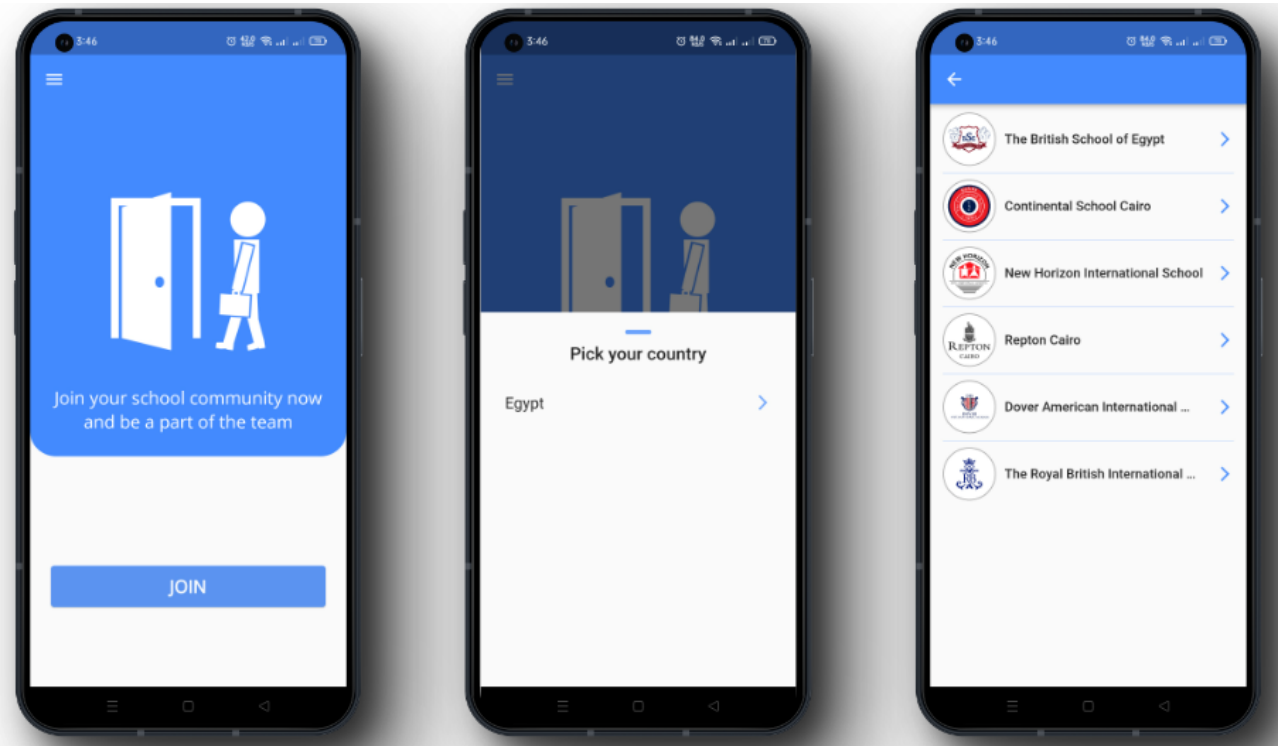


Figure 6.34

Main Screen:

The main screen (as shown in fig 70) is the first interactive screen that the canteen worker deals with. It shows the daily sales report for how many orders have been done on the system, it also shows the total cash that students cashed in. The main screen consists of two main parts (Menu and Inventory) which we will discuss further.

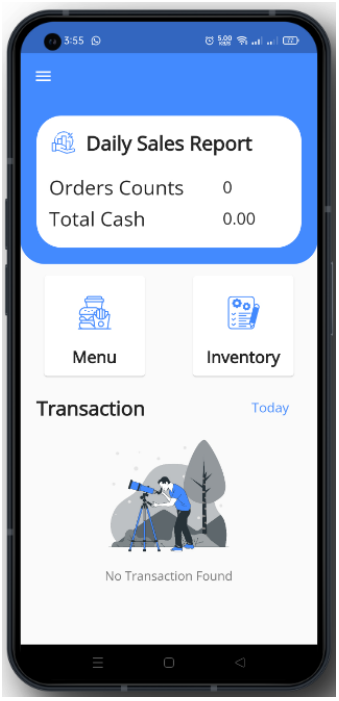


Figure 6.35

Main Screen & Date Ordering:

(As shown in fig 71) Main screen can be ordered by date, meaning that the canteen worker can adjust the app with the date that he wants to revise any transactions and order counts. The canteen worker can tap on any transaction to see its details and revise what was purchased.

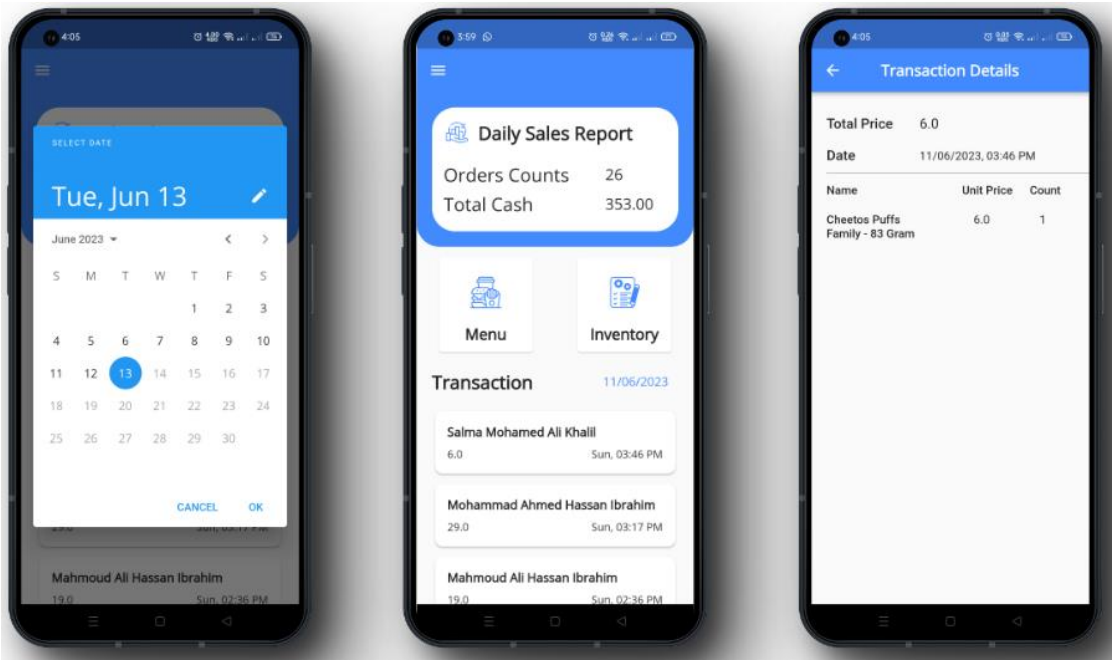


Figure 6.36

MENU:

The menu screen (as shown in fig 72) contains a list of a wide variety of items such as biscuits, beverages, chips, dips and snacks that students can buy. The canteen worker can press on the cart button to add an item to the cart list.

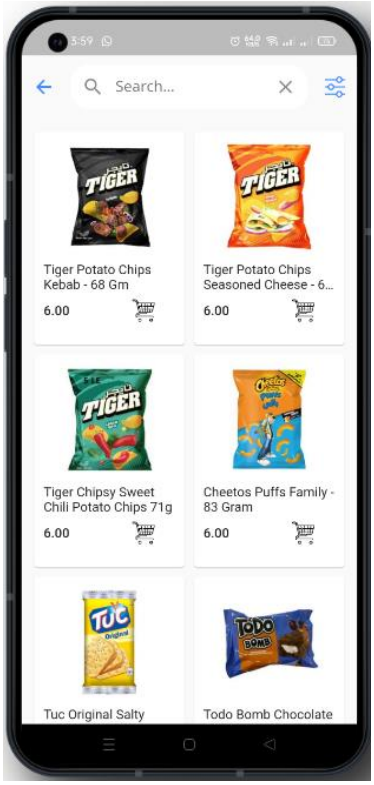


Figure 6.37

product details:

This part (as shown in fig 73) contains details of every item in the shop including the price, Calories, ingredients and the type of allergies that this product may cause.

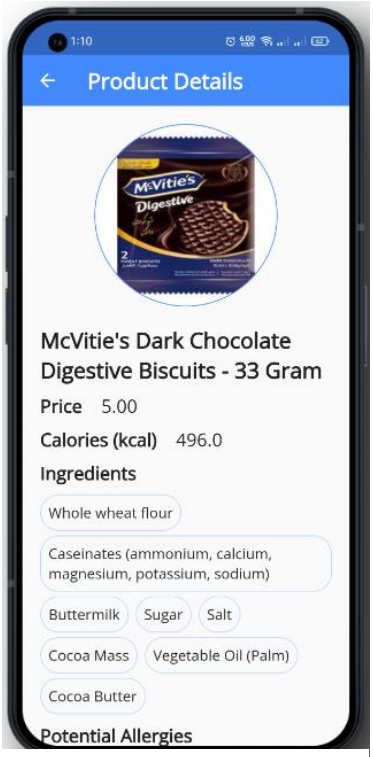


Figure 6.38

Menu Categories Sorting:

The menu screen (as shown in fig) can be sorted by the category of the item into 3 types (biscuits & beverages & chips, dips and snacks), this sorting helps the canteen worker make it easier to find the items he is looking for. After adding the items to the cart, he presses the proceed button to go to checkout.

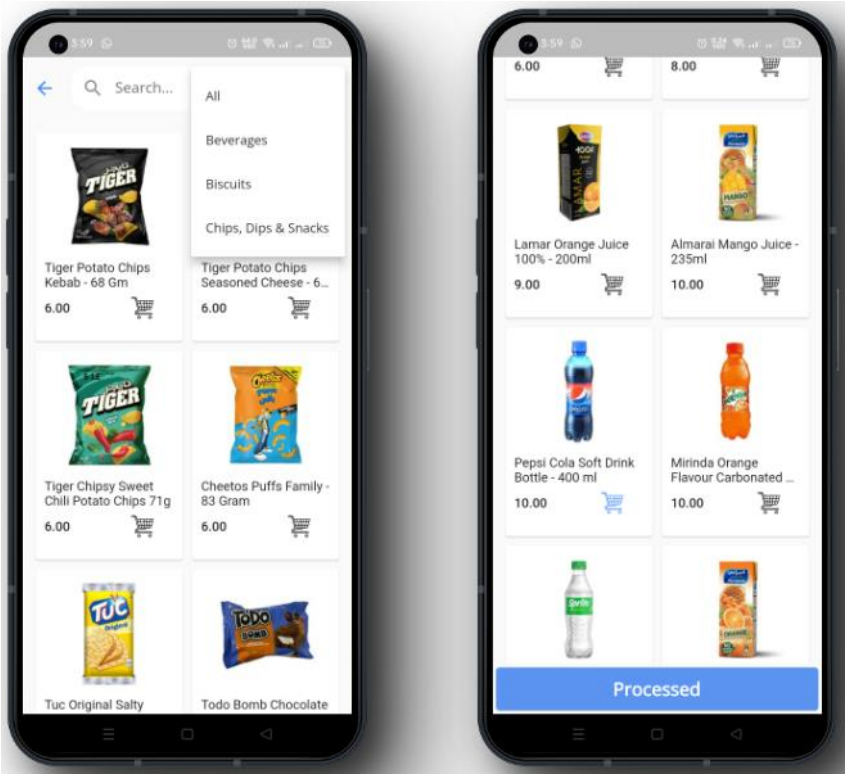


Figure 6.39

Menu checkout:

In this section the canteen worker revises the order that he made, He can edit, add or delete any unwanted item (as shown in fig). There is an item count at the top of the screen with the amount of money that the student must buy. After that, he confirms the order and now it's time for the student to pass his wristband on to the NFC reader to confirm the order.

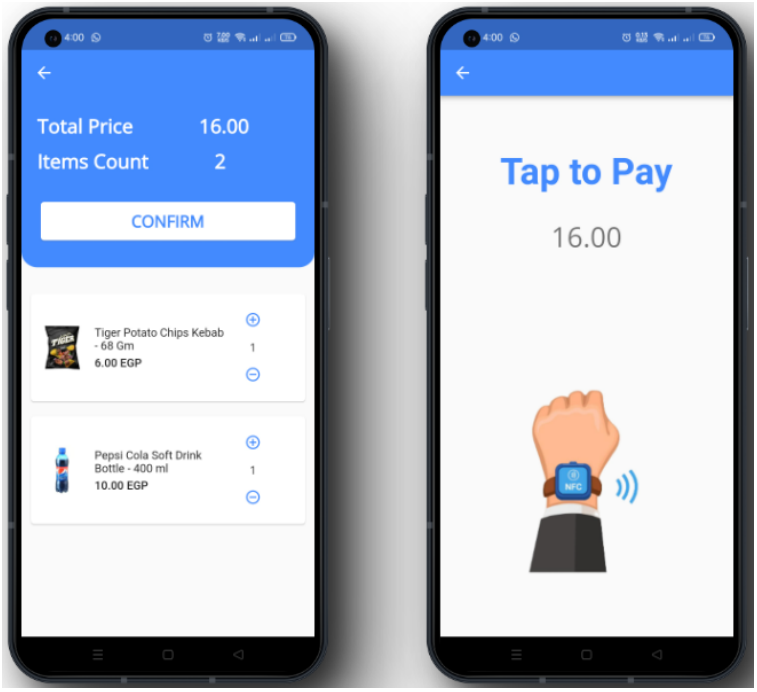


Figure 6.40

Transaction success:

After passing the wristband on to the NFC reader there is a chance that the transaction may be completed or denied. In this case the transaction was accepted (as shown in fig 76) as the balance was proper with the items that the students bought, and there were no allergies that conflict with the students purchased items and there was no exceeding in the calories limit.

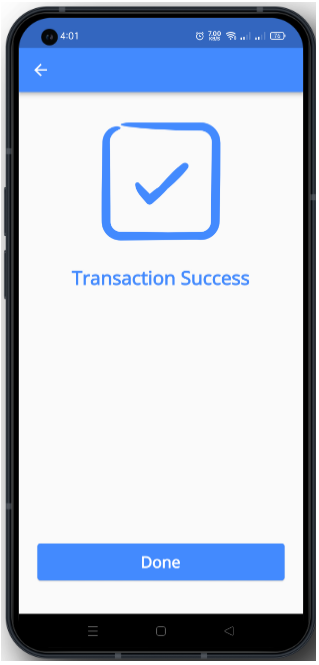


Figure 6.41

Transaction failed:

In these screens there is a problem with transaction processing (as shown in fig 77), this is maybe several problems. First, the items exceeded the calorie limit stated by the student's parent. Second, the student exceeded the daily spending limit stated by their parents. Finally, some items may contain allergens that may cause allergies for those students.

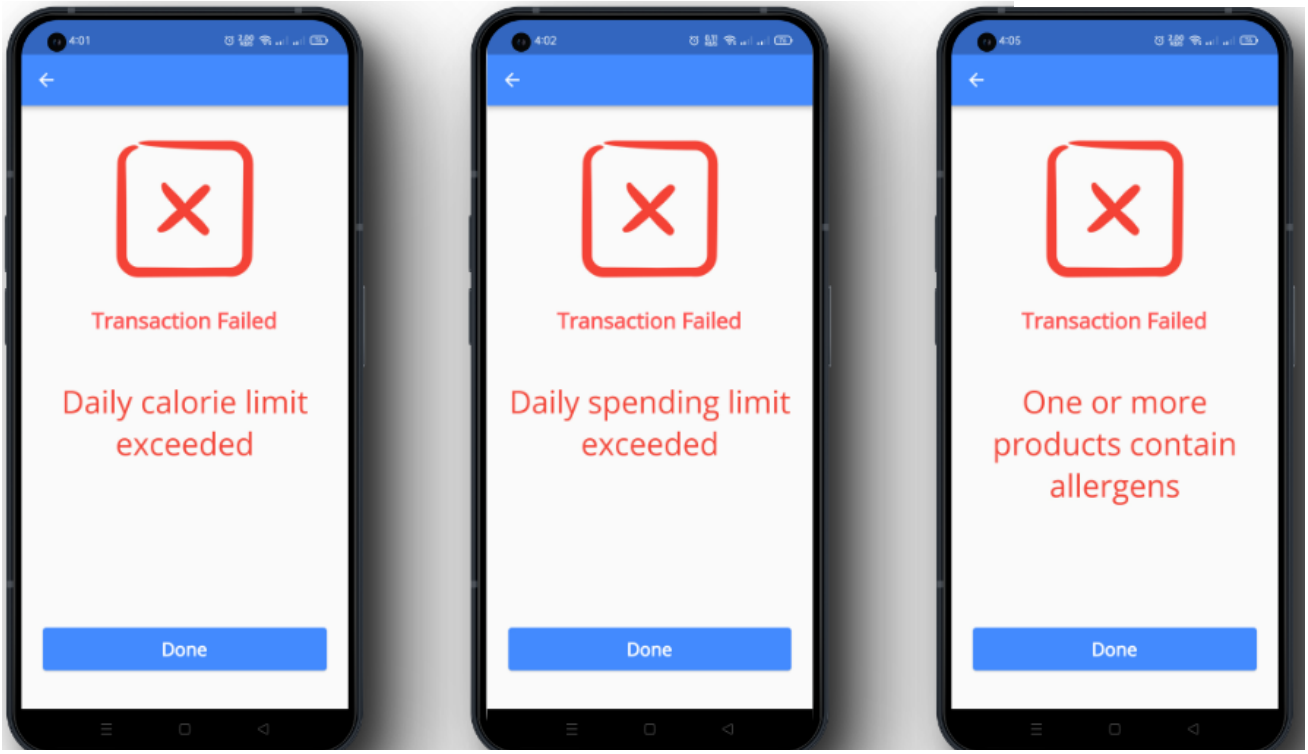


Figure 6.42

Inventory:

The inventory section (as shown in fig 78) contains all the items that are in the shop. The canteen worker can add or delete any item in addition to editing the prices of them, He can also add new categories or delete them.

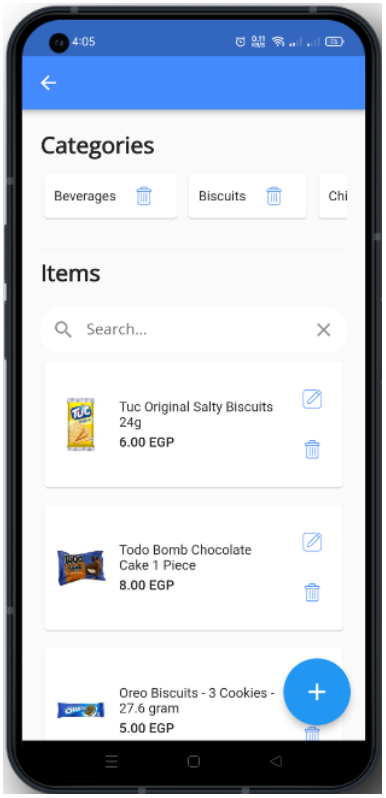


Figure 6.43

Add New Item:

By pressing on the plus button, the canteen worker is directed to the new item adding screen (As shown in fig 79) where he can write all the details of the items such as the name of the item, the price, the category from the list of categories we made, the calories that this item contains and the ingredient of this item where allergies are caused from.

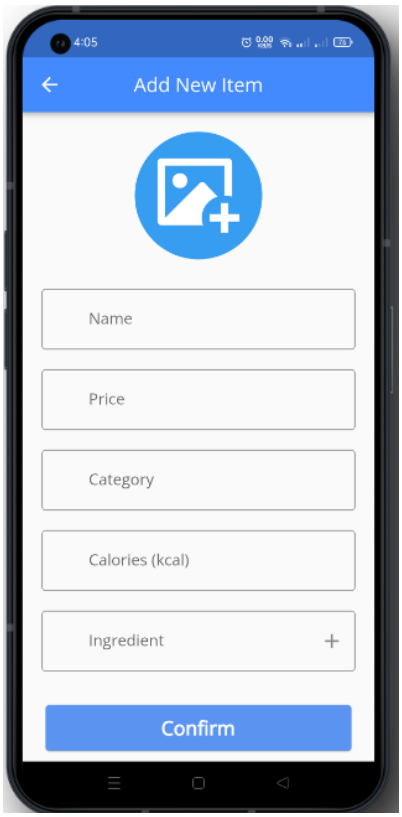


Figure 6.44

Deleting, Editing Items:

(As shown in fig 80) the canteen worker can delete a category by pressing on the trash button beside categories or he can delete an item by also pressing the trash button nut beside items, He can also edit the price of any item by pressing on the pen tool beside them.

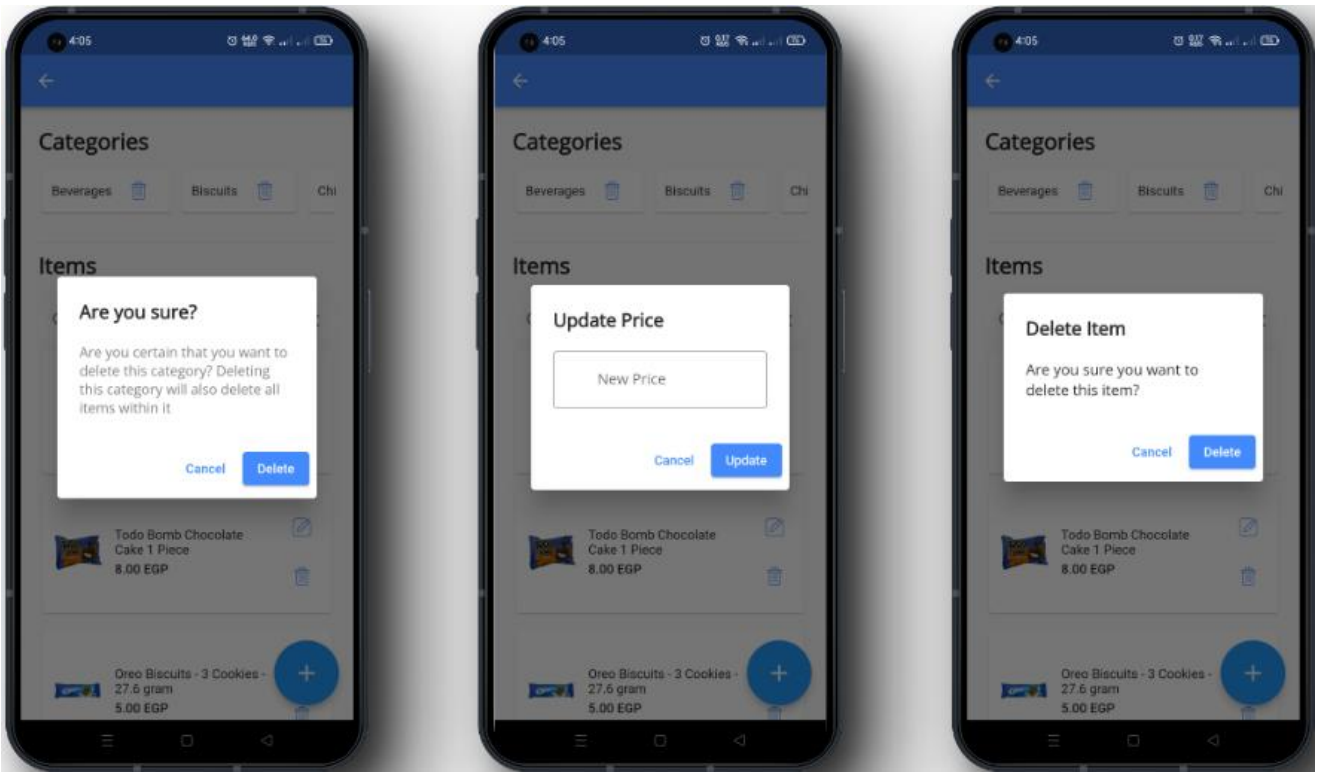


Figure 6.45

7. Appendices

7.1 Microcontroller Code

7.1.1 Bracelet Device Code

```
//Select your modem
//SSL/TLS is currently supported only with SIM8xx series
#define TINY_GSM_MODEM_SIM800
//Increase RX buffer
#define TINY_GSM_RX_BUFFER 256
#include <SoftwareSerial.h>
#include <TinyGPS++.h> //https://github.com/mikalhart/TinyGPSPlus
#include <TinyGsmClient.h> //https://github.com/vshymanskyy/TinyGSM
#include <ArduinoHttpClient.h> //https://github.com/arduino-
libraries/ArduinoHttpClient
#include <ArduinoJson.h>

const char FIREBASE_HOST[]  = "firestore.googleapis.com";
const String FIREBASE_AUTH  = "AIzaSyAzxeW0A5HnyOQ_gnLI0wVySnVm30RTjpc";
const String FIREBASE_PATH = "/v1/projects/smartschool-
6aee1/databases/(default)/documents/Countries/RBH9GduunnyuyMqZ3kcz/Schools/EsiA0KfrRPB
TJM2WBd87/Students/4125209170162137/Location/location";
const int SSL_PORT  = 443;

// kThe serial connection to the GPS device
String str_date = "";
String str_time = "";
double latitude , longitude;
String lat_str , lng_str;

char apn[]  = "internet.vodafone.net";
char user[] = "";
char pass[] = "";

#define rxPin 4
#define txPin 2
HardwareSerial sim800(1);
TinyGsm modem(sim800);

#define RXD2 18
#define TXD2 19
HardwareSerial neogps(2);
TinyGPSPlus gps;
```

```
TinyGsmClientSecure gsm_client_secure_modem(modem, 0);
HttpClient http_client = HttpClient(gsm_client_secure_modem, FIREBASE_HOST, SSL_PORT);

unsigned long previousMillis = 0;
long interval = 10000;

void setup() {
  Serial.begin(115200);
  Serial.println("esp32 serial initialize");

  sim800.begin(9600, SERIAL_8N1, rxPin, txPin);
  Serial.println("SIM800L serial initialize");

 // neogps.begin(9600, SERIAL_8N1, RXD2, TXD2);
  //Serial.println("neogps serial initialize");
  neogps.begin(9600, SERIAL_8N1, RXD2, TXD2);
  Serial.println("neogps serial initialize");
  delay(3000);

  Serial.println("Initializing modem...");
  modem.restart();
  String modemInfo = modem.getModemInfo();
  Serial.print("Modem: ");
  Serial.println(modemInfo);

  http_client.setHttpResponseTimeout(90 * 1000); //^0 secs timeout
}

void loop() {
  Serial.print(F("Connecting to "));
  Serial.print(apn);
  if (!modem.gprsConnect(apn, user, pass)) {
    Serial.println(" fail");
    delay(1000);
    return;
  }
  Serial.println(" OK");

  http_client.connect(FIREBASE_HOST, SSL_PORT);

  while (true) {
    if (!http_client.connected()) {
      Serial.println();
      http_client.stop();// Shutdown
```

```
      Serial.println("HTTP not connect");
      break;
    }
    else{
      gps_loop();
    }
  }
}

void PostToFirebase(const char* method, const String & path, double latitude, double
longitude, const String & time, const String & date, HttpClient* http) {
  String response;
  int statusCode = 0;
  http->connectionKeepAlive(); // Currently, this is needed for HTTPS

  String url = path + "?key=" + FIREBASE_AUTH;
  Serial.print(method);
  Serial.print(":");
  Serial.println(url);
    // Create a StaticJsonDocument with enough capacity for the JSON object
    StaticJsonDocument<256> jsonDoc;

    // Set the fields in the JSON object
    jsonDoc["fields"]["latitude"]["doubleValue"] = latitude;
    jsonDoc["fields"]["longitude"]["doubleValue"] = longitude;
    jsonDoc["fields"]["date"]["stringValue"] = date.c_str();
    jsonDoc["fields"]["time"]["stringValue"] = time.c_str();

    // Convert the JSON object to a String
    String data;
    serializeJson(jsonDoc, data);

  // Create a JSON object with the GPS data
  Serial.print("Data:");
  Serial.println(data);

  String contentType = "application/json";
  http->beginRequest();

  // Use the method parameter to determine the type of request
  if (strcmp(method, "PATCH") == 0) {
    http->patch(url, contentType, data);
  } else if (strcmp(method, "POST") == 0) {
    http->post(url, contentType, data);
```

```
  } else if (strcmp(method, "PUT") == 0) {
    http->put(url, contentType, data);
  } else {
    Serial.println("Invalid method specified.");
    return;
  }

  statusCode = http->responseStatusCode();
  Serial.print("Status code: ");
  Serial.println(statusCode);
  response = http->responseBody();
  Serial.print("Response: ");
  Serial.println(response);

  if (!http->connected()) {
    Serial.println();
    http->stop(); // Shutdown
    Serial.println("HTTP disconnected");
  }
}


void gps_loop() {
  latitude = gps.location.lat();
  longitude = gps.location.lng();
  printFloat(gps.location.lat(), gps.location.isValid(), 11, 6);
  printFloat(gps.location.lng(), gps.location.isValid(), 12, 6);

  str_date = printDate(gps.date);
  Serial.print(' ');
  str_time = printTime(gps.time);
  lat_str = String(latitude , 14);
  lng_str = String(longitude , 14);
  Serial.println();

    if (gps.location.isValid())
   {
     //String jsonString = "{\"fields\":{\"to\":{\"stringValue\":\"ELOMDA\"}}}";
      PostToFirebase("PATCH", FIREBASE_PATH, latitude, longitude, str_time, str_date,
&http_client);
     //network.FirestoreDataUpdate(str_date, str_time, latitude, longitude);
     delay(10000);

   }
```

```
  smartDelay(1000);

  if (millis() > 5000 && gps.charsProcessed() < 10)
    Serial.println(F("No GPS data received: check wiring"));
}

// This custom version of delay() ensures that the gps object
// is being "fed".
static void smartDelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (neogps.available())
      gps.encode(neogps.read());
  } while (millis() - start < ms);
}

static void printFloat(float val, bool valid, int len, int prec)
{
  if (!valid)
  {
    while (len-- > 1)
      Serial.print('*');
    Serial.print(' ');
  }
  else
  {
    Serial.print(val, prec);
    int vi = abs((int)val);
    int flen = prec + (val < 0.0 ? 2 : 1); // . and -
    flen += vi >= 1000 ? 4 : vi >= 100 ? 3 : vi >= 10 ? 2 : 1;
    for (int i=flen; i<len; ++i)
      Serial.print(' ');
  }
  smartDelay(0);
}

static void printInt(unsigned long val, bool valid, int len)
{
  char sz[32] = "*****************";
  if (valid)
    sprintf(sz, "%ld", val);
```

```
  sz[len] = 0;
  for (int i=strlen(sz); i<len; ++i)
    sz[i] = ' ';
  if (len > 0)
    sz[len-1] = ' ';
  Serial.print(sz);
  smartDelay(0);
}
String printDate(TinyGPSDate &d)
{
  if (!d.isValid())
  {
    Serial.print(F("********** "));
  }
  else
  {
    char sz[32];
    sprintf(sz, "%02d/%02d/%02d", d.month(), d.day(), d.year());
    Serial.print(sz);
    return sz;
  }
}
String printTime(TinyGPSTime &t)
{
  if (!t.isValid())
  {
    Serial.print(F("******** "));
  }
  else
  {
    char sz[32];
    sprintf(sz, "%02d:%02d:%02d", t.hour() + 2, t.minute(), t.second());
    Serial.print(sz);
    return sz;
  }
}
static void printStr(const char *str, int len)
{
  int slen = strlen(str);
  for (int i=0; i<len; ++i)
    Serial.print(i<slen ? str[i] : ' ');
  smartDelay(0);
}
```

7.1.2 Canteen Device Main Code

```
#ifndef ESP32
#define ESP32
#endif

#include "PN53.h"
#include "Network.h"

String studentUID = "";

Network network;

void setup()
{

  Serial.begin(115200);
  Serial.println("Hello!");

  network.initWiFi();
  network.initFirebase();

  Set_Up_PN53();
  pinMode(BUZZER, OUTPUT);
}

void loop()
{
   studentUID = readNFC();
   if(studentUID)
   {
     network.FirestoreDataUpdate(studentUID);
   }
}
```

7.1.3 Gate Device Main Code

```
#ifndef ESP32
#define ESP32
#endif

#include "PN53.h"
#include "Network.h"
const int redled = 27;
const int button = 12;

String studentUID = "";
String actionType = "";


/*********************************************************************/
/* Project how to read PN532 nfc reader from an ESP32 using multi-      */
/* tasking.                                                            */
/* This project depends on the <Adafruit_PN532.h>, <SPI.h> libraries  */
/*                                                                    */
/*********************************************************************/

Network network;

void setup()
{

  Serial.begin(115200);
  Serial.println("Hello!");

  network.initWiFi();
  network.initFirebase();

  Set_Up_PN53();
  pinMode(redled, OUTPUT);
  pinMode(button, INPUT_PULLUP);
  pinMode(BUZZER, OUTPUT);

  attachInterrupt(digitalPinToInterrupt(button), changeMode, CHANGE);

  //default state
  digitalWrite(redled, LOW);
  actionType = "Arrived";
```

```
}

void loop()
{

  // default action
  // you will use the button here to update the action type

   studentUID = readNFC();
   if(studentUID)
   {
    network.FirestoreDataUpdate(studentUID, actionType);

   }

}

void changeMode()
{
      delayMicroseconds(200000);
      digitalWrite(redled,!digitalRead(redled));

      if(digitalRead(redled) == HIGH)
        {
          actionType = "Left";
        }else
        {
          actionType = "Arrived";
        }
        Serial.println(actionType);
}
```

7.2 Mobile Application Code

7.2.1 Main Code

```
import 'package:connectivity_plus/connectivity_plus.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_messaging/firebase_messaging.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:stguard/layout/canteen/cubit/cubit.dart';
import 'package:stguard/layout/parent/cubit/cubit.dart';
import 'package:stguard/layout/teacher/cubit/cubit.dart';
import 'package:stguard/modules/login/login_screen.dart';
import 'package:stguard/shared/bloc_observer.dart';
import 'package:stguard/shared/components/constants.dart';
import 'package:stguard/shared/internet_cubit/cubit.dart';
import 'package:stguard/shared/main_cubit/main_cubit.dart';
import 'package:stguard/shared/network/local/cache_helper.dart';
import 'package:stguard/shared/styles/themes.dart';

Future<void> _firebaseMessagingBackgroundHandler(RemoteMessage message) async {
  print(message.data.toString());
  print('onBackgroundMessage');
}

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();

  Bloc.observer = MyBlocObserver();
  await CacheHelper.init();

  userID = CacheHelper.getData(key: 'id');
  userRole = CacheHelper.getData(key: 'role');
  Widget startScreen = LoginScreen();
  if (userID != null) {
    await FirebaseAuth.instance.currentUser!.reload();
    emailVerified = FirebaseAuth.instance.currentUser!.emailVerified;
    await FirebaseMessaging.instance.getToken();

    FirebaseMessaging.onMessageOpenedApp.listen((event) {
      print(event.data.toString());
      print('onMessageOpenedApp');
```

```
    });

    FirebaseMessaging.onBackgroundMessage(_firebaseMessagingBackgroundHandler);

    startScreen = homeScreens[userRole]!;
  }

  runApp(BlocProvider(
    create: (context) => InternetCubit(Connectivity()),
    child: MyApp(
      startScreen: startScreen,
    ),
  ));
}

class MyApp extends StatelessWidget {
  Widget startScreen;
  MyApp({super.key, required this.startScreen});
  @override
  Widget build(BuildContext context) {
    return MultiBlocProvider(
      providers: [
        BlocProvider(create: (context) => MainCubit()),
        BlocProvider(
          create: (context) => ParentCubit(),
        ),
        BlocProvider(
          create: (context) => TeacherCubit(),
        ),
        BlocProvider(create: (context) => CanteenCubit()),
      ],
      child: MaterialApp(
        home: startScreen,
        debugShowCheckedModeBanner: false,
        theme: ThemeData(
            appBarTheme: const AppBarTheme(color: defaultColor),
            primaryColor: defaultColor,
            textTheme: TextTheme(
                titleLarge: Theme.of(context)
                    .textTheme
                    .titleLarge!
                    .copyWith(fontFamily: 'OpenSans'),
                titleMedium: Theme.of(context)
                    .textTheme
```

```
                    .titleMedium!
                    .copyWith(fontFamily: 'OpenSans'),
                headlineSmall: Theme.of(context)
                    .textTheme
                    .headlineSmall!
                    .copyWith(
                        fontFamily: 'OpenSans', fontWeight: FontWeight.w700),
                headlineMedium: Theme.of(context)
                    .textTheme
                    .headlineMedium!
                    .copyWith(fontFamily: 'OpenSans'),
                bodyLarge: Theme.of(context)
                    .textTheme
                    .bodyLarge!
                    .copyWith(fontFamily: 'OpenSans'),
                bodySmall: Theme.of(context)
                    .textTheme
                    .bodySmall!
                    .copyWith(fontFamily: 'OpenSans', fontSize: 15))),
      ),
    );
  }
}
```

7.2.2 Parent Home Screen Code

```
import 'package:conditional_builder_null_safety/conditional_builder_null_safety.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:stguard/layout/parent/cubit/cubit.dart';
import 'package:stguard/layout/parent/cubit/states.dart';
import 'package:stguard/modules/login/login_screen.dart';
import 'package:stguard/modules/parent/credit_card/credit_card_screen.dart';
import 'package:stguard/modules/parent/notifications/notifications_screen.dart';
import 'package:stguard/modules/parent/pick_school/pick_school_screen.dart';
import 'package:stguard/shared/components/components.dart';
import 'package:stguard/shared/components/constants.dart';
import 'package:stguard/shared/internet_cubit/cubit.dart';
import 'package:stguard/shared/main_cubit/main_cubit.dart';
import 'package:stguard/shared/main_cubit/states.dart';
import 'package:stguard/shared/styles/themes.dart';

class ParentHomeScreen extends StatelessWidget {
  ParentHomeScreen({super.key});
  @override
  Widget build(BuildContext context) {
    ParentCubit.get(context)
      ..createDatabase()
      ..getMyStudents()
      ..getParentInfo();
    return BlocListener<InternetCubit, ConnectionStatus>(
      listener: (context, state) {
        if (state == ConnectionStatus.disconnected) {
          showSnackBar(context,
              message: 'You are currently offline.', icon: Icons.wifi_off);
        } else if (state == ConnectionStatus.connected) {
          showSnackBar(context,
              message: 'Your internet connection has been restored.',
              icon: Icons.wifi);
        }
      },
      child: BlocConsumer<ParentCubit, ParentStates>(
        listener: (context1, state) async {
          if (state is GetCountriesSucessState) {
            showModalBottomSheet(
              context: context,
              builder: (context) => Padding(
                padding:
```

```
                const EdgeInsets.symmetric(horizontal: 20, vertical: 10),
            child: Column(
              children: [
                Expanded(
                  child: Column(
                    children: [
                      Container(
                        width: 30,
                        height: 5,
                        margin: const EdgeInsets.only(top: 10),
                        decoration: BoxDecoration(
                            color: defaultColor.withOpacity(0.8),
                            borderRadius: const BorderRadius.all(
                                Radius.circular(10))),
                      ),
                      const SizedBox(
                        height: 10,
                      ),
                      const Text(
                        'Pick your country',
                        style: TextStyle(
                            fontSize: 20, fontWeight: FontWeight.w500),
                      ),
                      const SizedBox(
                        height: 15,
                      ),
                    ],
                  ),
                ),
                Expanded(
                  flex: 5,
                  child: ListView.separated(
                      shrinkWrap: true,
                      itemBuilder: (context, index) => CountryItem(
                            country: ParentCubit.get(context)
                                .countries[index]
                                .name,
                            onTap: () =>
                                ParentCubit.get(context).pickCountry(index),
                          ),
                      separatorBuilder: (context, index) => const Divider(),
                      itemCount: ParentCubit.get(context).countries.length),
                ),
              ],
```

```
            ),
          ),
        );
      } else if (state is GetSchoolsSucessState) {
        navigateTo(context, const PickSchoolScreen());
      } else if (state is PickCountryState) {
        await ParentCubit.get(context).getSchools();
      } else if (state is UserSignOutSuccessState) {
        navigateAndFinish(context, LoginScreen());
      }
    },
    builder: (context, state) {
      return Scaffold(
          appBar: AppBar(
            elevation: 0.0,
            actions: [
              IconButton(
                  onPressed: () {
                    ParentCubit.get(context).getNotifications();
                    navigateTo(context, NotificationsScreen());
                    ParentCubit.get(context).resetNotificationCount();
                  },
                  icon: Badge(
                      isLabelVisible:
                          ParentCubit.get(context).notificationCount == null
                              ? false
                              : true,
                      label: ParentCubit.get(context).notificationCount ==
                              null
                          ? null
                          : Text(
                              '${ParentCubit.get(context).notificationCount}'),
                      child: const ImageIcon(
                          AssetImage('assets/images/notification.png'))))
            ],
          ),
          drawer: Drawer(
              child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                Expanded(
                  flex: 1,
                  child: DrawerHeader(
                    padding:
```

```
                    const EdgeInsets.only(top: 20, left: 20, right: 20),
                child: ParentCubit.get(context).parent != null
                    ? UserInfo(
                        userModel: ParentCubit.get(context).parent!,
                      )
                    : const Center(
                        child: CircularProgressIndicator(),
                      ),
              ),
            ),
            Expanded(
                flex: 2,
                child: Padding(
                  padding: const EdgeInsets.symmetric(
                      horizontal: 20, vertical: 10),
                  child: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      Text('Settings',
                          style: Theme.of(context)
                              .textTheme
                              .headlineMedium!
                              .copyWith(color: defaultColor)),
                      const SizedBox(
                        height: 20,
                      ),
                      DrawerItem(
                        text: 'Add Family Member',
                        icon: const Image(
                          color: defaultColor,
                          image: AssetImage('assets/images/member.png'),
                          width: 40,
                          height: 40,
                        ),
                        ontap: () async =>
                            await ParentCubit.get(context)
                                .getCountries(),
                      ),
                      const SizedBox(
                        height: 15,
                      ),
                      DrawerItem(
                        text: 'Recharge',
                        icon: const Image(
```

```
                    color: defaultColor,
                    image:
                        AssetImage('assets/images/recharge.png'),
                    width: 40,
                    height: 40,
                  ),
                  ontap: () =>
                      navigateTo(context, CreditCardScreen()),
                ),
                const SizedBox(
                  height: 15,
                ),
                DrawerItem(
                  text: 'Clear History',
                  icon: const Image(
                      color: defaultColor,
                      image:
                          AssetImage('assets/images/delete.png'),
                      width: 40,
                      height: 40),
                  ontap: () async =>
                      await ParentCubit.get(context)
                          .clearHistory(),
                ),
                const SizedBox(
                  height: 15,
                ),
                DrawerItem(
                  text: 'Sign Out',
                  icon: const Image(
                    color: defaultColor,
                    image:
                        AssetImage('assets/images/signout.png'),
                    width: 40,
                    height: 40,
                  ),
                  ontap: () {
                    showDefaultDialog(
                      context,
                      title: 'Are you sure?',
                      content: const Text(
                        'Are you sure you want to log out?',
                        style: TextStyle(fontSize: 15),
                      ),
```

```
                                buttonText1: "NEVERMIND",
                                onPressed1: () {
                                  Navigator.of(context).pop();
                                },
                                buttonText2: "SIGN OUT",
                                onPressed2: () {
                                  ParentCubit.get(context).signOut();
                                },
                              );
                            },
                          ),
                          const SizedBox(
                            height: 10,
                          ),
                        ],
                      ),
                    )),
            ])),
        body: SingleChildScrollView(
          child: BlocConsumer<MainCubit, MainStates>(
            listener: (context, state) {
              if (state is EmailVerificationSuccessState) {
                ShowToast(
                    message: 'Email Verification Link Sent to Inbox',
                    state: ToastStates.SUCCESS);
              }

              if (state is EmailVerificationChangeState) {
                ShowToast(
                    message: 'Email Verified Successfully',
                    state: ToastStates.SUCCESS);
              }
            },
            builder: (context, state) {
              return Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                if (emailVerified == false)
                  EmailVerificationMessage(
                    reload: MainCubit.get(context).reload,
                    onPressed: MainCubit.get(context).reload
                        ? MainCubit.get(context).refreshUser
                        : MainCubit.get(context).verifyEmail,
                  ),
```

```
                    // balance
                    Container(
                      padding: const EdgeInsets.all(20),
                      alignment: AlignmentDirectional.topStart,
                      height: 150,
                      width: double.infinity,
                      decoration: BoxDecoration(
                          borderRadius: const BorderRadius.only(
                              bottomLeft: Radius.circular(30),
                              bottomRight: Radius.circular(30)),
                          color: Theme.of(context).primaryColor),
                      child: state is GetUserInfoLoading
                          ? const Center(
                              child: CircularProgressIndicator(
                                color: Colors.white,
                              ),
                            )
                          : Column(
                              children: [
                                Row(
                                  children: [
                                    SizedBox(
                                      width: 170,
                                      child: Column(
                                        crossAxisAlignment:
                                            CrossAxisAlignment.start,
                                        children: [
                                          const Text(
                                            'Balance',
                                            style: TextStyle(
                                                color: Colors.white,
                                                fontSize: 40,
                                                fontFamily: 'OpenSans'),
                                          ),
                                          const SizedBox(
                                            height: 5,
                                          ),
                                          Text(
                                            currencyFormat(
                                                ParentCubit.get(context)
                                                            .parent !=
                                                        null
                                                    ? ParentCubit.get(context)
                                                        .parent!
```

```
                                                .balance
                                          : 0),
                                maxLines: 1,
                                overflow: TextOverflow.ellipsis,
                                style: const TextStyle(
                                    color: Colors.white,
                                    fontSize: 25),
                              )
                            ],
                          ),
                        ),
                        const SizedBox(
                          width: 40,
                        ),
                        const Image(
                          color: Colors.white,
                          image: AssetImage(
                              'assets/images/purse.png'),
                          width: 100,
                          height: 100,
                        )
                      ],
                    ),
                  ],
                ),
              ),
              const SizedBox(
                height: 15,
              ),
              // family
              Padding(
                padding: const EdgeInsets.symmetric(horizontal: 20.0),
                child: Column(
                  mainAxisSize: MainAxisSize.max,
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text('Family',
                        style:
                            Theme.of(context).textTheme.headlineSmall),
                    const SizedBox(
                      height: 5,
                    ),
                    ConditionalBuilder(
                        condition: !ParentCubit.get(context)
```

```
                    .studentDataLoading,
                builder: (context) => ParentCubit.get(context)
                        .studentsData
                        .isNotEmpty
                    ? SizedBox(
                        height: 140,
                        width: double.infinity,
                        child: ListView.separated(
                            scrollDirection: Axis.horizontal,
                            shrinkWrap: true,
                            itemBuilder: (context, index) =>
                                FamilyMemberCard(
                                  ParentCubit.get(context)
                                      .studentsData
                                      .values
                                      .toList()[index],
                                ),
                            separatorBuilder:
                                (context, index) =>
                                    const SizedBox(
                                      width: 5,
                                    ),
                            itemCount: ParentCubit.get(context)
                                .studentsData
                                .length),
                      )
                    : Row(
                        children: [
                          Container(
                            padding: EdgeInsets.zero,
                            height: 140,
                            width: 130,
                            child: Card(
                              child: SizedBox(
                                width: double.infinity,
                                child: Column(
                                    crossAxisAlignment:
                                        CrossAxisAlignment
                                            .center,
                                    children: [
                                      const SizedBox(
                                        height: 10,
                                      ),
                                      CircleAvatar(
```

```
                                                    radius: 36,
                                                    backgroundColor:
                                                        Theme.of(context)
                                                            .primaryColor,
                                                    child: CircleAvatar(
                                                        radius: 35,
                                                        backgroundColor:
                                                            Colors.white,
                                                        child: IconButton(
                                                            onPressed:
                                                                () async {
                                                              await ParentCubit
                                                                      .get(
                                                                          context)
                                                                  .getCountries();
                                                            },
                                                            icon: const Icon(
                                                                Icons
                                                                    .add))),
                                                  ),
                                                  const SizedBox(
                                                    height: 10,
                                                  ),
                                                  SizedBox(
                                                    width: 80,
                                                    child: Text(
                                                      'Add Family Member',
                                                      style:
                                                          Theme.of(context)
                                                              .textTheme
                                                              .bodyLarge,
                                                      textAlign:
                                                          TextAlign.center,
                                                    ),
                                                  )
                                                ]),
                                            ),
                                          ),
                                        ),
                                      ],
                                    ),
                                fallback: (context) => const SizedBox(
                                      height: 140,
                                      child: Center(
```

```
                                child: CircularProgressIndicator(),
                              ),
                            )),
                    ],
                  ),
                ),
                const SizedBox(
                  height: 10,
                ),
                Padding(
                  padding: const EdgeInsets.symmetric(horizontal: 20.0),
                  child: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      Text('Activity',
                          style:
                              Theme.of(context).textTheme.headlineSmall),
                      const SizedBox(
                        height: 5,
                      ),
                      //activity
                      ConditionalBuilder(
                        condition:
                            !ParentCubit.get(context).activityLoading,
                        builder: (context) => ParentCubit.get(context)
                                    .activities
                                    .isNotEmpty &&
                                ParentCubit.get(context)
                                    .studentsData
                                    .isNotEmpty
                            ? ListView.separated(
                                physics:
                                    const NeverScrollableScrollPhysics(),
                                shrinkWrap: true,
                                itemBuilder: (context, index) =>
                                    ActivityItem(
                                        model:
                                            ParentCubit.get(context)
                                                .activities[index],
                                        studentsData:
                                            ParentCubit.get(context)
                                                .studentsData
                                                .values
                                                .toList()),
```

```
                                separatorBuilder: (context, index) =>
                                    const SizedBox(
                                      height: 5,
                                    ),
                                itemCount: ParentCubit.get(context)
                                    .activities
                                    .length)
                            : Center(
                                child: SizedBox(
                                  width: 200,
                                  height: 230,
                                  child: Column(
                                    mainAxisAlignment:
                                        MainAxisAlignment.center,
                                    children: const [
                                      Image(
                                        height: 130,
                                        width: 130,
                                        image: AssetImage(
                                            'assets/images/no_activity.png'),
                                        fit: BoxFit.cover,
                                      ),
                                      SizedBox(
                                        height: 10,
                                      ),
                                      Text(
                                        'No activity Found',
                                        style: TextStyle(
                                            fontSize: 15,
                                            color: Colors.black54,
                                            fontWeight: FontWeight.w500),
                                      ),
                                    ],
                                  ),
                                ),
                              ),
                        fallback: (context) => const Padding(
                          padding: EdgeInsets.only(top: 50.0),
                          child:
                              Center(child: CircularProgressIndicator()),
                        ),
                      )
                    ],
                  ),
```

```
                    )
                  ],
                )
              ;
                },
                  ),
              ));
        },
      ),
    );
  }
}
```

7.2.3 Teacher Home Screen Code

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:stguard/layout/teacher/cubit/cubit.dart';
import 'package:stguard/layout/teacher/cubit/states.dart';
import 'package:stguard/modules/login/login_screen.dart';
import 'package:stguard/modules/teacher/add_new_task/add_new_task.dart';
import 'package:stguard/modules/teacher/join_community/join_community_screen.dart';
import 'package:stguard/shared/components/components.dart';
import 'package:stguard/shared/internet_cubit/cubit.dart';
import 'package:stguard/shared/styles/themes.dart';

class TeacherHomeScreen extends StatelessWidget {
  const TeacherHomeScreen({super.key});

  @override
  Widget build(BuildContext context) {
    TeacherCubit.get(context)
      .getTeacherPath();
    return BlocListener<InternetCubit, ConnectionStatus>(
      listener: (context, state) {
        if (state == ConnectionStatus.disconnected) {
          showSnackBar(context,
              message: 'You are currently offline.', icon: Icons.wifi_off);
        } else if (state == ConnectionStatus.connected) {
          showSnackBar(context,
              message: 'Your internet connection has been restored.',
              icon: Icons.wifi);
        }
      },
```

```
      child: BlocConsumer<TeacherCubit, TeacherStates>(
        listener: (context, state) {
          if (state is UserSignOutSuccessState) {
            navigateAndFinish(context, LoginScreen());
          }
        },
        builder: (context, state) {
          return Scaffold(
              appBar: AppBar(
                elevation: 0,
              ),
              drawer: Drawer(
                  child: Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      children: [
                    Expanded(
                      flex: 1,
                      child: DrawerHeader(
                        padding: const EdgeInsets.only(left: 20, top: 20),
                        child: TeacherCubit.get(context).teacher != null
                            ? UserInfo(
                                userModel: TeacherCubit.get(context).teacher!,
                              )
                            : const Center(
                                child: CircularProgressIndicator(),
                              ),
                      ),
                    ),
                    Expanded(
                      flex: 2,
                      child: Padding(
                        padding: const EdgeInsets.symmetric(horizontal: 20),
                        child: Column(
                          crossAxisAlignment: CrossAxisAlignment.start,
                          children: [
                            Text('Settings',
                                style: Theme.of(context)
                                    .textTheme
                                    .headlineMedium!
                                    .copyWith(color: defaultColor)),
                            const SizedBox(
                              height: 20,
                            ),
                            DrawerItem(
```

```
                    text: 'Reset ID',
                    ontap: () => TeacherCubit.get(context).resetId(),
                    icon: const Image(
                        color: defaultColor,
                        image: AssetImage('assets/images/undo.png'),
                        width: 30,
                        height: 30),
                  ),
                  const SizedBox(
                    height: 15,
                  ),

                  DrawerItem(
                    text: 'Sign Out',
                    icon: const Image (
                        color: defaultColor,
                        image:AssetImage('assets/images/signout.png'),
                        width: 30,height: 30),
                    ontap: () {
                      showDefaultDialog(
                        context,
                        title: 'Are you sure?',
                        content: const Text(
                          'Are you sure you want to log out?',
                          style: TextStyle(fontSize: 15),
                        ),
                        buttonText1: "NEVERMIND",
                        onPressed1: () {
                          Navigator.of(context). pop();
                        },
                        buttonText2: "SIGN OUT",
                        onPressed2: () {
                          TeacherCubit.get(context).signOut();
                        },
                      );
                    },
                  ),
                  const SizedBox(
                    height: 10,
                  ),
                ],
              ),
            ),
          ),
```

```
                ])),
            body: TeacherCubit.get(context).teacherPathLoading
                ? const Center(
                    child: CircularProgressIndicator(),
                  )
                : TeacherCubit.get(context).teacherPath == null
                    ? JoinCommunityScreen(
                        image:'assets/images/teacher-and-open-class-door.png')
                    : AddNewTaskScreen());
        },
      ),
    );
  }}
```

7.2.4 Canteen Home Screen Code

```
import 'package:conditional_builder_null_safety/conditional_builder_null_safety.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:stguard/layout/canteen/cubit/cubit.dart';
import 'package:stguard/layout/canteen/cubit/states.dart';
import 'package:stguard/modules/canteen/inventory/inventory_screen.dart';
import 'package:stguard/modules/canteen/join_community/join_community_screen.dart';
import 'package:stguard/modules/canteen/products/products_screen.dart';
import 'package:stguard/modules/canteen/trans_details/trans_details_screen.dart';
import 'package:stguard/modules/login/login_screen.dart';
import 'package:stguard/shared/components/components.dart';
import 'package:stguard/shared/components/constants.dart';
import 'package:stguard/shared/internet_cubit/cubit.dart';
import 'package:stguard/shared/main_cubit/main_cubit.dart';
import 'package:stguard/shared/main_cubit/states.dart';
import 'package:stguard/shared/styles/themes.dart';

class CanteenHomeScreen extends StatelessWidget {
  const CanteenHomeScreen({super.key});

  @override
  Widget build(BuildContext context) {
    CanteenCubit.get(context)
      ..getCanteenInfo()
      ..getCanteenPath();
    return BlocListener<InternetCubit, ConnectionStatus>(
      listener: (context, state) {
        if (state == ConnectionStatus.disconnected) {
          showSnackBar(context,
```

```
            message: 'You are currently offline.', icon: Icons.wifi_off);
        } else if (state == ConnectionStatus.connected) {
          showSnackBar(context,
              message: 'Your internet connection has been restored.',
              icon: Icons.wifi);
        }
      },
      child: BlocConsumer<CanteenCubit, CanteenStates>(
        listener: (context, state) {
          if (state is UserSignOutSuccessState) {
            navigateAndFinish(context, LoginScreen());
          }


        },
        builder: (context, state) {
          return Scaffold(
              appBar: AppBar(
                elevation: 0,
              ),
              drawer: Drawer(
                  child: Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      children: [
                    Expanded(
                      child: SizedBox(
                        height: 300,
                        child: DrawerHeader(
                          padding: const EdgeInsets.only(left: 20, top: 20),
                          child: CanteenCubit.get(context).canteen != null
                              ? UserInfo(
                                  userModel: CanteenCubit.get(context).canteen!,
                                )
                              : const Center(
                                  child: CircularProgressIndicator(),
                                ),
                        ),
                      ),
                    ),
                    Expanded(
                      flex: 2,
                      child: Padding(
                        padding: const EdgeInsets.symmetric(horizontal: 20),
                        child: Column(
```

```
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      Text(
                        'Settings',
                        style: Theme.of(context)
                            .textTheme
                            .headlineMedium!
                            .copyWith(color: defaultColor),
                      ),
                      const SizedBox(
                        height: 20,
                      ),
                      DrawerItem(
                        text: 'Reset ID',
                        ontap: () => CanteenCubit.get(context).resetId(),
                        icon: const Image(
                            color: defaultColor,
                            fit: BoxFit.scaleDown,
                            image: AssetImage('assets/images/undo.png'),
                            width: 35,
                            height: 35),
                      ),
                      const SizedBox(
                        height: 15,
                      ),
                      DrawerItem(
                        text: 'Sign Out',
                        icon: const Image(
                            color: defaultColor,
                            image:
                                AssetImage('assets/images/signout.png'),
                            width: 35,
                            height: 35),
                        ontap: () {
                          showDefaultDialog(
                            context,
                            title: 'Are you sure?',
                            content: const Text(
                              'Are you sure you want to log out?',
                              style: TextStyle(fontSize: 15),
                            ),
                            buttonText1: "NEVERMIND",
                            onPressed1: () {
                              Navigator.of(context).pop();
```

```
                    },
                    buttonText2: "SIGN OUT",
                    onPressed2: () {
                      CanteenCubit.get(context).signOut();
                    },
                  );
                },
              ),
              const SizedBox(
                height: 10,
              ),
            ],
          ),
        ),
      ),
    ])),
  body: CanteenCubit.get(context).canteenPathLoading
      ? const Center(
          child: CircularProgressIndicator(),
        )
      : ConditionalBuilder(
          condition:
              CanteenCubit.get(context).schoolCanteenPath != null,
          builder: (context) => SingleChildScrollView(
                child: BlocConsumer<MainCubit, MainStates>(
                  listener: (context, state) {
                    if (state is EmailVerificationSuccessState) {
                      ShowToast(
                          message:
                              'Email Verification Link Sent to Inbox',
                          state: ToastStates.SUCCESS);
                    }

                    if (state is EmailVerificationChangeState) {
                      ShowToast(
                          message: 'Email Verified Successfully',
                          state: ToastStates.SUCCESS);
                    }
                  },
                  builder: (context, state) {
                    return Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      children: [
                        if (emailVerified == false)
```

```
                    EmailVerificationMessage(
                      reload: MainCubit.get(context).reload,
                      onPressed: MainCubit.get(context).reload
                          ? MainCubit.get(context).refreshUser
                          : MainCubit.get(context)
                              .verifyEmail,
                    ),
                  Container(
                    height: 200,
                    padding: const EdgeInsets.symmetric(
                        horizontal: 20),
                    alignment: AlignmentDirectional.center,
                    width: double.infinity,
                    decoration: BoxDecoration(
                        borderRadius: const BorderRadius.only(
                            bottomLeft: Radius.circular(30),
                            bottomRight: Radius.circular(30)),
                        color:
                            Theme.of(context).primaryColor),
                    child: Container(
                      height: 160,
                      width: double.infinity,
                      decoration: BoxDecoration(
                          color: Colors.white,
                          borderRadius:
                              BorderRadiusDirectional
                                  .circular(30)),
                      child: Padding(
                        padding: const EdgeInsets.all(15.0),
                        child: Column(
                          children: [
                            Row(
                              children: [
                                const Image(
                                    width: 35,
                                    height: 35,
                                    image: AssetImage(
                                        'assets/images/sales.png')),
                                const SizedBox(
                                  width: 10,
                                ),
                                Text(
                                  'Daily Sales Report',
                                  style: Theme.of(context)
```

```
                                                    .textTheme
                                                    .headlineSmall!
                                                    .copyWith(
                                                        fontWeight:
                                                            FontWeight.w700,
                                                        fontFamily:
                                                            'OpenSans'),
                                                ),
                                            ],
                                        ),
                                        const SizedBox(
                                          height: 20,
                                        ),
                                        Row(
                                          children: [
                                            SizedBox(
                                                width: 200,
                                                child: Text(
                                                  'Orders Counts',
                                                  style: Theme.of(context)
                                                      .textTheme
                                                      .headlineSmall!
                                                      .copyWith(
                                                          fontWeight:
                                                              FontWeight
                                                                  .w500),
                                                )),
                                            const SizedBox(
                                              width: 10,
                                            ),
                                            Expanded(
                                              child: Text(
                                                '${CanteenCubit.get(context).can
teenDetails != null ? CanteenCubit.get(context).canteenDetails!.dailyTransactions ?? 0
: 0}',
                                                style: Theme.of(context)
                                                    .textTheme
                                                    .titleLarge,
                                              ),
                                            ),
                                          ],
                                        ),
                                        const SizedBox(
                                          height: 5,
```

```
                    ),
                    Row(
                      children: [
                        SizedBox(
                            width: 200,
                            child: Text(
                              'Total Cash',
                              style: Theme.of(context)
                                  .textTheme
                                  .headlineSmall!
                                  .copyWith(
                                      fontWeight:
                                          FontWeight
                                              .w500),
                            )),
                        const SizedBox(
                          width: 10,
                        ),
                        Expanded(
                          child: Text(
                            currencyFormat(CanteenCubit
                                            .get(
                                                context)
                                        .canteenDetails !=
                                    null
                                ? CanteenCubit.get(
                                            context)
                                        .canteenDetails!
                                        .dailyRevenue ??
                                    0
                                : 0),
                            style: Theme.of(context)
                                .textTheme
                                .titleLarge!,
                            overflow:
                                TextOverflow.ellipsis,
                          ),
                        ),
                      ],
                    ),
                  ],
                ),
              ),
            ),
```

```
                    ),
                    const SizedBox(
                      height: 10,
                    ),
                    Row(
                      mainAxisAlignment:
                          MainAxisAlignment.center,
                      children: [
                        Expanded(
                            child: DefaultButton2(
                          width: 130,
                          height: 130,
                          image: 'assets/images/menu.png',
                          text: 'Menu',
                          imageWidth: 70,
                          imageHeight: 70,
                          textStyle: Theme.of(context)
                              .textTheme
                              .titleLarge!
                              .copyWith(
                                  fontWeight: FontWeight.bold),
                          onPressed: () => navigateTo(
                              context, ProductsScreen()),
                        )),
                        Expanded(
                            child: DefaultButton2(
                          width: 130,
                          height: 130,
                          text: 'Inventory',
                          textStyle: Theme.of(context)
                              .textTheme
                              .titleLarge!
                              .copyWith(
                                  fontWeight: FontWeight.bold),
                          image: 'assets/images/inventory.png',
                          imageHeight: 70,
                          imageWidth: 70,
                          onPressed: () => navigateTo(context,
                              CanteenInventoryScreen()),
                        )),
                      ],
                    ),
                    const SizedBox(
                      height: 10,
```

```
                    ),
                    Padding(
                      padding: const EdgeInsets.symmetric(
                          horizontal: 20.0),
                      child: Column(
                        crossAxisAlignment:
                            CrossAxisAlignment.start,
                        children: [
                          Row(
                            children: [
                              Text(
                                'Transaction',
                                style: Theme.of(context)
                                    .textTheme
                                    .headlineSmall,
                              ),
                              const Spacer(),
                              TextButton(
                                  onPressed: () {
                                    showDatePicker(
                                            context: context,
                                            initialDate:
                                                DateTime.now(),
                                            firstDate:
                                                DateTime(2020),
                                            lastDate:
                                                DateTime.now())
                                        .then((value) {
                                      CanteenCubit.get(context)
                                          .setTransDate(
                                              date: value);
                                    });
                                  },
                                  child: Text(
                                    checkTodayCanteenCubit.get(
                                                context)
                                            .getTransBy) ??
                                        getDate(
                                            CanteenCubit.get(
                                                    context)
                                                .getTransBy,
                                            format:
                                                'dd/MM/yyyy'),
                                    style: Theme.of(context)
```

```
                                                    .textTheme
                                                    .titleMedium!
                                                    .copyWith(
                                                        color:
                                                            defaultColor),
                                            )),
                                        const SizedBox(
                                          width: 5,
                                        )
                                      ],
                                    ),
                                    const SizedBox(
                                      height: 10,
                                    ),
                                    ConditionalBuilder(
                                      condition:
                                          !CanteenCubit.get(context)
                                              .getTransLoading,
                                      builder: (context) =>
                                          ConditionalBuilder(
                                        condition:
                                            CanteenCubit.get(context)
                                                        .transactions !=
                                                    null &&
                                                CanteenCubit.get(context)
                                                    .transactions!
                                                    .isNotEmpty,
                                        builder: (context) =>
                                            ListView.separated(
                                          shrinkWrap: true,
                                          physics:
                                           const NeverScrollableScrollPhysics(),
                                          itemBuilder: (context, index) =>
                                              TransactionItem(
                                                  onTap: () {
                                                    navigateTo(
                                                        context,
                                                        CanteeenTransactionDetai
lsScreen(
                                                          trans: CanteenCubit
                                                                  .get(
                                                                      context)
                                                          .transactions![index],
                                                        ));
```

```
                                                    },
                                                    trans: CanteenCubit.get(
                                                                context)
                                                            .transactions![
                                                        index]),
                                              separatorBuilder:
                                                  (context, index) =>
                                                      const SizedBox(
                                                height: 5,
                                              ),
                                              itemCount:
                                                  CanteenCubit.get(context)
                                                      .transactions!
                                                      .length,
                                            ),
                                            fallback: (context) => Padding(
                                              padding: const EdgeInsets.only(
                                                  right: 40,
                                                  left: 40,
                                                  bottom: 40),
                                              child: Center(
                                                child: Column(
                                                  children: [
                                                    const Image(
                                                        height: 150,
                                                        width: 150,
                                                        image: AssetImage(
                                                            'assets/images/no_activi
ty.png')),
                                                    const SizedBox(
                                                      height: 10,
                                                    ),
                                                    Text(
                                                      'No Transaction Found',
                                                      style: Theme.of(context)
                                                          .textTheme
                                                          .bodySmall,
                                                    )
                                                  ],
                                                ),
                                              ),
                                            ),
                                          ),
                                          fallback: (context) => const Center(
```

```
                                                child:
                                                    CircularProgressIndicator()),
                                            )
                                        ],
                                    ),
                                )
                            ],
                        );
                    },
                ),
            ),
        fallback: (context) =>
            const CanteenJoinCommunityScreen()));
    },
  ),
);
}
}
```

8. Future Work

1) **Publishing school news:**

   In later updates of the mobile application, there will be ability for the parent to know the latest news of the school, in other words, school news will be shown to the parent on the mobile application, the news of the school will include every event happens in the school like that include occasions and celebrations.

2) **School fees payment:**

   In later updates of the mobile application, the parent will be able to pay the school fees of his child through the mobile application and could pay school tuitions on stages and, he could the school uniform and books.

3) **A student role will be added:**

   In later updates of the mobile application, the student will have a role in the mobile application that makes the students able to use the app in many things like sending the homework and attending online quizzes.

4) **Bracelet detects bad language:**

   The student bracelet will be embedded by an AI component that would be able detect the words said by the student and would detect if the student talked in bad language.

5) **Ranking students:**

   In later updates of the mobile application, the parent will be able to know his child rank on the class and on other class.

6) **Full ready database for canteen products:**

   In newer versions of the school system a full ready database will be added and connected toadding itobile application that will contain data of the most popular products and the data about products would be entered according to nutrition specialists that will revise the ingredients of every product and the allergy that

the ingredients cause, then when the canteen worker adds new products he could check if he can find this product in the database and if not he would add it manually but before adding it there would be a revision from the specialists on these products that are added by the canteen worker.

7) Apply leave:

One of the features that will be in later updates is that the parent would be able to make an apply leave request to the school throw the mobile application to take the child during school time. Also, the parent will be able to mention the reason of the leaving and the time that he will take his child by the mobile application.

9. Conclusion

In the traditional schools making parents connected to their children is not effective and does not give the parents enough information about their children, also it does not allow parents to monitor their children's activities in school.

For the purposes of connecting students with their parents and overcoming previous problems, a smart school system is made.

The smart school system guarantees that parents will know everything about their children in school, this would be accomplished by making each student wear a bracelet that holds an SMS card and keeps track of the child's current location.

A mobile application is used by the parents to know their children's locations at any time. Also, make them know the arrival of their children at school time and departure time from school, this happens by scanning the student's bracelet when arriving or leaving the school.

Parents can put limitations and restrictions on their children where in some circumstances there are students that suffers from a specific allergy or more than one allergy, The allergy could be specified by the parent on the mobile application and student won't be able to buy products from the school canteen that contain the allergens that cause this allergy.

Moreover, Parents can put a balance for his children to buy from school canteen by scanning bracelet with spending limit, also parent can put calories limit for his child.

Canteen workers also use the mobile application but with a role different from the role of parents to make canteen transactions where the mobile application abort any transaction that contain allergies causes an allergic reaction to the student and abort transactions when the total products calories exceed the limit of calories specified by the parent, also it does not make transactions when the balance or spending limit of student is less than the total price of the transaction.

Teachers also can use the mobile application by their role where it is used to record classes attendance and notify parents with it, share classes notes then send them to parents and share students grades with their parents.

10.References

- https://www.stanfordchildrens.org/en/topic/default?id=egg-allergy-diet-for-children-90-P01684
- https://www.stanfordchildrens.org/en/topic/default?id=peanut-allergy-diet-for-children-90-P01700
- https://www.stanfordchildrens.org/en/topic/default?id=milk-allergy-diet-for-children-90-P01696
- https://www.stanfordchildrens.org/en/topic/default?id=wheat-allergy-diet-for-children-90-P01712
- https://www.stanfordchildrens.org/en/topic/default?id=soy-allergy-diet-for-children-90-P01709
- https://www.perfecto.io/blog/what-is-flutter-framework
- https://blog.logrocket.com/state-management-flutter-bloc-pattern/
- https://firebase.google.com/docs/firestore
- https://circuitdigest.com/microcontroller-projects/interfacing-sim800l-module-with-esp32
- https://randomnerdtutorials.com/esp32-pinout-reference-gpios/
- https://how2electronics.com/interfacing-pn532-nfc-rfid-module-with-arduino/
- https://blog.asksensors.com/iot-cloud-based-gps-tracking-esp32-gps-neo-6m-module/
- https://circuitdigest.com/microcontroller-projects/interfacing-sim800l-module-with-esp32