المعهد العالي للدراسات التكنولوجيـة في المواصلات بتونس
Institut Supérieur des Etudes Technologiques en Communications de Tunis

Departement : STIC

Reference : L3

Licence Appliquée en Sciences et technologies de l'Information et de la Communication

Tutored Project

Digital access and Payment system with QR code recognition "RESTO'COM"

Internship carried out by: **SAIFEDDINE KHIR**
ISMAIL GHODHBANI

Class: **STIC L3 SR C**
STIC L3 RST A

Supervised by: **Mr. MOEZ BALTI**

University Year: 2025-2026

Acknowledgment

We would like to express our deepest gratitude to **ISET'COM** for providing us with the opportunity to carry out this project, which has been an invaluable experience for developing our practical skills and deepening our understanding of IoT and digital payment systems.

We extend our sincere appreciation to **Mr. Moez BALTI**, our supervisor, for his continuous guidance, expertise and encouragement throughout the project. His mentorship has been essential in helping us overcome challenges and achieve our objectives.

Finally, we would like to thank everyone who contributed, directly or indirectly, to the success of this project, making this journey both enriching and rewarding.

Table of Contents:

List of figures:

List of tables:

List of acronyms:

AP — Access Point

API — Application Programming Interface

CLI — Command Line Interface

GPIO — General Purpose Input/Output

ID — Identifier

JWT — JSON Web Token

LCD — Liquid Crystal Display

QR — Quick Response

REST API — Representational State Transfer Application Programming Interface

SPI — Serial Peripheral Interface

TFT — Thin-Film Transistor

General Introduction:

In recent years, the digitalization of daily activities has become a major focus for institutions and organizations aiming to improve efficiency and user experience. In particular, the management of transactions in university cafeterias remains a challenge due to the high number of students, cash handling errors and long waiting times. Traditional payment methods are often slow, prone to human error and lack transparency, which can lead to operational inefficiencies.

The objective of this project is to design and implement a digital payment system for the university cafeteria using **IoT technologies and a mobile application**. The system is based on an **ESP32 module**, which acts as the transaction terminal, a **Raspberry Pi**, which serves as a local server and database manager and a **mobile application** that allows students to manage their accounts, view their balance and make cashless payments.

This project aims to provide a fast, secure and user-friendly payment solution while offering real-time tracking and transaction management for the cafeteria administration. The proposed solution not only reduces waiting times but also improves transparency and operational efficiency, making it easier to analyze sales and consumption patterns.

The report is structured into two main chapters. **Chapter 1: Problem statement and study of the existent situation** present the context, highlights the limitations of current systems, identifies user needs and justifies the proposed solution. **Chapter 2: Realization** details the design and implementation of the system, including hardware selection, circuit design, software development, communication protocols, 3D modeling and testing results. Finally, a general conclusion summarizes the achievements of the project and suggests potential future improvements.

Chapter 1

1. Problem statement and study of the existent situation

1.1. Introduction:

This chapter provides an overview of the project's context including the current problems, the possible solutions and the proposed idea.

1.2. Context and Problem Statement:

1.2.1. University Cafeteria Environment:

University cafeterias play a critical role in student life, particularly in institutions such as ISETCOM and SUP'COM, where a large number of students depend on subsidized meals for their daily nutrition. These cafeterias must manage a high flow of users during short lunch and dinner time slots, which requires fast, reliable and well-organized payment and control systems. Ensuring fairness, efficiency and transparency in meal distribution is therefore an essential operational challenge for the administration.

1.2.2. Current Payment Method:

Currently, the payment system used in the university restaurant relies on paper-based meal tickets. Students purchase a booklet of tickets from the financial department, where each ticket represents one meal with a fixed value (200 millimes per meal). At each service, students must physically present a ticket to the cafeteria staff in order to access a meal. The staff then manually collects and counts these tickets to estimate the number of meals served and to compare them with food consumption, including main dishes and additional items such as yogurts, fruits, etc.

This traditional approach involves significant cash handling and manual operations, which introduce several limitations. First, the management of physical tickets and cash increases the risk of loss, misplacement and accounting inconsistencies. Moreover, the absence of a digital system makes it difficult to maintain an accurate record of transactions, forcing the administration to rely on manual counting and estimations that can be imprecise.

Figure 1-1 University restaurant Iset'COM.

1.2.3. Service Efficiency and Queuing Issues:

Another major issue is the formation of long queues, especially during peak hours. Since each transaction requires physical verification, ticket collection and visual inspection of the student, the service speed is considerably reduced. This congestion not only increases waiting times but also puts significant pressure on cafeteria staff. During busy periods, this pressure can lead to rushed decisions and reduced verification accuracy. Another point worth mentioning is that since there are no efficient way to determine and optimize the portions of food that need to be prepared each day, it can result in meals shortages where students have to wait for another batch of meals to be prepared and served.

1.2.4. Lack of Traceability and Resource Management:

In addition, the current system provides no reliable transaction tracking. There is no automated way to verify how many meals a student has consumed, how many meals are served per service or how resources are distributed over time. This lack of traceability prevents effective statistical analysis, makes auditing difficult and limits the administration's ability to optimize food supply and reduce waste.

1.2.5. Security Risks and Human Errors:

The heavy dependence on human intervention also exposes the system to human errors and security issues. Staff members are required to visually recognize students to prevent unauthorized access such as individuals from other institutions attempting to benefit from the cafeteria. This method is unreliable, especially during crowded periods and can easily lead to mistakes, fraud or unfair treatment.

1.3. Needs Analysis:

Based on the limitations of the current paper ticket system, the identified functional and non-functional needs for a modernized cafeteria management system are as follows, the table down below summarizes these needs based on their priority:

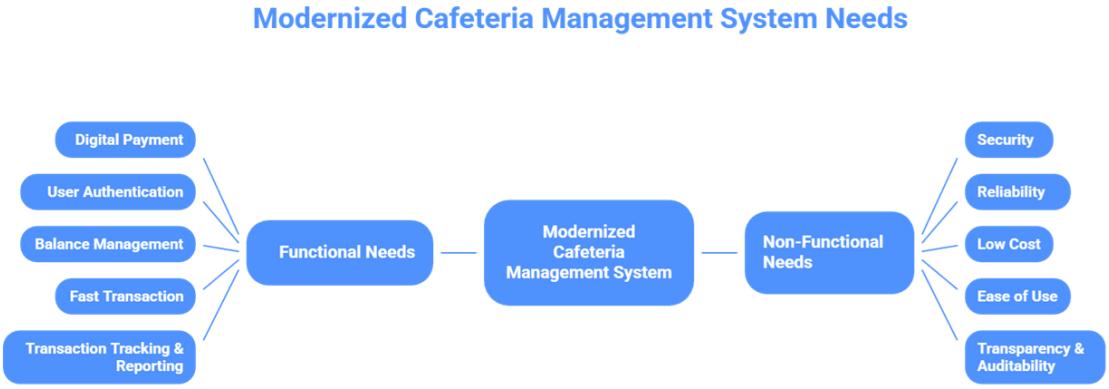


Figure 1-2 Modernized Cafeteria Management System Needs

1.3.1. Functional Needs:

- **Digital Payment:** The system must replace physical cash and ticket handling with a secure digital transaction method to eliminate loss, misplacement and accounting errors.
- **User Authentication:** A robust mechanism is required to reliably identify and authenticate authorized students during each service to prevent unauthorized access, duplicate meals and fraud.
- **Balance Management:** Users must be able to digitally check their meal balance or credit. The administration needs a centralized system to manage student accounts, track individual consumption and facilitate easy top-ups.
- **Fast Transaction:** To manage the high flow of users during short lunch and dinner slots, the payment and verification process must be extremely quick to avoid long queues and ensure efficient service.
- **Transaction Tracking & Reporting:** The system must automatically and accurately record every meal served, linking it to a user and a timestamp. This enables precise statistics, consumption analysis and auditing for the administration.

1.3.2. Non-Functional Needs:

- **Security:** The system must protect financial transactions, prevent unauthorized access to meals or data and safeguard user account information.
- **Reliability:** It must operate consistently during every service period without failure, as any downtime directly disrupts student access to meals.
- **Low Cost:** The solution should aim to minimize initial investment and long-term operational costs to remain aligned with the subsidized nature of the university cafeteria service.
- **Ease of Use:** The interface for both students (to pay/check balance) and staff (to verify/ manage) must be intuitive and require minimal training to ensure smooth adoption and operation.
- **Transparency & Auditability:** The system should provide clear, accurate and accessible records for the administration to monitor usage, reconcile food supply with actual consumption and perform audits effectively.

Table 1 Requirements table

Category	Requirement ID	Description	Priority
Functional	FN-01	Digital Payment: The system shall process meal payments electronically, eliminating physical tickets and cash.	High
	FN-02	User Authentication: The system shall verify a user's identity and institutional eligibility before granting meal access.	High
	FN-03	Balance Management: Students shall be able to view their remaining meal credits/balance. The administration shall be able to manage user accounts and top-up funds.	High
	FN-04	Fast Transaction: The payment authentication process shall be completed in under 3 seconds per user.	High

	FN-05	Transaction Tracking: The system shall automatically log all meal transactions for reporting and analysis.	Medium
Non-Functional	NF-01	Security: The system shall prevent fraud and unauthorized access and encrypt all sensitive user and transaction data.	High
	NF-02	Reliability: The system shall maintain an operational uptime of ≥ 99% during service hours (lunch and dinner).	High
	NF-03	Low Cost: The solution shall prioritize affordable, scalable technology with low maintenance requirements.	Medium
	NF-04	Ease of Use: The user interface shall be intuitive for students and staff, requiring less than 30 minutes of training.	High
	NF-05	Auditability: The system shall generate accurate daily and monthly reports on meals served, user consumption and revenue for administrative review.	Medium

1.4. Study of the State of the Art:

In order to find the best solution, it is important to study and compare the available methods and technologies used in similar fields to pinpoint exactly the most appropriate one. After analyzing multiple methods, these are points that stood out:

1.4.1. RFID card systems:

Principle: Radio Frequency Identification (RFID) is a wireless identification technology that uses radio waves to automatically identify and track objects or individuals. An RFID system typically consists of a tag containing stored data and a reader that communicates with the tag without physical contact, enabling fast, secure and contactless data exchange.

Students are issued personalized contactless RFID cards. To pay for a meal, the card is tapped on an RFID reader installed at the cafeteria entrance or point-of-sale. The system verifies the student's identity and eligibility, debits the card's prepaid balance or meal credit and grants access.

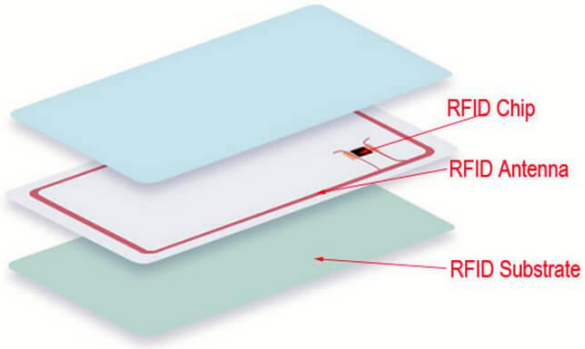


Figure 1-3 RFID card components.

Advantages:

- Very Fast Transaction: Tap-and-go process takes less than a second, ideal for high-traffic periods.
- High Reliability: Physical cards are durable and the technology is mature and stable.
- Excellent Ease of Use: Simple physical interaction requires minimal training for users and staff.

Disadvantages:

- Medium Initial Cost: Requires investment in RFID cards, readers and backend software infrastructure.
- Physical Card Dependency: Risk of loss, forgetfulness, or damage, requiring replacement procedures.
- Limited Functionality: The card itself is typically a single-purpose device for cafeteria access unlike a multi-function smartphone.

1.4.2. QR code payments:

Principle: Each student has a unique QR code linked to their account, displayed on a mobile phone app or a printed card (Student card). At the point of service, a staff member scans the code using a scanner or a tablet/phone camera. The system validates the code and processes the transaction.

Figure 1-4 A QR code example.

Advantages:

- Low Implementation Cost: Utilizes existing smartphones or student cards and low-cost scanners or tablets; no need for specialized hardware per user.
- High Flexibility: QR codes can be dynamically generated for added security (e.g., time-based) and can be integrated into various mobile applications.
- Good Traceability: Easily captures transaction data linked to the user's digital profile.

Disadvantages:

- Slower Transaction Speed: Scanning can be slower than RFID, especially in poor lighting or if the code is damaged on-screen.
- Potential for Fraud: Static QR codes can be easily copied or shared, allowing unauthorized users to benefit unless paired with additional authentication (like a photo ID check).

1.4.3. Mobile Wallet Applications:

Principle: Each student owns a digital wallet integrated into a mobile application, where a virtual balance or linked payment account is stored. At the point of service, the payment is initiated through the application and validated via an authentication method such as a PIN code, biometric verification, or an in-app confirmation. The transaction is then processed over the network and recorded in the system in real time.

Figure 1-5 Mobile Wallet Application

Advantages:

- High User Convenience: Payments are performed directly from the smartphone without the need for physical cards or cash.
- Enhanced Security: Authentication mechanisms such as PIN codes or biometrics reduce the risk of unauthorized transactions.
- Real-Time Monitoring: Users and administrators can instantly track balances and transaction history.
- Easy Scalability: The system can be easily updated and extended with new features through software updates.

Disadvantages:

- Smartphone Dependency: Requires users to own a compatible smartphone with sufficient battery life.
- Internet Connectivity Requirement: Transactions may be delayed or unavailable in case of network issues.

1.5. Proposed Solution:

After understanding and analyzing various aspects of each type of method and technology used in similar fields. We decided to gather most of the different advantages encountered into one solution.

Figure 1-6 RESTO'COM logo.

The solution is based on three main components: an **ESP-based payment terminal**, a **Raspberry Pi** acting as a local server and controller and a **mobile application** for users.

The ESP-based terminal is installed at the cafeteria point of sale and serves as the user interaction interface. It manages transaction initiation, displays payment information and communicates with the server over a wireless network. Thanks to its low cost, integrated Wi-Fi and sufficient processing capabilities, the ESP module is well suited for real-time transaction handling.

The Raspberry Pi acts as the central server and system controller. Hosting an administrative dashboard that helps staff visualize the information and logs as well as interacting with students' accounts.

The mobile application allows users to access their digital account, check their balance and share funds with others whilst keeping track of all the transactions that took place. It provides a user-friendly interface and serves as the main interaction tool, enabling a fully cashless payment experience.

The project is titled **"RESTO'COM"**, the general concept is to enable fast and secure QR code transactions by combining IoT devices with a local server architecture and a mobile interface. The main features of the system include digital user authentication, real-time balance updates, transaction traceability, reduced waiting time at the cafeteria and centralized management of payments.

1.6. Conclusion:

In summary, the current payment methods in the university cafeteria presents several challenges. Cash handling is slow, prone to human error and lacks transparency, leading to long waiting times and difficulties in tracking transactions. Existing solutions such as RFID, QR codes or commercial mobile wallets are either costly to deploy, limited in flexibility or dependent on external services and internet connectivity, making them less suitable for a university environment.

The proposed solution directly addresses these limitations. It offers a low-cost, secure and user-friendly approach, ensuring fast, traceable and reliable transactions while giving the university full control over data and system management.

This justification naturally leads to the next step: the implementation phase, where the design and development of the hardware, software and communication architecture will be realized to bring the proposed solution from concept to a working prototype.

Chapter 2

2. Realization:

2.1. Introduction:

This chapter focuses on the implementation phase; choosing the right hardware and software for the project, choosing the most appropriate means of communication and protocols and most importantly the steps taken to build the first prototype of RESTO'COM.

2.2. Global System Architecture:

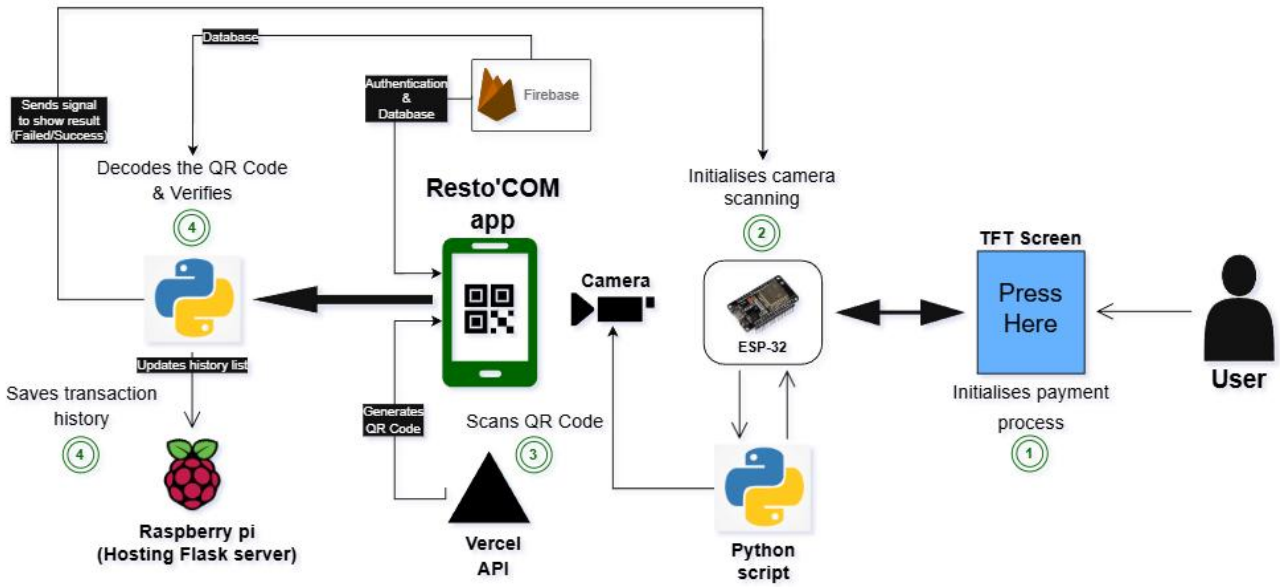


Figure 2-1 Overall architecture diagram

The system workflow, illustrated in the diagram, begins when a user presses the ESP32-connected TFT screen to initiate payment. This triggers the ESP32 to signal a Python script, which activates the camera for scanning while the screen updates to instruct the user to present a QR code.

Concurrently, the user must open a mobile app to authenticate via Firebase by inserting his e-mail and password and then generate a valid QR code through a Vercel API, then place it before the camera. Upon scanning, the Python script verifies the code's validity and authentication data. If successful, the transaction is logged and sent to a Flask server on a Raspberry Pi and the TFT screen confirms completion. If verification fails, the screen prompts a retry without logging. The entire process is designed for efficiency, completing in under two seconds to minimize time and labor.

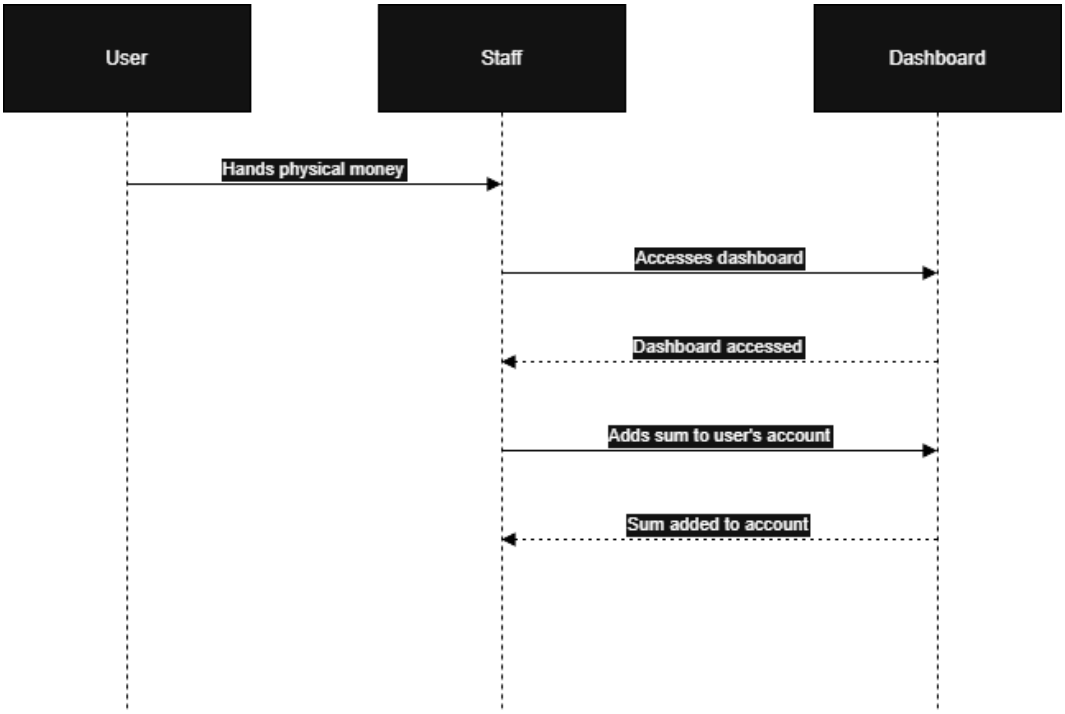


Figure 2-2 User and Staff interactions UML diagram.

Additionally, when the user hands a physical sum of money, instead of receiving a booklet of tickets, that amount gets added directly to his account. Therefor he is not strictly obliged to pay for fixed sum of 10 tickets anymore like the current system.

2.3. Hardware Design:

2.3.1. Choice of Components:

ESP32: The ESP32 was selected for this project due to its strong balance between performance, integration and cost, making it well-suited for embedded systems such as an automatic university restaurant payment system with a TFT display. It integrates a dual-core processor, sufficient RAM and built-in Wi-Fi and Bluetooth, which enables reliable network communication for payment processing without requiring external modules. The ESP32 also provides multiple GPIOs and native SPI support, ensuring efficient and fast communication with the TFT screen while maintaining a responsive user interface. In addition, its low power consumption, wide community support and mature development ecosystem simplify development, debugging and future scalability of the system, for more specification (See Appendix).

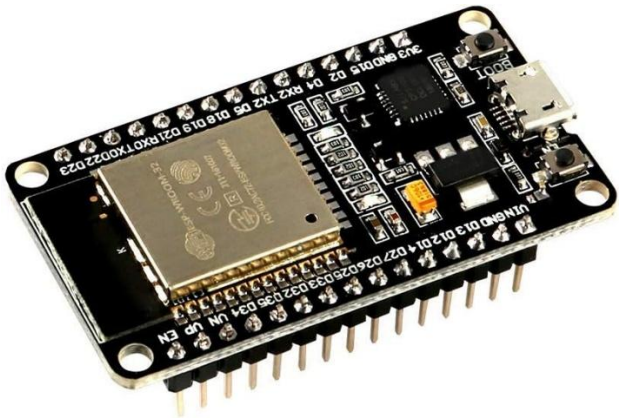


Figure 2-3 ESP32 DEVKIT V1

Raspberry Pi: The model used in this project is the Raspberry Pi 3 Model B+. It was initially selected due to its versatility and ability to run a full Linux operating system, which made it suitable for hosting a Flask server and managing high-level application logic. It was also originally intended to handle QR code scanning using a camera module. However, during implementation, it became clear that this model has limited processing power and memory for simultaneously hosting a web server, handling network communication and performing real-time verification of high-resolution QR codes. This resulted in performance constraints and increased latency, making it unsuitable for reliable QR code processing under these conditions. Consequently, the Raspberry Pi 3 Model B+ was retained for server-side tasks only, while more appropriate hardware was considered for real-time scanning and user-interface operations, for more specification (See Appendix).

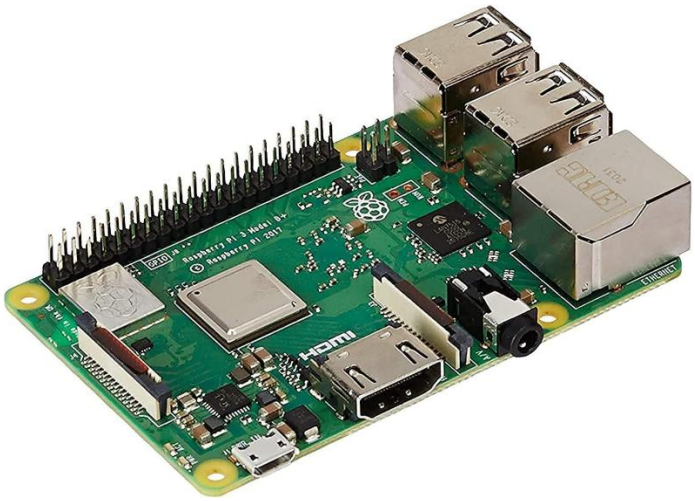


Figure 2-4 Raspberry Pi 3 Model B+

TFT LCD Touch screen: The project incorporates a touch screen module to provide an interactive user interface for payment, display menus and feedback. The screen has a resolution of 240×320 pixels driven by a controller such as the ILI9341, offering rich color depth and sufficient clarity for small graphical elements and text. It uses an SPI (Serial Peripheral Interface) for communication, which conserves GPIO pins while maintaining responsive updates from the microcontroller and includes a resistive touch panel for basic touch input compatible with microcontrollers like the ESP32. This type of display is compact, low-power and widely supported by graphics libraries, making it suitable for embedded GUI applications in IoT, for more specification (See Appendix).

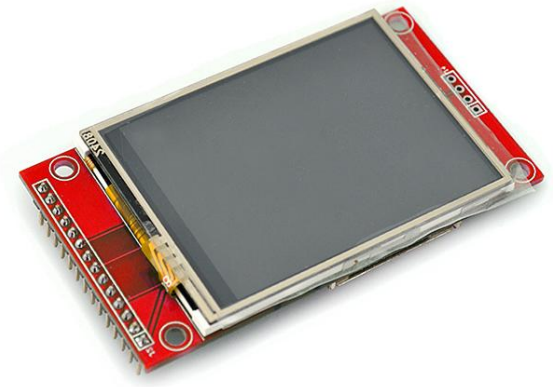


Figure 2-5 TFT LCD Touch screen module.

Webcam Camera: As mentioned before, due to the performance issues of the current model of the Raspberry Pi, it is not ideal to make it handle the process of scanning the QR codes and host the interactive dashboard at the same time. For this reason, a dedicated webcam connected to a laptop is used to handle this heavy task.

Figure 2-6 Laptop's dedicated webcam.

Ideally, it would be better to use the dedicated webcam that comes with the Raspberry Pi in case we switch to a much stronger model that can handle multiple tasks.

Figure 2-7 Raspberry Pi Camera.

2.3.2. Electronic Circuit:

The wiring between the ESP32 and the Raspberry Pi was designed to ensure stable communication and reliable power distribution. A common ground (GND) was established between the two devices, which is essential for correct signal referencing and to avoid communication errors.

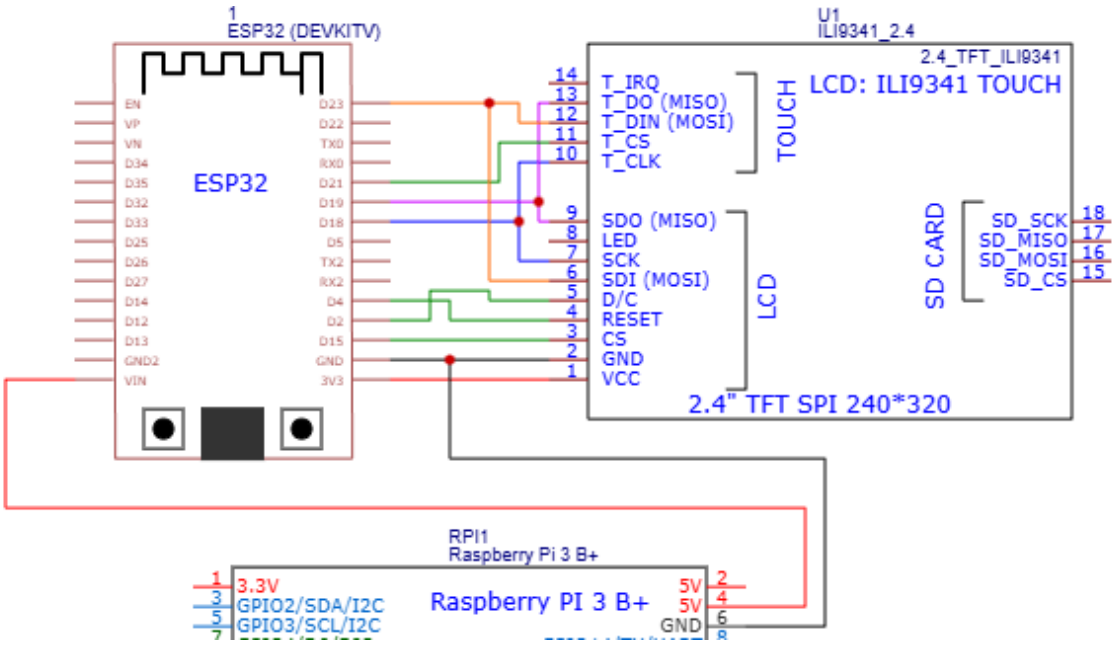


Figure 2-8 Components wiring schematic.

The main power source of the system is the Raspberry Pi, which is powered via a mini-USB cable connected either to its dedicated charger or to a PC USB port. Although the ESP32 operates internally at 3.3 V, it is equipped with an onboard voltage regulator that allows it to be safely powered using a 5 V supply. For this reason, the ESP32 was powered from the 5 V output of the Raspberry Pi in order to reduce voltage drops and ensure stable operation, especially during Wi-Fi communication and display usage.

Additionally, the ESP32 was interfaced with a TFT display using the SPI communication protocol. The wiring includes the necessary SPI lines (MOSI, MISO, SCK and CS), along with control pins such as DC and RESET and a shared ground to guarantee proper display operation and reliable graphical performance. The full wiring schematics can be found in the Appendix section.

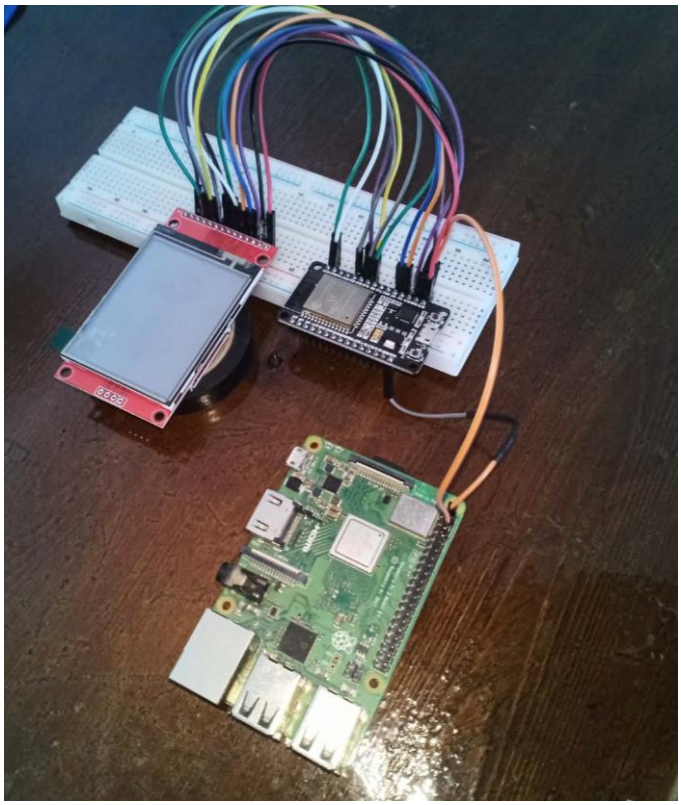


Figure 2-9 Testing the components I.

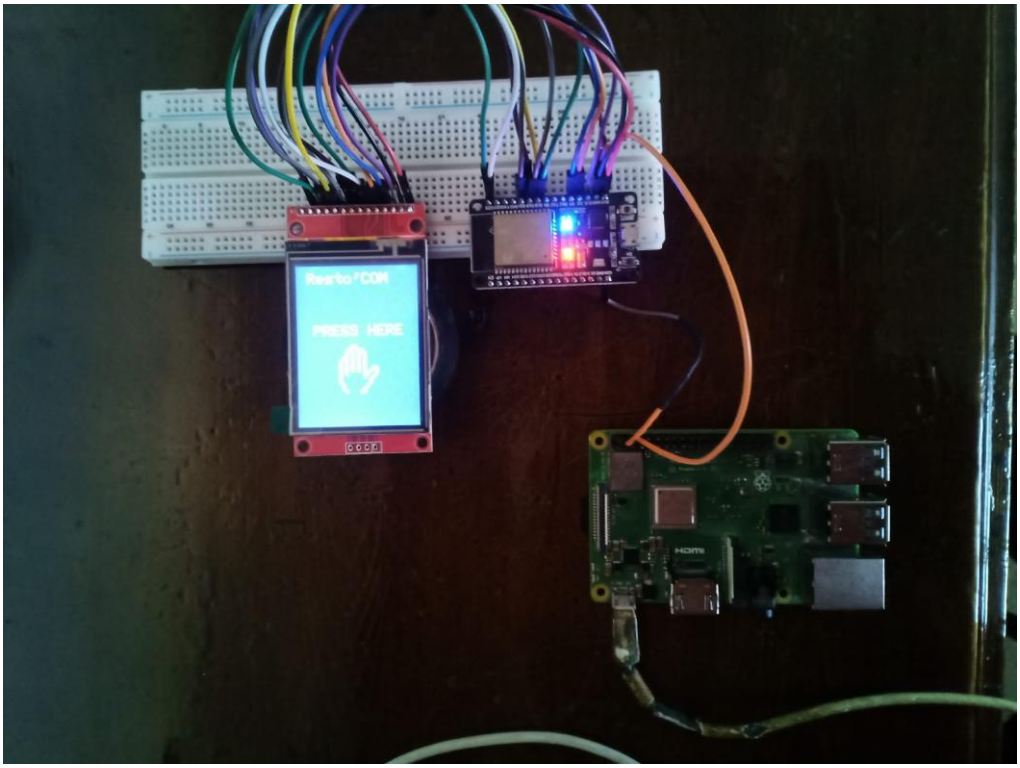


Figure 2-10 Testing the components II.

2.4. Mobile Application:

2.4.1. Framework and services:

- **Flutter:** Flutter is an open-source UI framework developed by Google that allows the creation of mobile applications using a single codebase for multiple platforms, mainly Android and iOS. Unlike native development, where separate applications must be developed and maintained for each platform (such as Java/Kotlin for Android and Swift for iOS), Flutter uses one shared codebase written in Dart, which is then compiled into native performance applications.

Figure 2-11 Flutter logo.

The mobile application is developed using Flutter framework and follows a modular, layered architecture that separates concerns between user interface, business logic and data

management. The application is structured as a campus wallet system that enables students to manage their digital balance, transfer funds and generate QR codes for transactions.

- **Firebase**: Firebase is a backend-as-a-service platform provided by Google that offers a set of ready-to-use services for application development, including user authentication, real-time databases and cloud storage. Unlike traditional backend development, which requires setting up and maintaining a dedicated server and database, Firebase simplifies data management by providing a scalable and secure cloud infrastructure.

Figure 2-12 Firebase logo.

Using Firebase's service allowed us to save a lot of time in deploying and setting up the mobile app and since both Firebase and Flutter were developed by Google. The integration process was seamless and highly responsive. One thing worth mentioning is that, in order to get the full experience, you need a paid subscription, still the free plan offers most of the functionalities we need. That's why we adopted another service to host our API.

- **Vercel**: Vercel is a cloud platform used for deploying and hosting web applications with minimal configuration. It enables fast deployment, automatic updates and reliable availability.

Figure 2-13 Vercel logo.

In this project, Vercel provides a simple and efficient solution for hosting the QR code generating API reducing server management complexity and allowing the system to focus on functionality rather than infrastructure.

2.4.2. Application Structure and Navigation Flow:

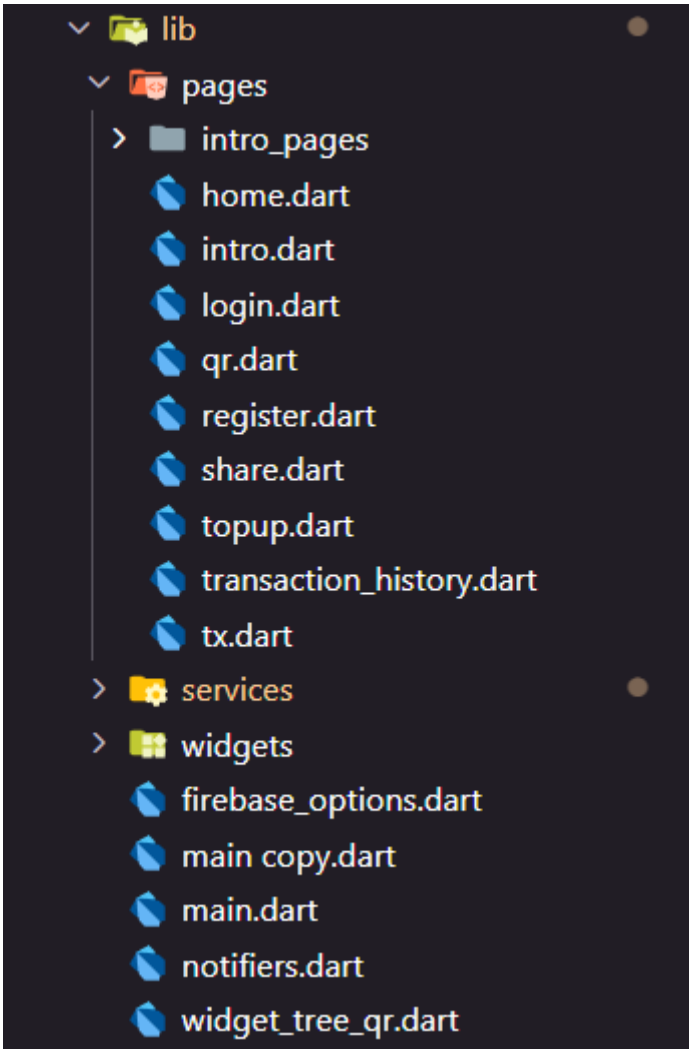


Figure 2-14 Mobile Application Structure.

The application implements a hierarchical page structure organized within the lib/pages/ directory, with each page serving a specific functional purpose. The main entry point (main.dart) initializes Firebase services and establishes authentication persistence before launching the application. The navigation flow begins with the LoginPage (login.dart), which serves as the authentication gateway and determines the user's subsequent path based on their account status.

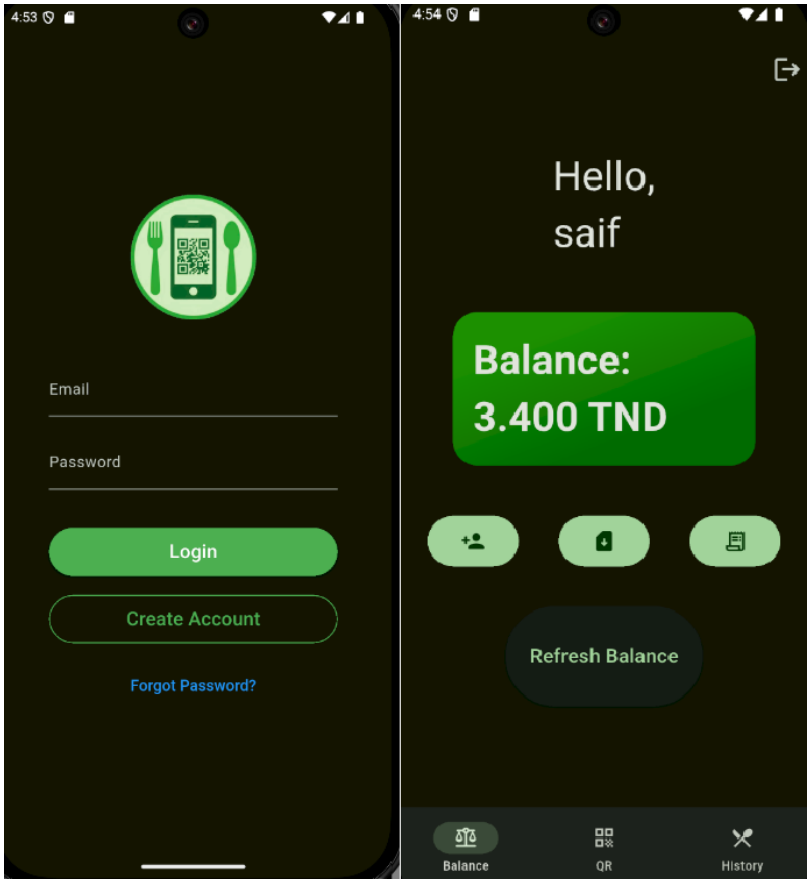


Figure 2-15 Login page and Main dashboard of RESTO'COM app.

After successfully login in using the appropriate credentials, the user will be prompted with the main dashboard page as seen in the figure above. This page will display his current balance and gives him access to other sections of the app. Alternatively; new users will first be prompted with an onboarding page that quickly highlights the key points of the app.

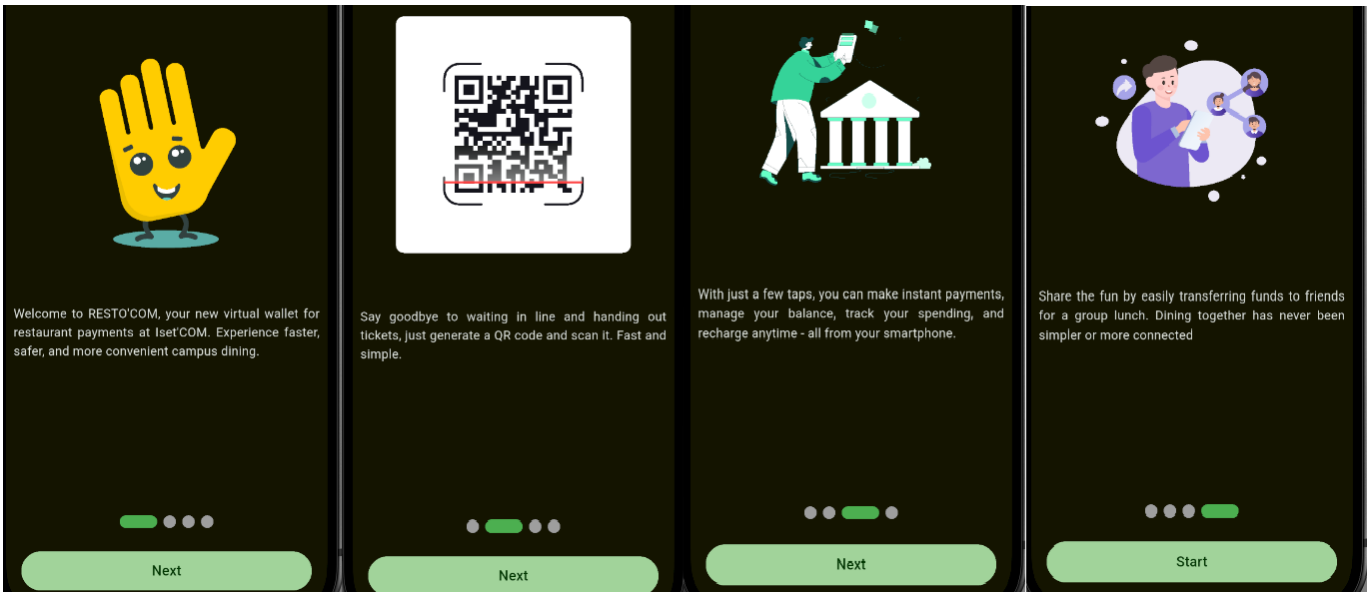


Figure 2-16 Onboarding pages of RESTO'COM app.

Figure 2-17 QR code generation page of RESTO'COM app.

The QR code section is connected with the Vercel API, it sends a *GET* request along with the current User ID and generates a proper code. The QR code is characterized with a duration of 60 seconds and a unique nonce to avoid replay attacks. Thanks to Vercel's built in firewall, we are able to counter spamming attacks as well so users won't be able to request too many codes in a short period of time. All of these information are encrypted using a private key shared with the camera system and additional layers of protection ensuring secure communication.

```
Future<void> fetchQrFromApi() async {
  try {
    final url = Uri.parse(
      "https:                                /api/generate_qr?student_id=${widget.studentId}"
    );
    final response = await http.get(url);
```

Figure 2-18 API code snippet.

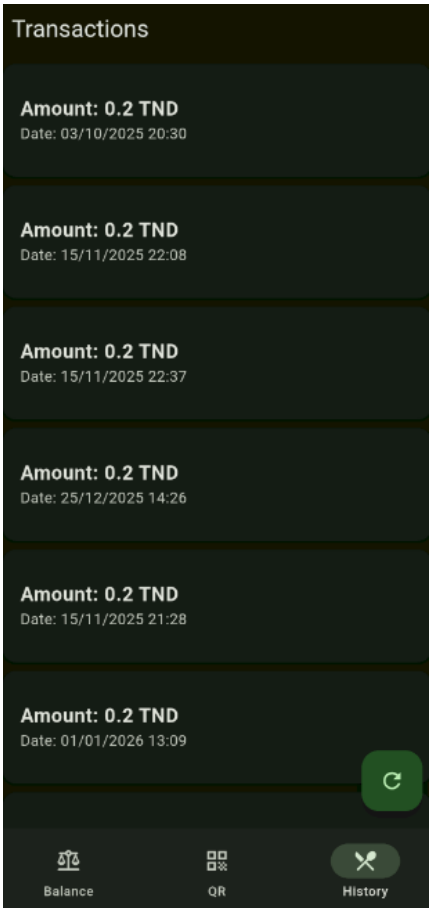


Figure 2-19 History page of RESTO'COM app.

The page represented in the figure above holds timestamps and information of the user's last meals, or rather his payment history.

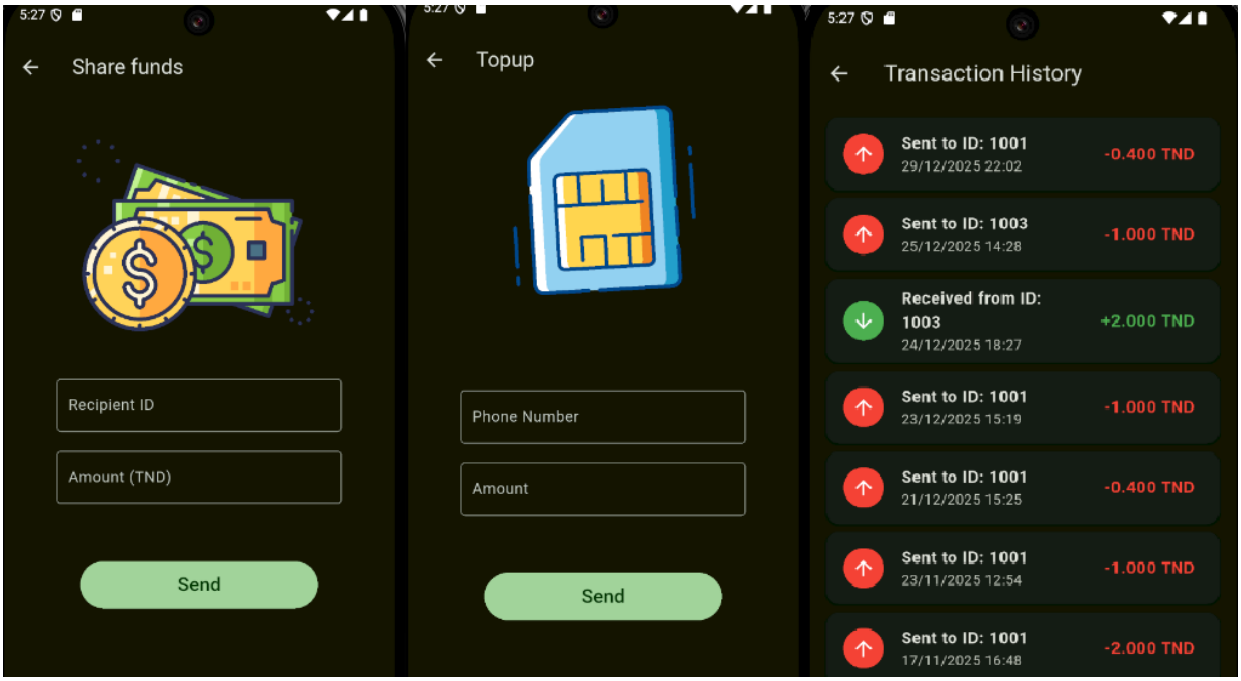


Figure 2-20 Share funds, Topup and Transaction history pages of RESTO'COM app.

The figure above showcases the remaining pages of the mobile app, a section where the user can share his balance with others, a section dedicated to adding funds to your balance using your mobile phone credit. This part is not fully functional, as for it to work, a special contract needs to be established with other phone operators (Tunisie Telecom, Ooredoo, Orange etc..). Therefore, this section behaves as a proof of concept, to further elaborate the potential of the phone application. Finally, a section dedicated to showcasing the user's transaction history, it works similarly to the history page mentioned before. However, this one focuses on the transactions made between users and topping up your balance.

2.4.3. Deployment and surveillance:

After finishing up coding the different pages of the mobile app, the next step was to connect it to different services to complete all the functionalities.

For the storage and authentication, we connected our app with **Firebase services**. The process went as following:

- Created an account in **Firebase**.
- Downloaded the Firebase CLI:

```
curl -sL https://firebase.tools | bash
```

- Used the newly created account to login:

```
firebase login
```

- Finally, connected the app with the account:

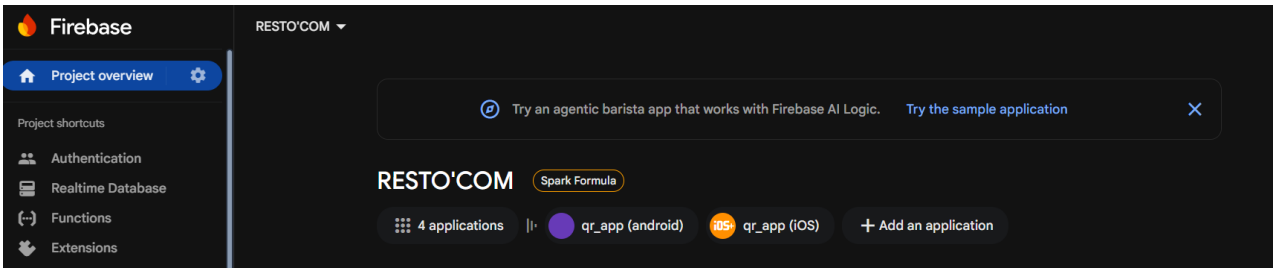


Figure 2-21 Firebase dashboard.

- Now we are able to access the different services from Authentication and Realtime Database:

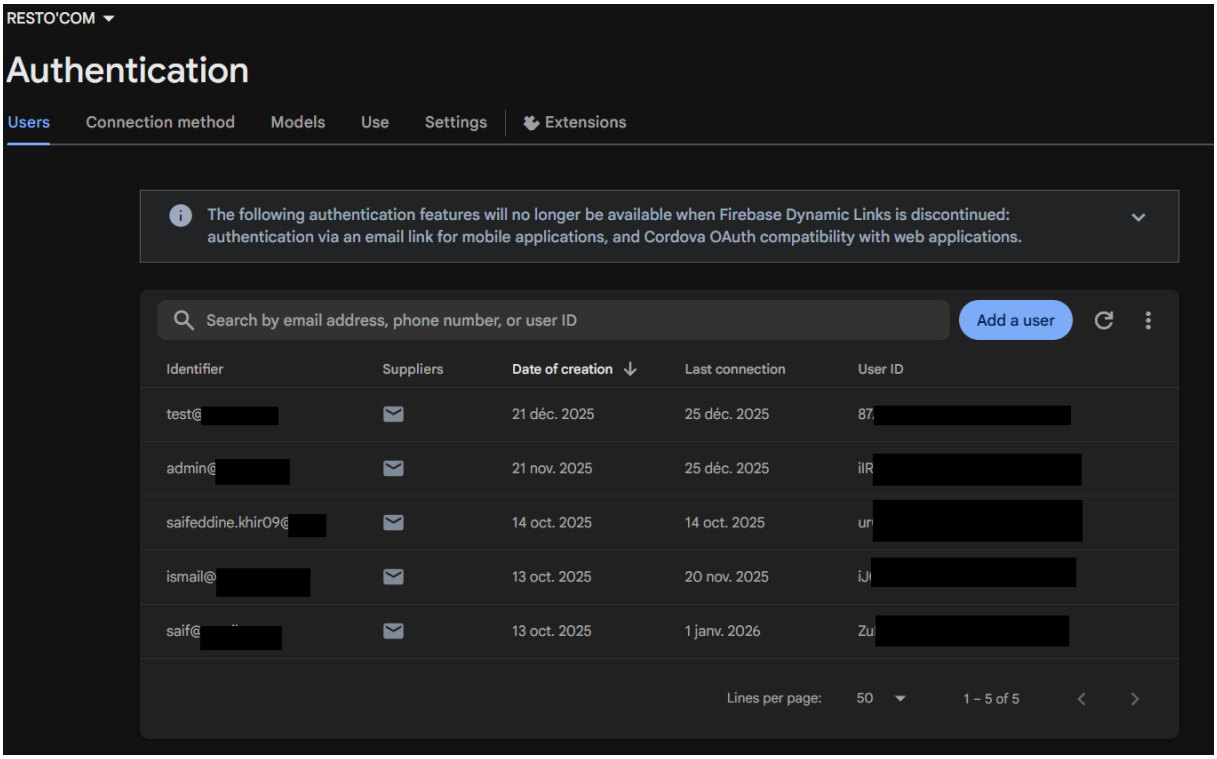


Figure 2-22 Firebase Authentication Panel.

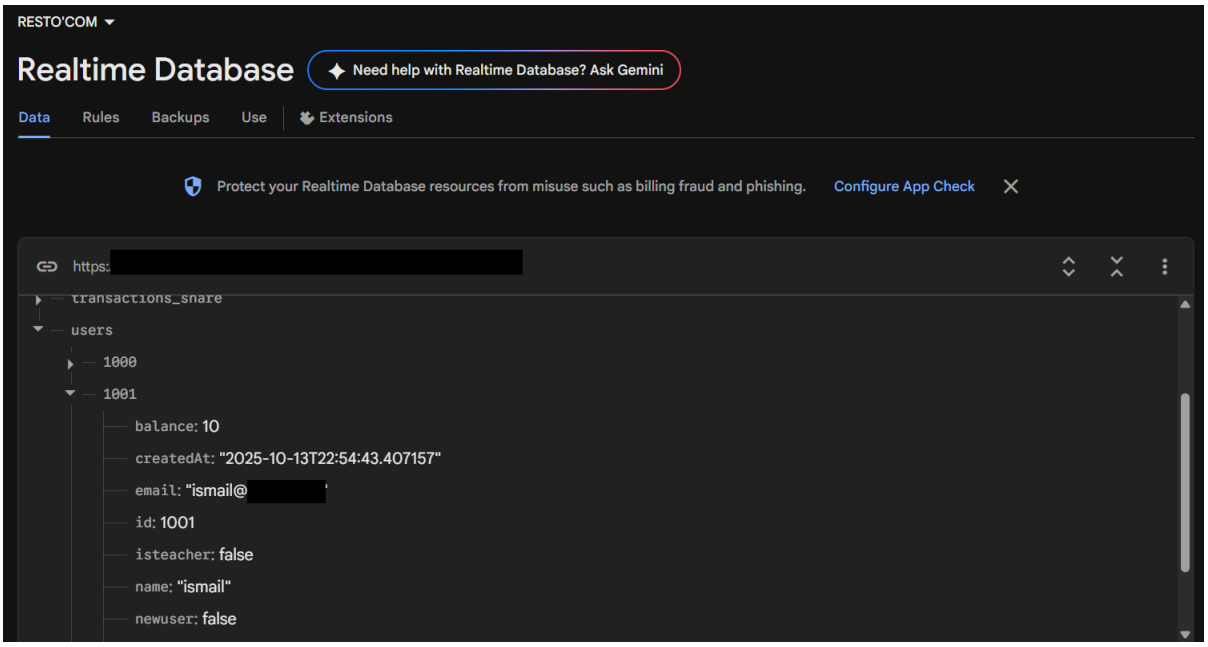


Figure 2-23 Firebase Realtime Database Panel.

- In order to communicate between our code and Firebase a private key is needed, it can be obtained from the website too:

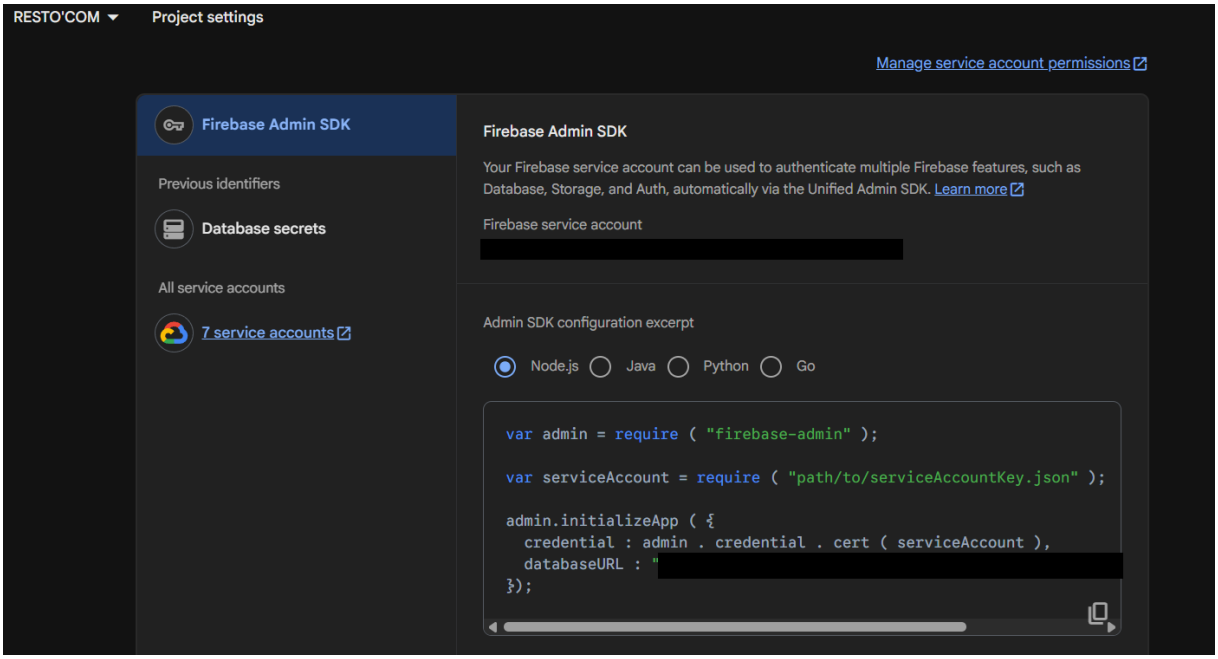


Figure 2-24 Obtaining Firebase Admin SDK key.

Now for our **API hosted by Vercel**:

- Created an account in **Vercel**.
- We first pushed our API code to GitHub in order for Vercel to access it, the files are structured as following:

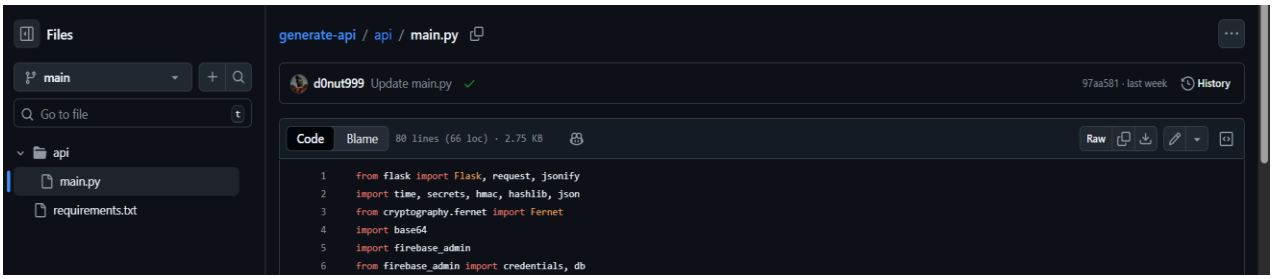


Figure 2-25 QR code API structure in GitHub.

- We deployed the API and configured the environment variables:

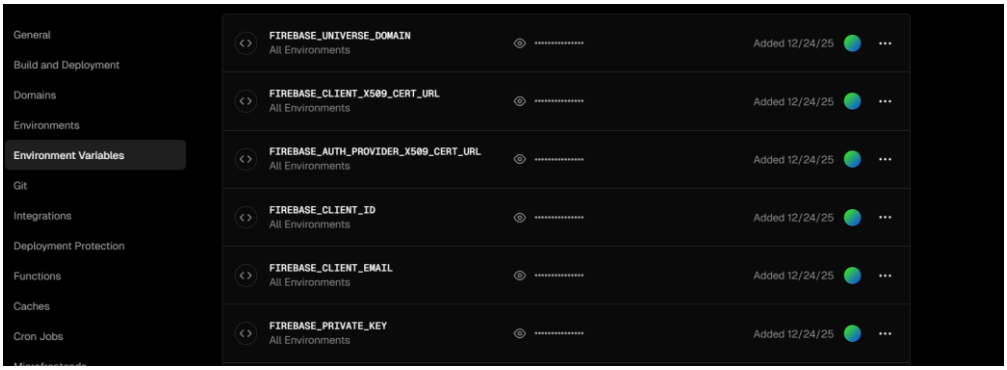


Figure 2-26 Vercel's Environment Variables configuration.

➢ We were able to log all the API activities thanks to Vercel's logs feature:

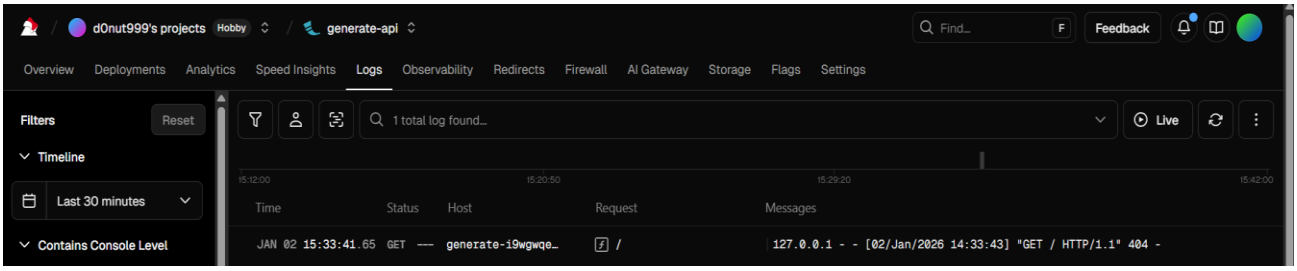


Figure 2-27 Vercel's logs section.

2.5. Staff Dashboard:

The Flask-based administrative dashboard provides a web interface for monitoring and managing the campus wallet system. Built using Python Flask framework, the dashboard integrates with Firebase services to offer real-time transaction oversight and user account management capabilities.

2.5.1. System Architecture and Authentication:

The dashboard, hosted by the Raspberry Pi, implements a secure authentication system using Firebase Admin SDK with email-based login. Access is restricted to authorized administrators through a whitelist mechanism, ensuring only designated personnel can view sensitive financial data. The system maintains session management with proper logout functionality and token verification for all administrative operations.

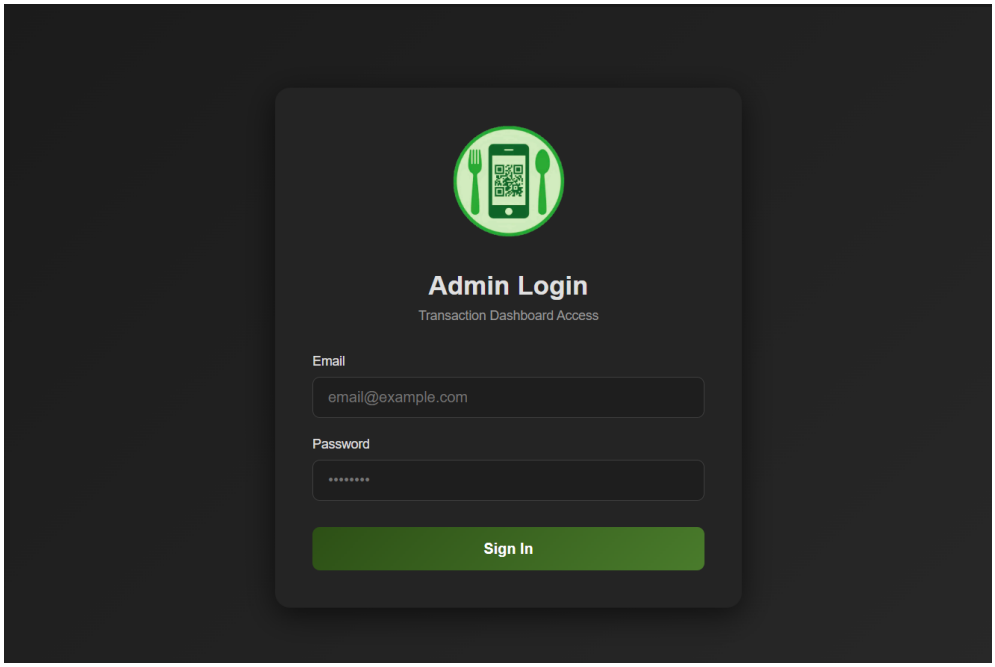


Figure 2-28 Staff dashboard login page.

2.5.2. Core Dashboard Features:

The main interface presents a comprehensive transaction history view displaying all campus wallet transactions in a responsive table format. Each transaction record includes ticket ID, user ID, teacher/student status and timestamp information. The dashboard automatically generates sequential ticket IDs for easy reference and tracking purposes.

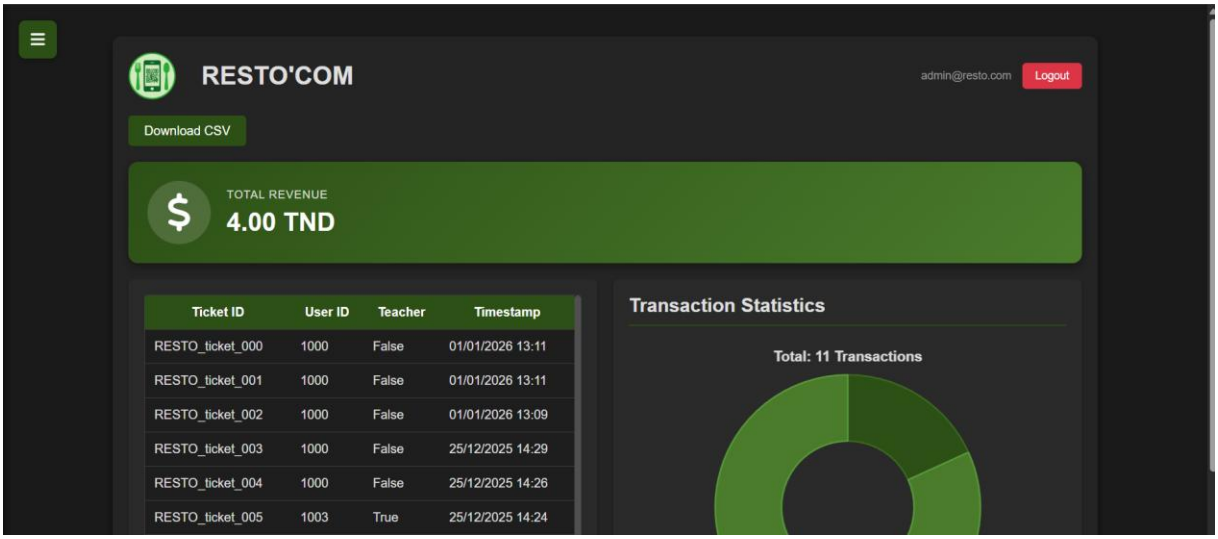


Figure 2-29 Staff dashboard main page.

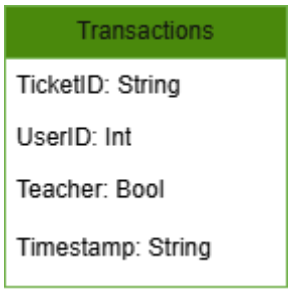


Figure 2-30 Transactions format.

A statistics panel provides visual analytics through Chart.js integration, displaying transaction distribution between teachers and students using an interactive doughnut chart. The system calculates and displays total revenue metrics, giving administrators quick insights into system usage patterns.

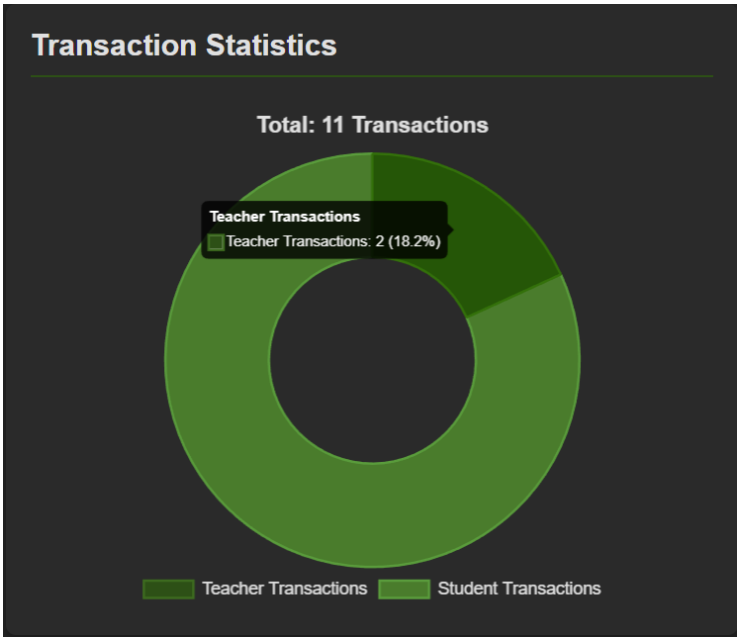


Figure 2-31 Transactions chart.

2.5.3. Administrative Functions:

The dashboard includes a dedicated fund management section where administrators can add money to user accounts. This feature includes user verification functionality that displays account details before fund addition, ensuring accuracy in financial operations. All administrative actions are logged with timestamps and admin identification for audit purposes.

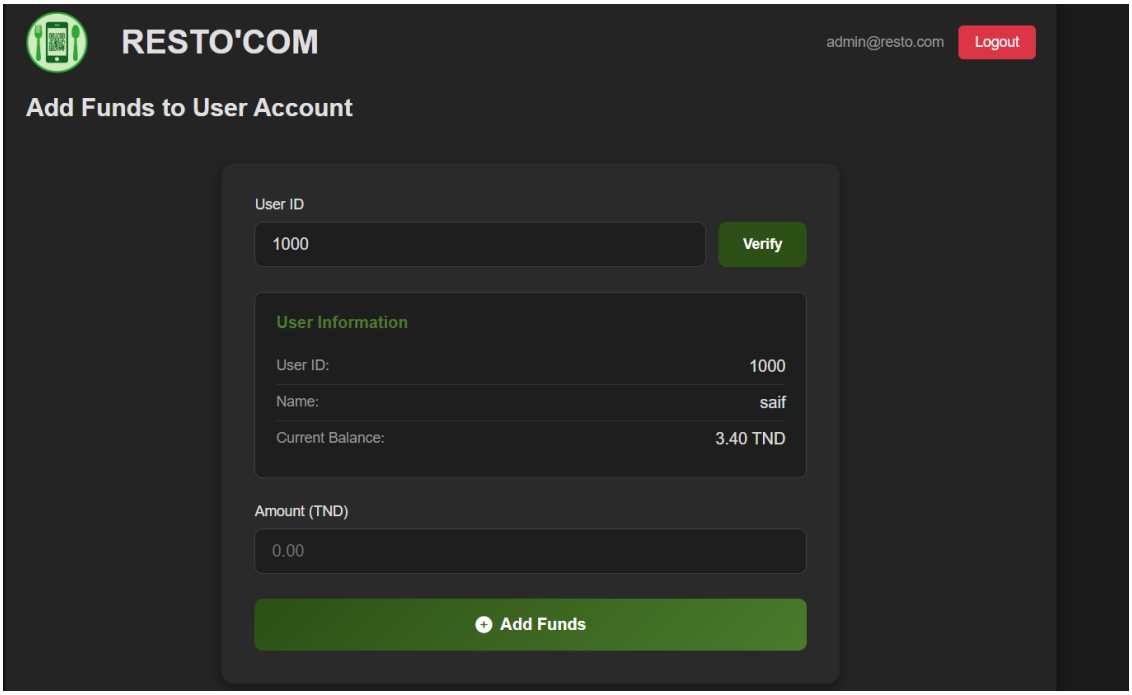


Figure 2-32 Staff dashboard add funds section.

Each staff action is stored in the following format represented by the figure down below.

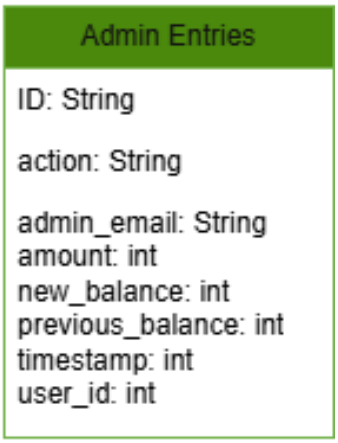
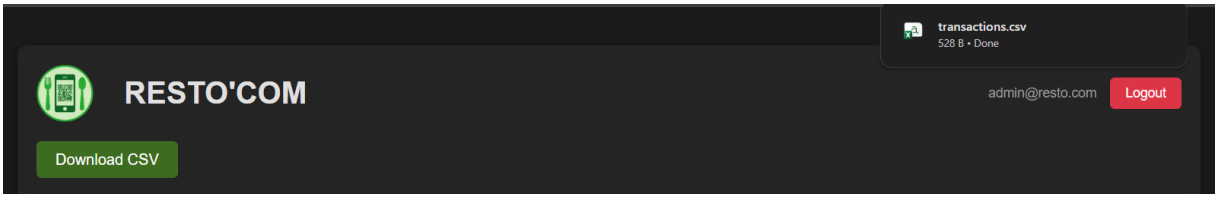


Figure 2-33 Admin entries format.

2.5.4. Data Export and Reporting:

The system provides CSV export functionality, allowing administrators to download complete transaction histories for external analysis or record-keeping. The export maintains the same data structure as the dashboard view, including sequential ticket numbering and formatted timestamps.



	A	B	C	D	E
1	Transaction ID	Student ID	Amount	Timestamp	
2	RESTO_ticket_000	1000	0.2	1/1/2026 13:11	
3	RESTO_ticket_001	1000	0.2	1/1/2026 13:11	
4	RESTO_ticket_002	1000	0.2	1/1/2026 13:09	
5	RESTO_ticket_003	1000	0.2	25/12/2025 14:29	
6	RESTO_ticket_004	1000	0.2	25/12/2025 14:26	
7	RESTO_ticket_005	1003	2	25/12/2025 14:24	
8	RESTO_ticket_006	1000	0.2	15/11/2025 22:37	
9	RESTO_ticket_007	1000	0.2	15/11/2025 22:12	
10	RESTO_ticket_008	1000	0.2	15/11/2025 22:08	
11	RESTO_ticket_009	1000	0.2	15/11/2025 21:28	
12	RESTO_ticket_010	1000	0.2	3/10/2025 20:30	

Figure 2-34 Saving logs into a CSV file.

2.6. ESP Firmware:

The ESP32 microcontroller serves as the hardware interface for the QR code payment terminal, establishing bidirectional communication with the Python backend through **TCP/IP** networking. The ESP32 is configured to broadcast its presence on the local network using **mDNS (esp32.local),** allowing the Python application to automatically discover and connect to the device on port **5050**.

The ESP32 was coded using Arduino IDE, it is an integrated development environment used for programming microcontrollers such as the ESP32. It provides a simple and efficient platform for writing, compiling and uploading firmware using C/C++. In this project, Arduino IDE was used to develop and test the ESP firmware, enabling reliable hardware control and smooth communication with the rest of the system. The code down below shows a snippet of the ESP's script that enables it to establish the connection and initialize the different screens.

```
const char* ssid = "*******";
const char* password = "********";
WiFiServer server(5050);
WiFiClient client;
enum ScreenState {
  BLACK,
  MENU,
  WAIT_QR,
  NUMPAD,
  SUCCESS,
  FAILED
};
ScreenState currentScreen = MENU;
// Forward declare functions
void setScreen(ScreenState newScreen);
void drawMenu();
void drawWaitQR();
void drawNumpad();
void drawSuccess();
void drawFailed();
void drawBlack();
void setScreen(ScreenState newScreen) {
  currentScreen = newScreen;
  switch (currentScreen) {
    case MENU:    drawMenu(); break;
    case WAIT_QR: drawWaitQR(); break;
    case NUMPAD:  drawNumpad(); break;
    case SUCCESS:  drawSuccess(); break;
    case FAILED:  drawFailed(); break;
    case BLACK: drawBlack(); break;
  }
}
```

```
void setup() {
  Serial.begin(115200);
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(TFT_BLACK);
    // WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  // mDNS
  MDNS.begin("esp32");
  // TCP server
  server.begin();
```

The communication protocol implements a simple command-based system where the ESP32 sends "QR_READ" signals to trigger camera activation and QR code scanning processes. The Python backend maintains a persistent TCP connection with the ESP32, handling automatic reconnection in case of network interruptions. Upon receiving QR scan requests, the system activates the camera module, processes the scanned QR codes through cryptographic verification and sends status responses back to the ESP32. These responses include "CAM_ON" to indicate camera activation, "PAY_SUCCESS" for successful transactions, "PAY_FAILED" for insufficient balance or verification errors and "QR_SCAN_CANCELLED" when operations are terminated.

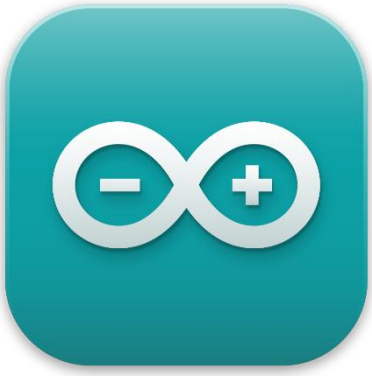


Figure 2-35 Arduino IDE logo.

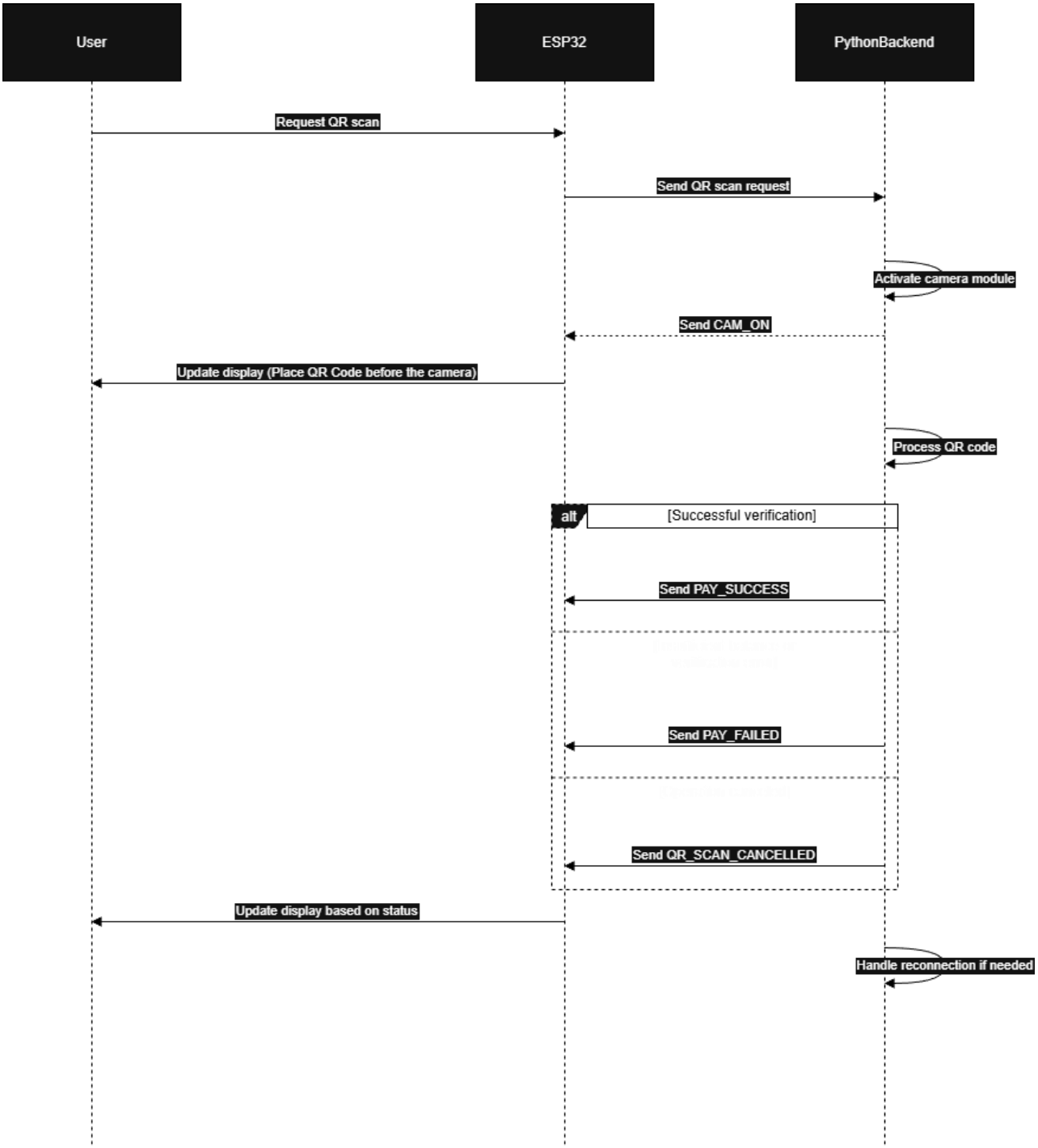


Figure 2-36 ESP32 & Python script communication UML Diagram.

To design and draw the visual outputs on the TFT display we used Lopaka, it is a graphical interface design tool specialized for small OLED and TFT displays used in embedded systems. It allows the visual creation of screen layouts and UI elements, which can then be exported as code compatible with microcontroller projects.

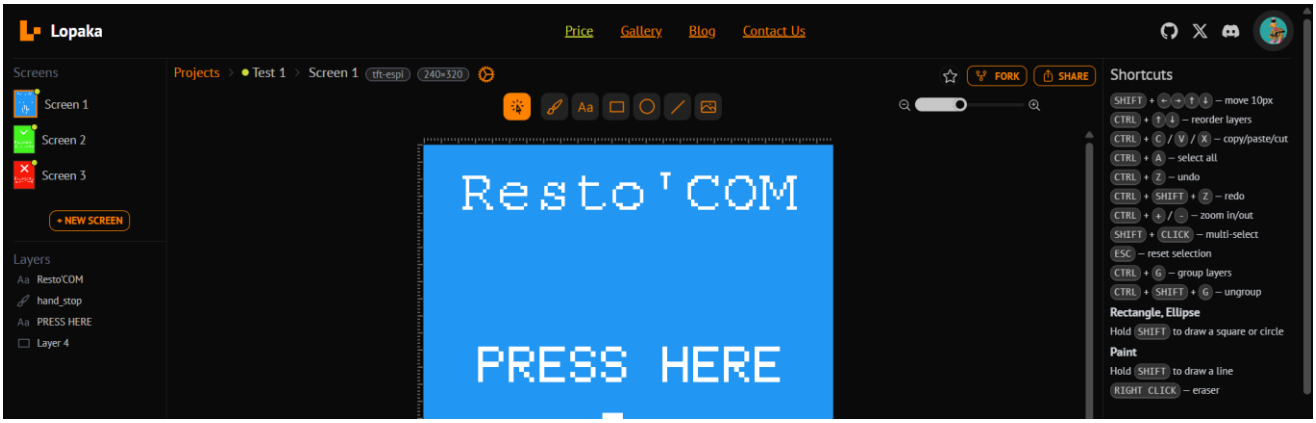


Figure 2-37 Creating screens using Lopaka.

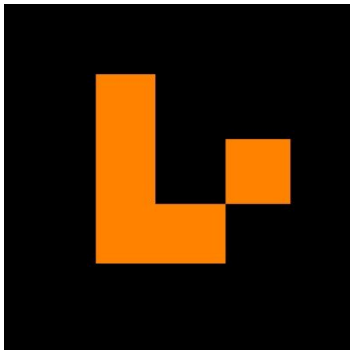


Figure 2-38 Lopaka logo.

The figure down below showcases the different screens that will be displayed.

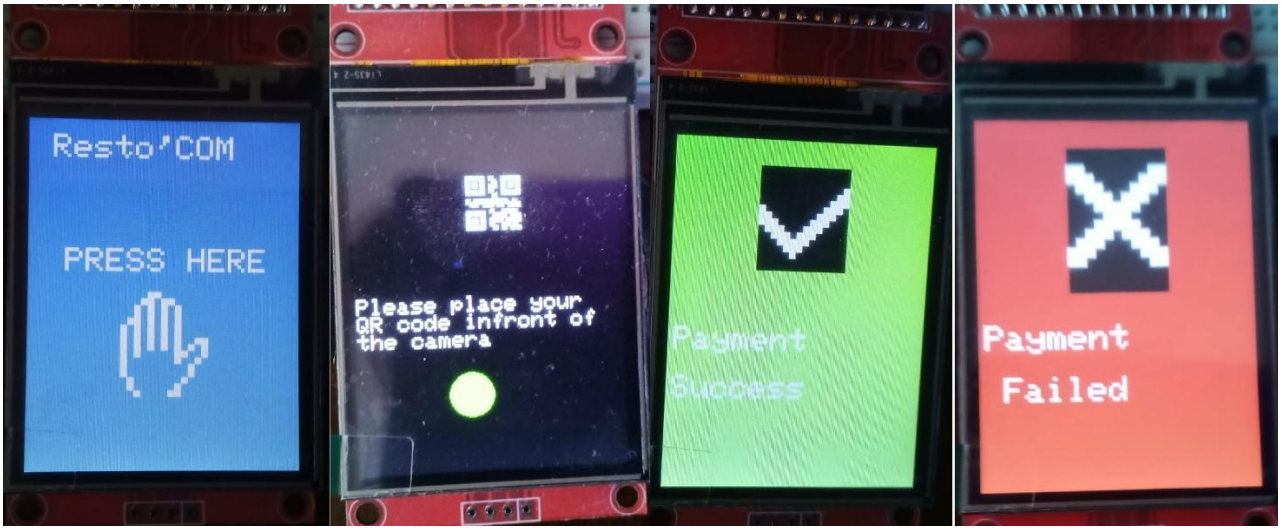


Figure 2-39 Different screens displayed.

2.7. Camera Detection Python script:

The Python-based QR processing system serves as the core transaction engine, integrating computer vision, cryptographic security and Firebase database operations to handle real-time payments. The system utilizes **OpenCV** for QR code detection and decoding, maintaining a continuous camera feed in a background thread to ensure rapid response times when scanning requests are received.

The payment processing logic implements role-based pricing with automatic balance verification, charging **0.2 TND for students and 2.0 TND for teachers** while ensuring sufficient account funds before transaction completion. Upon successful verification, the system performs atomic database operations through Firebase, updating user balances and creating transaction records with timestamps and user classification data. The system includes comprehensive error handling for various failure scenarios including insufficient balance, invalid QR codes, expired tokens, network connectivity issues and database operation failures.

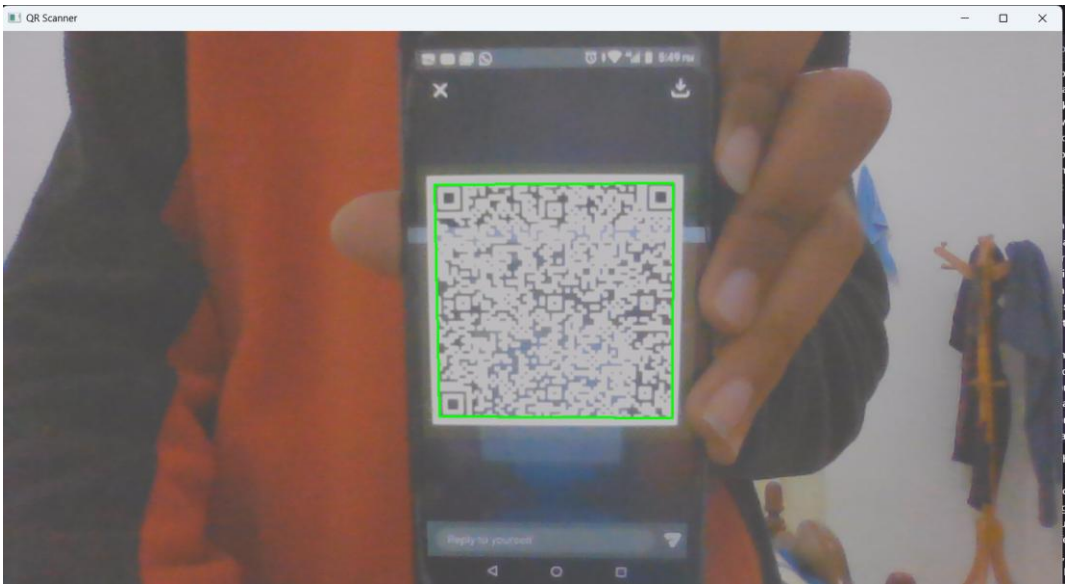


Figure 2-40 QR code detection.

This is what the script outputted when scanning a test QR code:

```
Firebase initialized successfully
Global camera started - always running
Found ESP32 at: 192.168.137.160
Connected to ESP32
From ESP32: ESP32_READY
From ESP32: QR_READ
Sent to ESP32: CAM_ON
=== SCANNED DATA DEBUG ===
RAW QR DATA:
'{"payload":{"exp":1763244036,"id":"gAAAAABpGPfIcGpTv_Wmf
_XFM-3qfJFu3a2OsbSHyP972HbpGyVZ-7iAUXXAOFMPAteyyzr-
```

```
vL2sIsBzG3BNwlbbYcTQCB1fVg==","n":"34e38c31b2d89a04","t":
false,"v":1},"signature":"UnZcvmQHmsmMVO1oa9Kw--
mz9QTulaPpxBW2uCmeJNo"}'
VERIFICATION FAILED: Expired
Sent to ESP32: PAY_FAILED
From ESP32: received: PAY_FAILED
```

In this test the field of the expiry date encrypted within the QR code payload has expired, that is why it shows that the verification failed with the message “Expired”.

The code down below showcases a snippet of the QR code detection algorithm.

```
def qr_scanner():
    global current_frame
    detector = cv2.QRCodeDetector()
    send_to_esp32("CAM_ON")
    while True:
        if current_frame is None:
            time.sleep(0.1)
            continue
        frame=current_frame.copy()
        data, bbox, _ = detector.detectAndDecode(frame)
        if bbox is not None:
            # Draw a box around the QR code
            bbox = bbox.astype(int)  # convert float to
int
            for i in range(len(bbox[0])):
                pt1 = tuple(bbox[0][i])
                pt2 = tuple(bbox[0][(i+1) %
len(bbox[0])])
                cv2.line(frame, pt1, pt2, (0, 255, 0), 2)
            if data:
                print("=== SCANNED DATA DEBUG ===")
                print(f"RAW QR DATA: '{data}'")  # Debug
line
                ctime_str = time.ctime(time.time())
                status, msg = verify_and_decrypt(data)
```

2.8. Communication & Security:

2.8.1. Communication Flow:

The figure down below represents the communication/network architecture between all the components of the project:

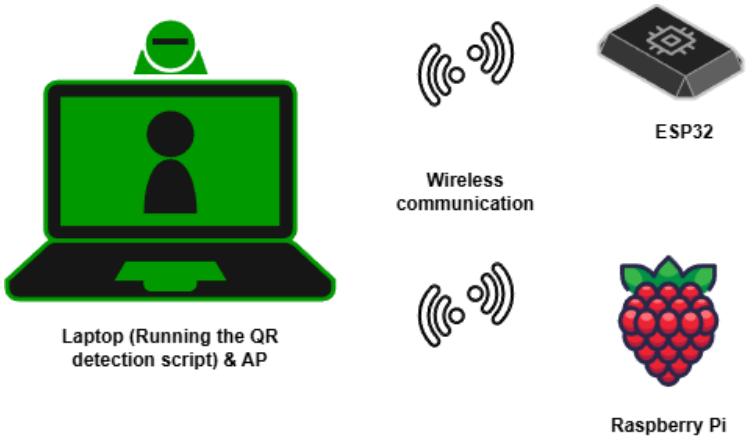


Figure 2-41 Communication process.

To ensure flexibility, all the components communicate wirelessly. With only a power cable connected to the Raspberry Pi, the whole system comes to life. There are a few configuration commands necessary to establish this connection.

Figure 2-42 Components connected to the laptop's hotspot.

As mentioned before the ESP32 utilizes the mDNS protocol in order to establish the connection.

```
// WiFi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
}
// mDNS
MDNS.begin("esp32");
// TCP server
server.begin();
```

For the Raspberry Pi, we added the laptop's network using these lines:

```
sudo raspi-config
1 System Options
 → S1 Wireless LAN
```

```
   → Enter SSID: HOTSPOT_NAME
   → Enter Passphrase: ******
Finish → Reboot (Yes)
```

2.8.2. Security Verification Process:

The QR code system implements a multi-layered security approach to ensure payment authenticity and prevent fraud. When a user requests a QR code through the mobile app, the system generates a secure token containing the user's encrypted ID, expiration timestamp, unique nonce and user type (student/teacher). The user ID is encrypted using industry-standard encryption to protect sensitive information, while the entire payload is digitally signed using a secret key shared between the generation and verification systems.

```
def generate_qr():
    try:
        student_id = request.args.get("student_id")
        if not student_id:
            return jsonify({"error": "student_id
required"}), 400
        # Your existing function logic here
        ref = db.reference(f"users/{student_id}")
        student = ref.get()
        if not student:
            return jsonify({"error": "User not found"}),
404
        expiry = int(time.time()) + 60
        nonce = secrets.token_hex(8)
        enc_id =
f.encrypt(str(student_id).encode()).decode()
        payload = {
            "v": 1,
            "id": enc_id,
            "exp": expiry,
            "n": nonce,
            "t": student["isteacher"]
        }
        payload_bytes = json.dumps(payload,
separators=(",", ":"), sort_keys=True).encode()
        signature = hmac.new(SECRET_KEY, payload_bytes,
hashlib.sha256).digest()
        sig_b64 = b64url(signature)
        response_data = {
            "payload": payload,
            "signature": sig_b64
        }
```

Each QR code has a built-in expiration time of 60 seconds, making it impossible to reuse old codes for unauthorized transactions. The system also includes a random nonce (number used once) to prevent replay attacks, ensuring that even if someone captures a QR code, they cannot duplicate it for fraudulent use. When the payment terminal scans a QR

code, it performs multiple verification steps: checking the digital signature to confirm authenticity, validating the expiration time, decrypting the user ID and verifying that the user exists in the database.

```
        # 2. Parse JSON with error handling
        try:
            qr_data = json.loads(qr_json_string)
        except json.JSONDecodeError as e:
            return False, f"Invalid JSON format: {str(e)}"

        # 3. Check required fields exist
        if "payload" not in qr_data:
            return False, "Missing 'payload' field"
        if "signature" not in qr_data:
            return False, "Missing 'signature' field"

        payload = qr_data["payload"]
        signature = qr_data["signature"]

        # 4. Check payload structure
        if not isinstance(payload, dict):
            return False, "Payload must be a dictionary"
        if "id" not in payload:
            return False, "Missing field in payload"
        if "exp" not in payload:
            return False, "Missing field in payload"
        if "n" not in payload:
            return False, "Missing field in payload"
        if "t" not in payload:
            return False, "Missing field in payload"
```

Figure 2-43 A snippet of the verification code.

2.8.3. API Authentication Implementation:

To enhance the security of the QR code generation system, authentication was implemented on the Vercel REST API endpoint. The mobile application now automatically includes Firebase ID tokens in the Authorization header of all API requests using a custom HTTP client wrapper. On the server side, the API validates these JWT tokens by verifying their issuer, expiration and extracting the authenticated user's ID. The system then cross-references this user ID with the requested student data in the Firebase Realtime Database to ensure users can only generate QR codes for their own accounts. This implementation uses the PyJWT library for token decoding and maintains the existing QR code format and verification process, ensuring backward compatibility with the scanning system while preventing unauthorized access to the API endpoints.

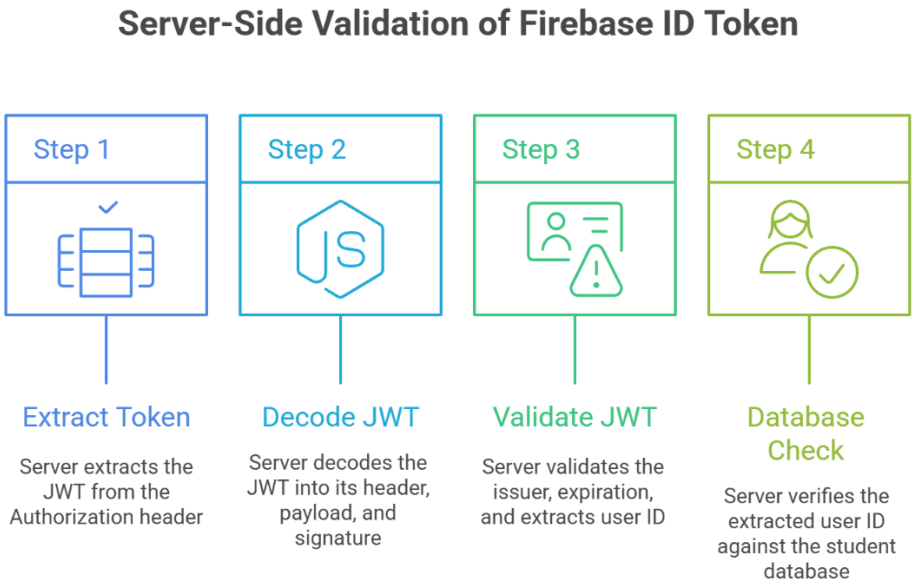


Figure 2-44 Server-Side Validation of Firebase ID Token.

Here is a code snippet of the verification function used:

```
def verify_token(request):
    auth_header = request.headers.get('Authorization')
    if not auth_header or not
auth_header.startswith('Bearer '):
        return None, "Authorization required"

    token = auth_header.split('Bearer ')[1]
    try:
        decoded = jwt.decode(token,
options={"verify_signature": False})
        # Basic checks
        if 'securetoken.google.com' not in
decoded.get('iss', ''):
            return None, "Invalid token"
        if decoded.get('exp', 0) < time.time():
            return None, "Token expired"

        return decoded.get('sub'), None  # Return user ID
    except:
        return None, "Invalid token"
```

We also tested the API with Postman to verify the functionalities; the figure down below showcases an attempt to access the API without the proper authentication parameters.

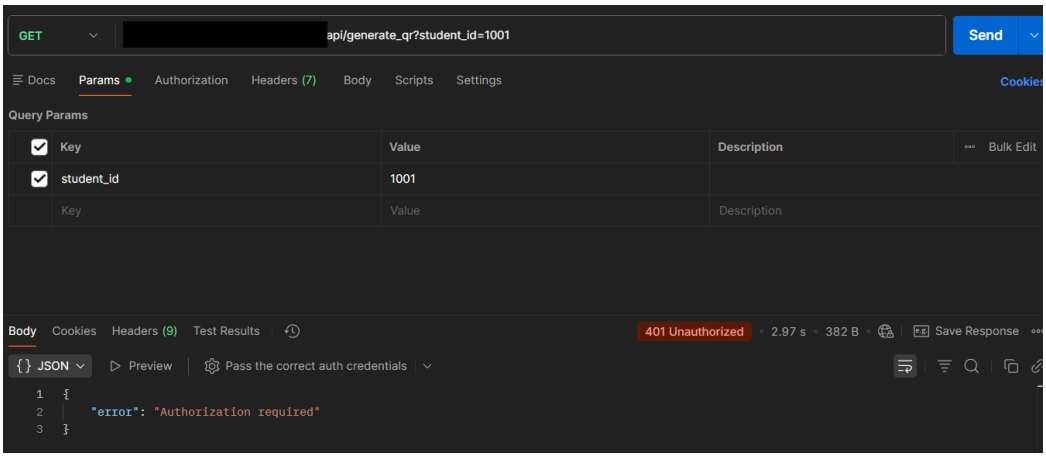


Figure 2-45 Testing the API authentication mechanism using Postman.

2.8.4. Firebase database rules:

Firebase Realtime Database security rules were configured to enforce strict access control and protect sensitive application data. In the current implementation, both read and write operations are permitted only for authenticated users by verifying the presence of a valid authentication context (auth != null). This mechanism prevents unauthenticated access and ensures that all database interactions originate from verified identities managed by Firebase Authentication. The rule structure follows Firebase's declarative security model, enabling centralized and enforceable security policies at the database level. This foundation allows for future enhancements such as role-based authorization, user-scoped data access, validation of data structures and conditional write constraints thereby supporting a progressive hardening of the system's security as the application evolves.

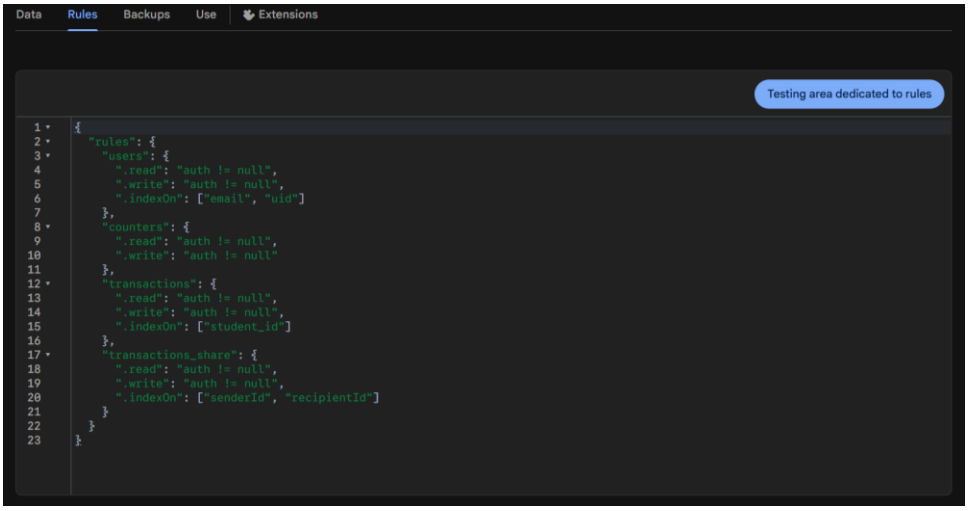


Figure 2-46 Firebase Realtime Database Rules.

2.9. 3D Design:

2.9.1. Tools used:

Tinkercad: Tinkercad is a web-based 3D modeling tool that allows rapid creation of three-dimensional prototypes. Its intuitive interface and simple geometric operations make it ideal for quickly visualizing ideas, verifying component placement and simulating the spatial organization of a system before moving to precise technical design.

Figure 2-47 Tinkercad logo.

TechCAD: TechCAD is a CAD software specialized in producing precise 2D technical drawings. It enables accurate dimensioning, geometric constraints and preparation of files for manufacturing processes such as laser cutting, ensuring that the physical realization matches the design specifications.

Figure 2-48 TechCAD logo.

2.9.2. Implementation:

In order to design the physical enclosure of the system, a 3D modeling process was adopted. Tinkercad was used to create a first 3D prototype of the device, allowing us to visualize the overall shape, component placement and dimensions of the system. This initial prototype played an important role in better understanding how the different hardware elements (ESP module, display and power connections) would be integrated and interact within the enclosure.

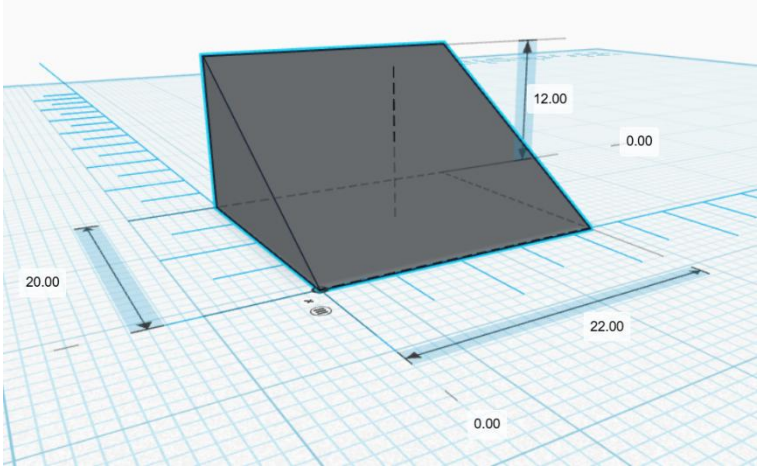


Figure 2-49 Modeling in Tinkercad.

Creating a first prototype is essential, as it helps explain and validate the functioning of the project before moving to physical fabrication. After validating the design concept, **TechCAD** was used to produce precise 2D sketches of the prototype, including exact dimensions, in order to prepare the model for **laser cutting**. This step ensured accuracy, reduced material waste and facilitated the manufacturing of a functional and well-adapted enclosure. The full design can be found in the Appendix section.

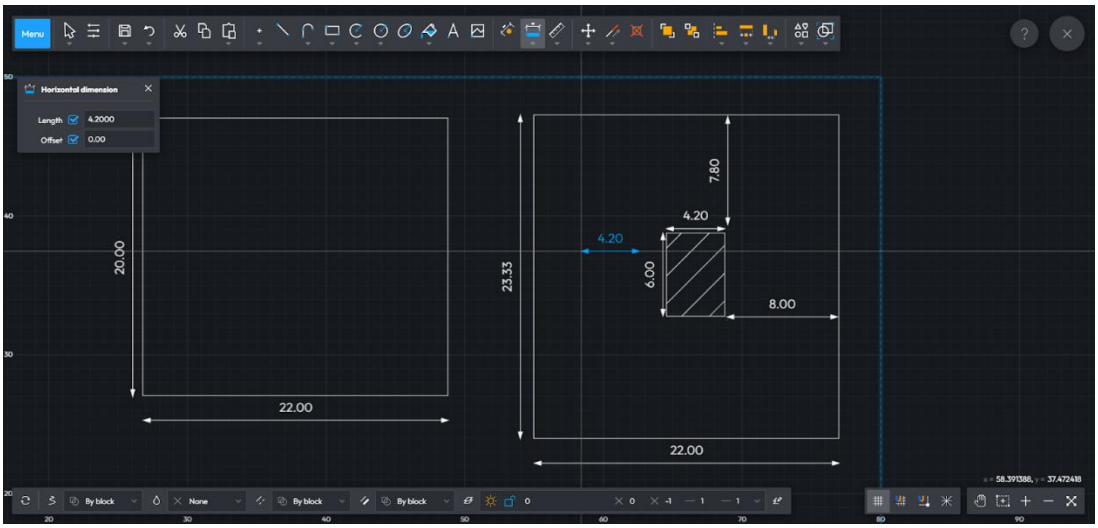


Figure 2-50 Drawing the 2D sketch.

2.9.3. Assembling the prototype:

With everything fully sketched out and measured, the schematics were handed to a plexiglass warehouse to be professionally cut. The figure down below showcases the parts after successfully being cut.

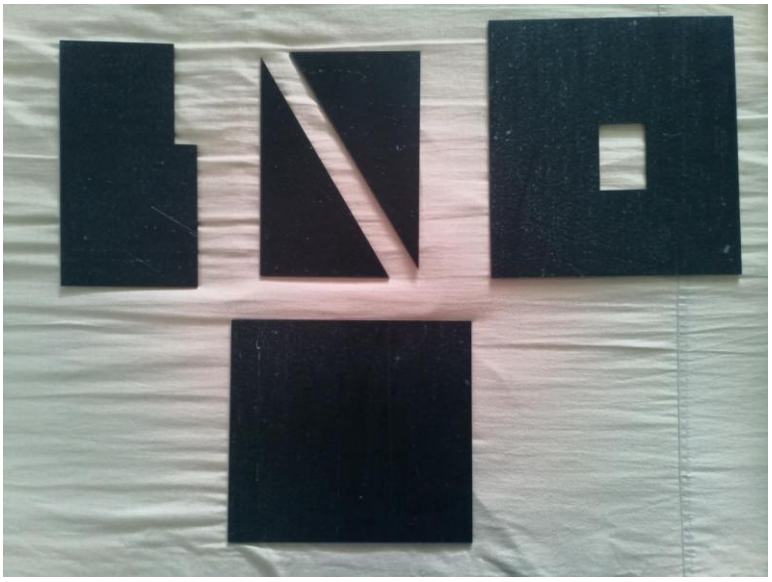


Figure 2-51 Prototype parts dissembled.

After retrieving the parts, the assembly of the first prototype started. We commenced by drilling holes precisely in order to hold the shell together firmly as well as making sure that the wires and the electronic parts don't interfere with each other. The progress is highlighted by these figures:

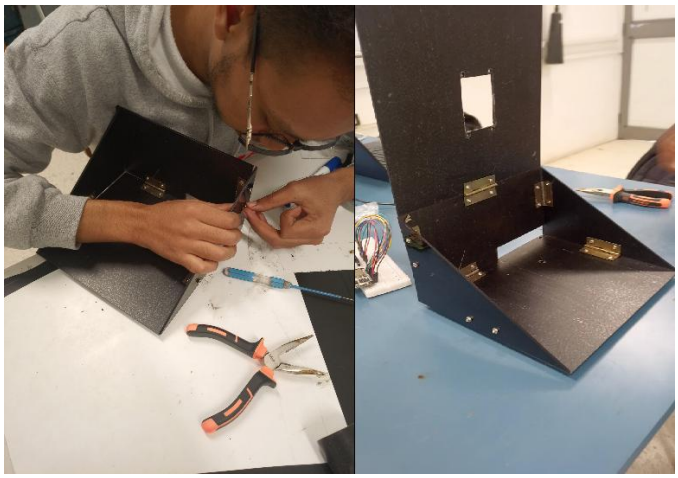


Figure 2-52 Assembling the prototype 1.

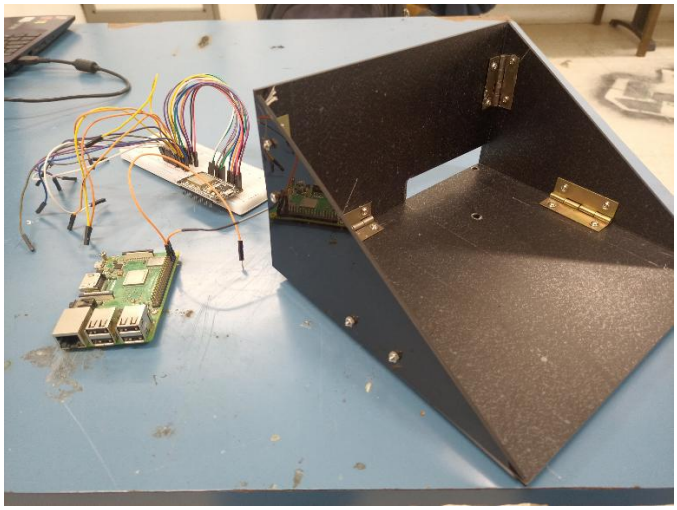


Figure 2-53 Assembling the prototype 2.

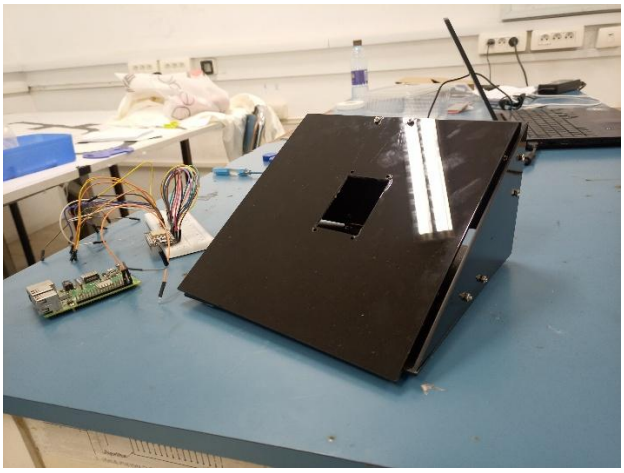


Figure 2-54 Assembling the prototype 3.

After the assembly is done, we tested the functionality and the connectivity and everything worked as intended. The figures down below showcase the fully built project from the inside and the outside.

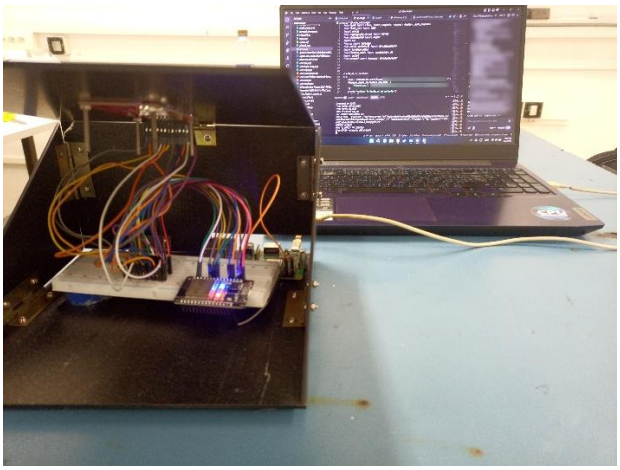


Figure 2-55 Testing the prototype 1.

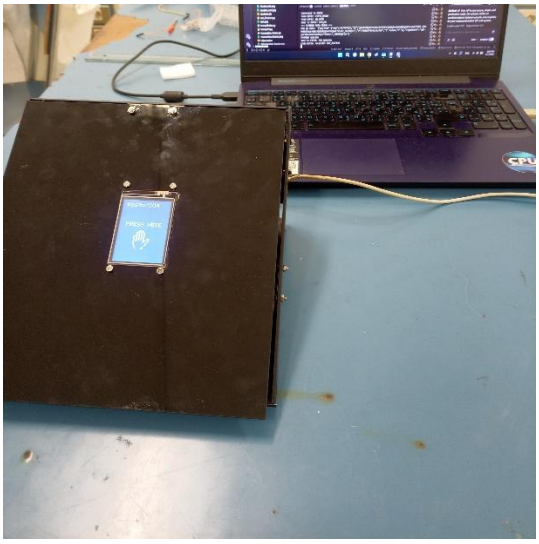


Figure 2-56 Testing the prototype 2.

2.10. Conclusion:

In conclusion, this realization phase allowed the complete validation of the proposed system by translating the theoretical design into a functional prototype. The full process, from user authentication and QR code generation to payment verification, was successfully implemented through a clear separation of roles between the different hardware and software components. Appropriate communication protocols were selected to ensure reliable data exchange between the ESP32, the Raspberry Pi and the user interface, while the TFT display provided intuitive and responsive interaction. In parallel, the 3D modeling and fabrication of the enclosure enabled the physical integration of all components, resulting in a coherent and practical first prototype. This chapter demonstrates the feasibility of the solution and establishes a solid foundation for further optimization, performance improvements and future deployment of the system.

General Conclusion:

This mini-project was conducted within the ICT program at **ISETCOM** with the aim of addressing the limitations of the traditional paper-based payment system used in the university cafeteria. Through the analysis of the existing situation, several issues were identified, including inefficiency, lack of transaction traceability, susceptibility to fraud, human errors and service congestion during peak hours. These constraints clearly demonstrated the need for a modern and automated digital solution.

To meet these needs, we designed and implemented an intelligent cafeteria payment system based on **IoT technologies, mobile applications and cloud services**. The proposed solution integrates a mobile application for student authentication and QR code generation, an ESP32-based terminal with a TFT interface, a camera-based verification mechanism and a Raspberry Pi server responsible for transaction validation and data storage. This architecture enables fast, secure and reliable meal payments while ensuring fair access for authorized students from ISETCOM and SUP'COM.

The developed system successfully replaces physical meal tickets with a **ticketless, traceable and automated payment process**. Testing results confirm that transactions are completed in a very short time, reducing queues and minimizing human intervention. In addition, the system improves administrative control by providing accurate transaction history, consumption statistics and better resource management, thereby reducing errors and food misuse.

From an educational and technical perspective, this project represents a significant contribution by combining multiple domains studied throughout the ICT curriculum. It allowed us to strengthen our skills in **embedded systems, IoT communication, networking, backend development and mobile application concepts**, as well as system integration and testing. The project also enhanced our teamwork, problem-solving and project management abilities.

Although the proposed solution meets its main objectives, several improvements can be considered for future work. These include the integration of **NFC technology**, deployment of a **fully cloud-based backend**, development of a **web-based administrative dashboard** and extension of the system to support multiple cafeterias or university sites.

In conclusion, this mini-project demonstrates the effectiveness of digital and IoT-based solutions in modernizing university services. It provides a practical, scalable and secure alternative to traditional payment methods and reflects the technical competencies acquired during our academic training at ISETCOM.

Webography:

[1] "ESP32 Wiki," 03 January 2026. [Online]. Available: https://en.wikipedia.org/wiki/ESP32.

[2] "Arduino IDE," 03 Januray 2026. [Online]. Available: https://www.arduino.cc/en/software.

[3] "Arduino ESP32 Documentation," 03 January 2026. [Online]. Available: https://docs.espressif.com/projects/arduino-esp32/en/latest/.

[4] "TFT LCD Displays (Arduino & ESP32)," 03 January 2026. [Online]. Available: https://doc-tft-espi.readthedocs.io/.

[5] "Raspberry Pi," 03 January 2026. [Online]. Available: https://www.raspberrypi.com.

[6] "Flutter," 03 January 2026. [Online]. Available: https://flutter.dev.

[7] "Firebase for Flutter," 03 January 2026. [Online]. Available: https://firebase.google.com/docs/flutter.

[8] "FlutterFire Plugins (Firebase + Flutter)," 03 January 2026. [Online]. Available: https://firebase.flutter.dev/docs/overview/.

[9] "Vercel," 03 January 2026. [Online]. Available: https://vercel.com/docs.

[10] "Flask Quickstart," 03 January 2026. [Online]. Available: https://flask.palletsprojects.com/en/stable/quickstart/.

[11] "REST API," 03 January 2026. [Online]. Available: https://flask.palletsprojects.com/en/stable/api/.

[12] "Build a Python web server with Flask (Raspberry Pi project)," 03 January 2026. [Online]. Available: https://projects.raspberrypi.org/en/projects/python-web-server-with-flask.

[13] "Deploying to Vercel," 03 January 2026. [Online]. Available: https://vercel.com/docs/deployments.

[14] "Tutorial — Python QR Code Generation," 03 January 2026. [Online]. Available: https://realpython.com/python-generate-qr-code/.

[15] "Vercel Frameworks Support," 03 January 2026. [Online]. Available: https://vercel.com/docs/frameworks/more-frameworks.

Appendix:

ESP32 Specifications:

Table 2 ESP32 Specifications.

Category	Specification
Microcontroller	Espressif ESP32 (ESP-WROOM-32 module)
CPU	Dual-core 32-bit Tensilica Xtensa LX6
Clock Frequency	Up to 240 MHz
Internal SRAM	~520 KB
Flash Memory	4 MB SPI Flash (typical)
Wireless Connectivity	Wi-Fi 802.11 b/g/n (2.4 GHz)
	Bluetooth v4.2 (Classic + BLE)
GPIO Pins	Up to 34 programmable GPIOs (board-dependent)
ADC	12-bit SAR ADC (multiple channels)
DAC	2 × 8-bit DAC
PWM	Up to 16 PWM channels
Touch Sensors	Capacitive touch pins supported
Communication Interfaces	UART, SPI, I²C, I²S, SDIO
Additional Interfaces	TWAI (CAN compatible), pulse counter
USB Interface	Micro-USB or USB-C (model dependent)
USB-to-Serial Chip	CP2102 / CH340 (board dependent)
Operating Voltage	3.3 V
Power Supply	USB 5 V or external VIN
Power Modes	Active, Light Sleep, Deep Sleep
Operating Temperature	−40 °C to +105 °C (module level)
Board Dimensions	~50 × 23 mm (DevKit V1 typical)
Programming Support	Arduino IDE, ESP-IDF, PlatformIO, MicroPython

Source:Espressif Systems, ESP32 Datasheet

Raspberry Pi 3 Model B+ Specifications:

Table 3 Raspberry Pi 3 Model B+ Specifications.

Category	Specification
Processor (SoC)	Broadcom BCM2837B0
CPU	Quad-core 64-bit ARM Cortex-A53
Clock Frequency	1.4 GHz
GPU	Broadcom VideoCore IV
RAM	1 GB LPDDR2 SDRAM
Wireless Connectivity	Dual-band Wi-Fi 802.11 b/g/n/ac (2.4 & 5 GHz)
	Bluetooth 4.2 / BLE
Ethernet	Gigabit Ethernet (over USB 2.0, ~300 Mbps max)
USB Ports	4 × USB 2.0
GPIO Header	40-pin GPIO header
Display Interface	HDMI (full size)
Camera Interface	CSI camera connector

Storage	microSD card slot
Audio / Video Output	HDMI, 3.5 mm audio jack
Power Supply	5 V via Micro-USB
Recommended Power	5 V / 2.5 A
Operating System	Raspberry Pi OS, Linux distributions
Networking Interfaces	Wi-Fi, Ethernet
Operating Temperature	0 °C to +50 °C
Board Dimensions	85 × 56 mm
Form Factor	Credit-card sized single-board computer

Source: Raspberry Pi Ltd., Raspberry Pi 3 Model B+ Product Brief

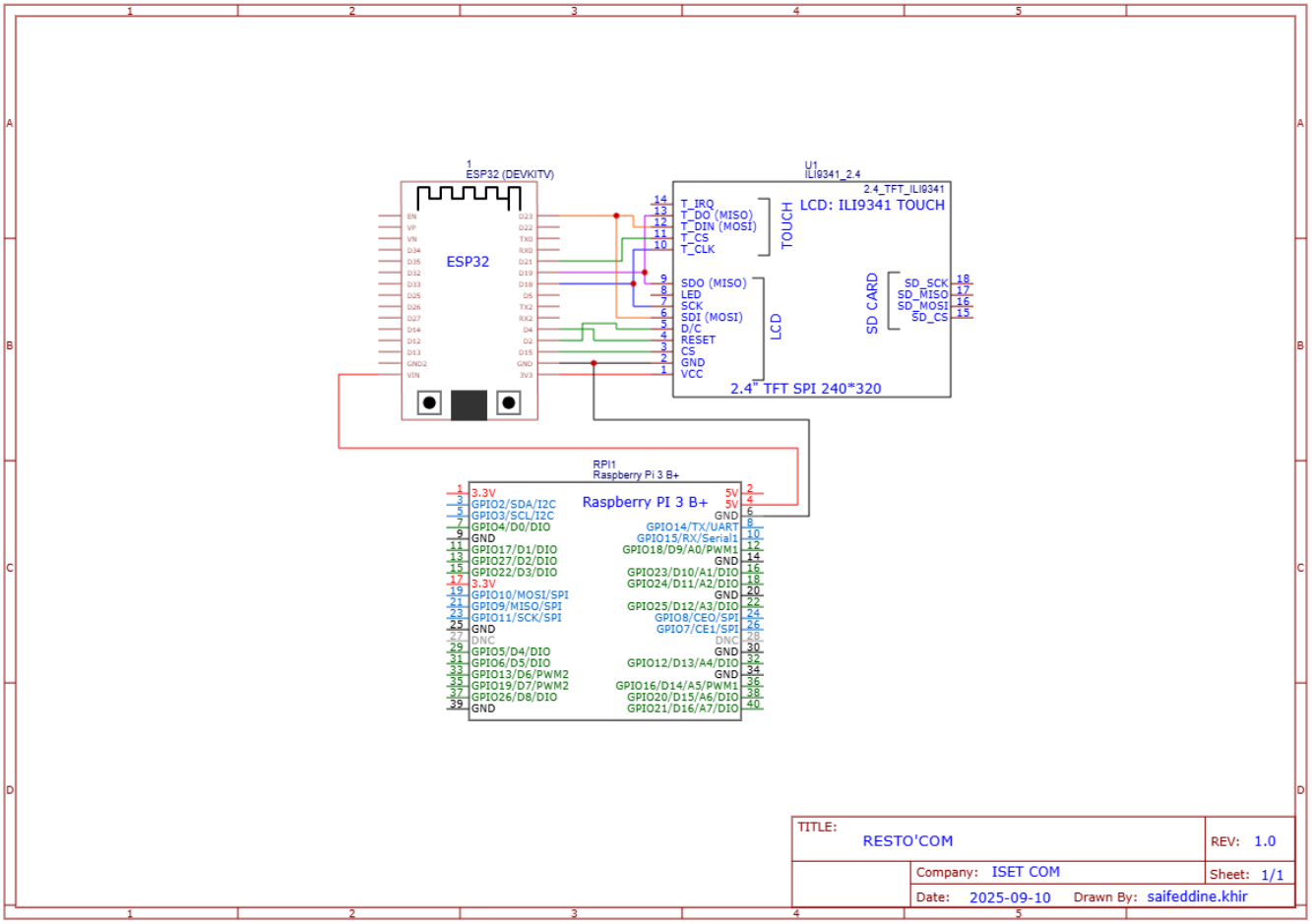
TFT LCD Touch Screen (240 × 320, ILI9341 Driver) Specifications:

Table 4 TFT LCD Touch Screen Specifications.

Category	Specification
Display Type	TFT LCD (Thin-Film Transistor)
Display Size	~2.4″ – 2.8″ (module dependent)
Resolution	240 × 320 pixels
Driver IC	ILI9341
Color Depth	16-bit (65K colors), supports 18-bit
Interface	SPI (4-wire / optional parallel on some modules)
Touch Function	Resistive touch panel
Touch Controller	XPT2046 / ADS7846 (module dependent)
Operating Voltage	3.3 V logic (some modules tolerate 5 V via onboard regulator)
Backlight	White LED (PWM controllable)
Viewing Angle	Typically ~160°
Refresh Orientation	Portrait and landscape supported
Communication Pins	CS, RESET, DC, MOSI, MISO, SCK
Touch Interface	SPI
Typical Current Consumption	~40–60 mA (with backlight on)
Mounting	PCB module with pin headers
Compatibility	Arduino, ESP32, Raspberry Pi
Common Libraries	Adafruit_ILI9341, TFT_eSPI, U8g2

Source : Ilitek, ILI9341 Datasheet

Wiring Schematics:



2D Sketch: