1.INTRODUCTION

The 'RentIt' project is a platform designed with a web interface for shops and a mobile app for users designed to streamline the process of renting various products such as tools, furniture, and electronics. The platform serves as a bridge between product owners (shops) and customers seeking to rent items for temporary use. By providing a user-friendly interface, 'RentIt' aims to simplify the rental experience, ensuring convenience for both parties. The primary objective of 'RentIt' is to create a reliable, efficient, and scalable rental platform that allows users to browse, select, and rent items effortlessly. The platform offers shops a dedicated space to manage their rental inventory, track bookings, and handle customer interactions. The system also ensures secure transactions by integrating a payment gateway for both cash on delivery and advance payments.

The platform's key features include an intuitive user interface that allows users to browse products, view details, and proceed with checkout seamlessly. Shops are provided with tools to add, update, and manage products efficiently. The system offers booking management features that enable shops to track bookings, handle early return requests, and manage customer interactions. Additionally, the platform maintains a rent history that displays past transactions with details such as rented items, dates, and payment status. To enhance security and flexibility in transactions, the system integrates a payment gateway that supports both cash on delivery and advance payments. A complaint management system is also included, allowing users to lodge complaints directly through the platform for better issue resolution.

The purpose of 'RentIt' is to address the growing demand for temporary product usage while providing a digital platform that enhances customer convenience. By utilizing technology, 'RentIt' streamlines the rental process, making it faster and more secure. The system is designed with scalability in mind, ensuring that new features and products can be added effortlessly in the future. The platform primarily focuses on tool rentals but is flexible enough to accommodate other product categories such as furniture and electronics. The project is targeted at local shops seeking a digital solution to manage their rental services efficiently. Future enhancements may include improved analytics, inventory tracking, and additional payment options to cater to evolving customer needs.

2. System Analysis

The system analysis phase of the 'RentIt' project involves a detailed study of the platform's functional and non-functional requirements. The analysis identifies the key features required to meet user expectations and outlines the technological framework necessary for efficient operation. The primary focus is on developing a scalable system that supports secure transactions, flexible booking management, and user profile customization. By examining potential risks, system performance, and data management strategies, the analysis ensures that 'RentIt' delivers a robust platform capable of handling a growing user base. This comprehensive assessment provides a clear roadmap for the system's development, ensuring that each component aligns with the project's overall objectives.

2.1 Existing System

In the current rental system, traditional methods of item rentals are inefficient and outdated. Customers usually need to visit multiple shops physically to find the desired items, resulting in time-consuming and frustrating experiences. Additionally, product availability information is often unreliable, leading to uncertainty and confusion. Payment methods are often limited to cash, creating inconvenience for customers preferring digital transactions. The absence of a centralized complaint system makes it challenging for users to report issues, further deteriorating customer satisfaction. Moreover, record-keeping for shops is usually manual, making inventory management cumbersome and prone to errors. These drawbacks highlight the need for a modernized and digital rental solution.

2.2 Proposed System

The 'RentIt' project proposes a comprehensive digital solution that overcomes the limitations of the existing system. By offering a centralized platform that connects shops and users, 'RentIt' allows customers to browse available items, view detailed descriptions, and rent products directly from the comfort of their homes. The system includes secure payment gateways supporting both cash on delivery and advance payments. Shops can efficiently manage their inventory, track bookings, and respond to customer requests using the web interface. The app for users ensures seamless navigation and improved accessibility, while the integrated complaint management system enhances user satisfaction by providing a platform for resolving issues promptly. This streamlined approach minimizes manual errors, reduces delays, and significantly improves the overall rental experience.

2.3 System Requirement Specification (SRS)

A software requirements specification (SRS) is a comprehensive description of the intended purpose and environment for software under development. The SRS fully describes what the software will do and how it will be expected to perform. An SRS minimizes the time and effort required by developers to achieve desired goals and the development cost. A good SRS defines how an application will interact with the system hardware, other programs, and human users in a wide variety of real-world simulations.

2.3.1 Hardware Specification

- Processor: Intel Core i3 or higher
- RAM: 4GB or higher
- Storage: Minimum 20GB free space
- Display: Minimum 1024x768 resolution
- Smartphone: Android 8.0 or higher for the mobile app

2.3.2 Software Specification

- Operating System: Windows 10 or higher, Android for the mobile app
- Development Tools: Flutter, Visual Studio Code
- Frontend: Flutter (Dart)
- Backend: Firebase / Node.js
- Database: Supa base
- Designing Tools: Figma
- Browser: Chrome

2.3.3 Front End

The front end of RENTIT is developed using Flutter, an open-source UI toolkit by Google that enables the development of cross-platform applications from a single codebase. Dart is used as the programming language for Flutter, ensuring fast performance and expressive UI capabilities.

Key Features of the Front-End

● Cross-Platform Compatibility: The application runs smoothly on both Android and iOS.

● Material Design UI: Provides a visually appealing and intuitive user experience.

● State Management: Efficient state management using Provider or Riverpod for seamless data handling.

● Real-Time Updates: Integrated with Firebase to fetch and display real-time event and club updates.

● Navigation & Routing: Implements a smooth navigation system with named routes for a structured UI flow.

● User Authentication: Secure login/signup functionality with role-based access. ● Responsive UI: Adaptable layouts ensure optimal display across different screen sizes.

2.3.4 Back End

The back-end is responsible for handling server-side operations, database management, authentication, and API communication. It processes user requests, retrieves, and stores data, and ensures smooth interaction between the front end and the database. A well-structured back-end is essential for ensuring data security, real-time updates, and efficient application performance.

Key Features of the Back-End

● Authentication & User Management: Secure role-based authentication for students, faculty, and admins.

● Real-Time Data Updates: Ensures that event registrations, announcements, and club memberships reflect instantly.

● Cloud-Based Storage: Provides scalability and security for managing user-generated content.

● API Communication: Seamless interaction between the Flutter front-end and Supabase via RESTful API calls

Integration with Dart

Although Supabase does not require a traditional back-end language, Dart is used for communicating with Supabase through API calls. The system handles authentication, database queries, and data retrieval using Dart's built-in HTTP and Supabase SDK. This integration ensures a secure, scalable, and efficient back-end without needing additional server-side frameworks. By leveraging Supabase and Dart, RENTIT ensures that event user authentication, and database interactions remain fast, secure, and responsive for a seamless user experience.

2.4 Feasibility Analysis

The feasibility analysis for 'RentIt' includes technical, economic, and operational considerations:

- **Technical Feasibility**

  Technical feasibility assesses whether the existing infrastructure and technology can support the development and deployment of RENTIT. The system utilizes Flutter for cross-platform compatibility, Firebase for real-time data synchronization, and PostgreSQL for secure and efficient data storage. Since these technologies are widely adopted and well-documented, implementing RENTIT is technically feasible without requiring significant additional resources.

  Key technical advantages:

  ● Cross-platform development using Flutter, reducing maintenance efforts.

  ● Cloud-based backend with Firebase ensures scalability and real-time updates

  ● Secure authentication mechanisms, including role-based access control.

  ● Minimal hardware requirements, making the system accessible to all users.

- **Economic Feasibility:**

The economic feasibility of the 'RentIt' project assesses the cost-effectiveness and financial benefits associated with its development and implementation. The project has been carefully designed to ensure that the overall development costs remain within a reasonable budget. Expenses are allocated towards essential elements such as software development tools, hosting services, and payment gateway integration. By leveraging cost-effective technologies and open-source solutions, the project minimizes unnecessary expenditures while maximizing functionality. Additionally, the platform's revenue model—based on rental service commissions and premium feature subscriptions—ensures sustainable income generation. The combination of low development costs and efficient revenue strategies makes 'RentIt' a financially viable solution that promises long-term profitability and stability.

- **Operational Feasibility**

The operational feasibility of the 'RentIt' project evaluates how effectively the system can function in its intended environment while meeting user requirements. The platform is designed to streamline rental transactions, making it easy for users to browse, book, and manage their rentals. Shop owners benefit from intuitive tools that enable efficient product management, order tracking, and complaint resolution. The user interface is developed with simplicity and clarity in mind to ensure minimal learning curves for both customers and shop owners. Additionally, the system's performance is optimized to handle multiple user interactions simultaneously, ensuring smooth navigation and fast response times. By implementing clear workflows and efficient data handling mechanisms, the 'RentIt' platform is equipped to operate reliably under real-world conditions, ensuring successful adoption and long-term usability.

- **The behavioural feasibility**

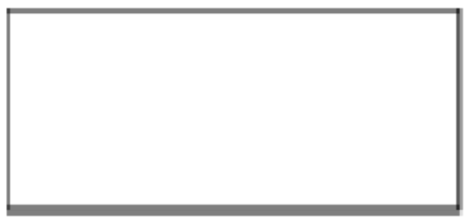
The behavioural feasibility of the 'RentIt' project evaluates how well the system aligns with user expectations, behaviours, and acceptance. The platform is designed with user engagement and satisfaction in mind, ensuring intuitive navigation, clear instructions, and minimal complexity in completing tasks. Customers are guided through straightforward processes for browsing items, booking rentals, and managing their accounts. Similarly, shop owners are provided with efficient tools to update product listings, handle orders, and track complaints. The platform’s user interface is crafted to reduce cognitive load, ensuring that users can adapt to the system easily without extensive training. By prioritizing user needs, enhancing usability, and ensuring a positive experience, the 'RentIt' project is well-positioned to gain acceptance and encourage frequent usage among its target audience

2.5 Data Flow Diagram (DFD)

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system. It differs from a flowchart as it illustrates the data flow instead of the control flow of the program. A data flow diagram can also be used to visualize data processing (structured design). Data flow diagrams were invented by Larry Constantine, the original developer of structured design, based on Martin and Estrin's model of computation known as the "data flow graph."\nData flow diagrams (DFDs) are one of the three essential perspectives of the Structured System Analysis and Design Method (SSADM). The project sponsor and end users need to be briefed and consulted throughout all stages of a system's evolution. With a data flow diagram, users can visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's data flow diagrams can be drawn up and compared with the new system's data flow diagrams to enable comparisons that help implement a more efficient system. Data flow diagrams can provide end users with a tangible idea of how their input data ultimately affects the structure of the entire system, from order to dispatch to reporting. The development of any system can be understood through a data flow diagram.\nDeveloping a data flow diagram helps identify the transaction data in the data model. There are various notations used to create data flow diagrams, defining different visual representations for processes, data stores, data flow, and external entities. The first step is to draw a data flow diagram (DFD). A DFD, also known as a "bubble chart," aims to clarify system requirements and identify major transformations that will be incorporated into system design. Therefore, it serves as the starting point of the design phase, functionally decomposing the requirements specification down to the lowest level of detail. A DFD consists of a series of bubbles joined by lines. The bubbles represent data transformations, while the lines represent data flow in the system.

BASIC DFD SYMBOLS

Square:

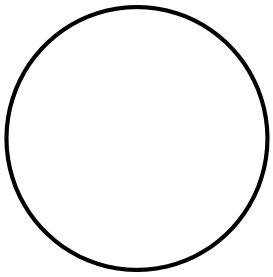


A source or sink is a person or part of an organization, which enters or receives information from the system, but is considered to be outside the context of data flow model.

Arrow:

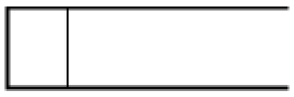
A data flow is a route, which enables packets of data to travel from one point to another. Data may flow from a source to a processor and from data store or process. An arrow lines depicts the flow, with arrowhead pointing in the direction of flow.

Circle:

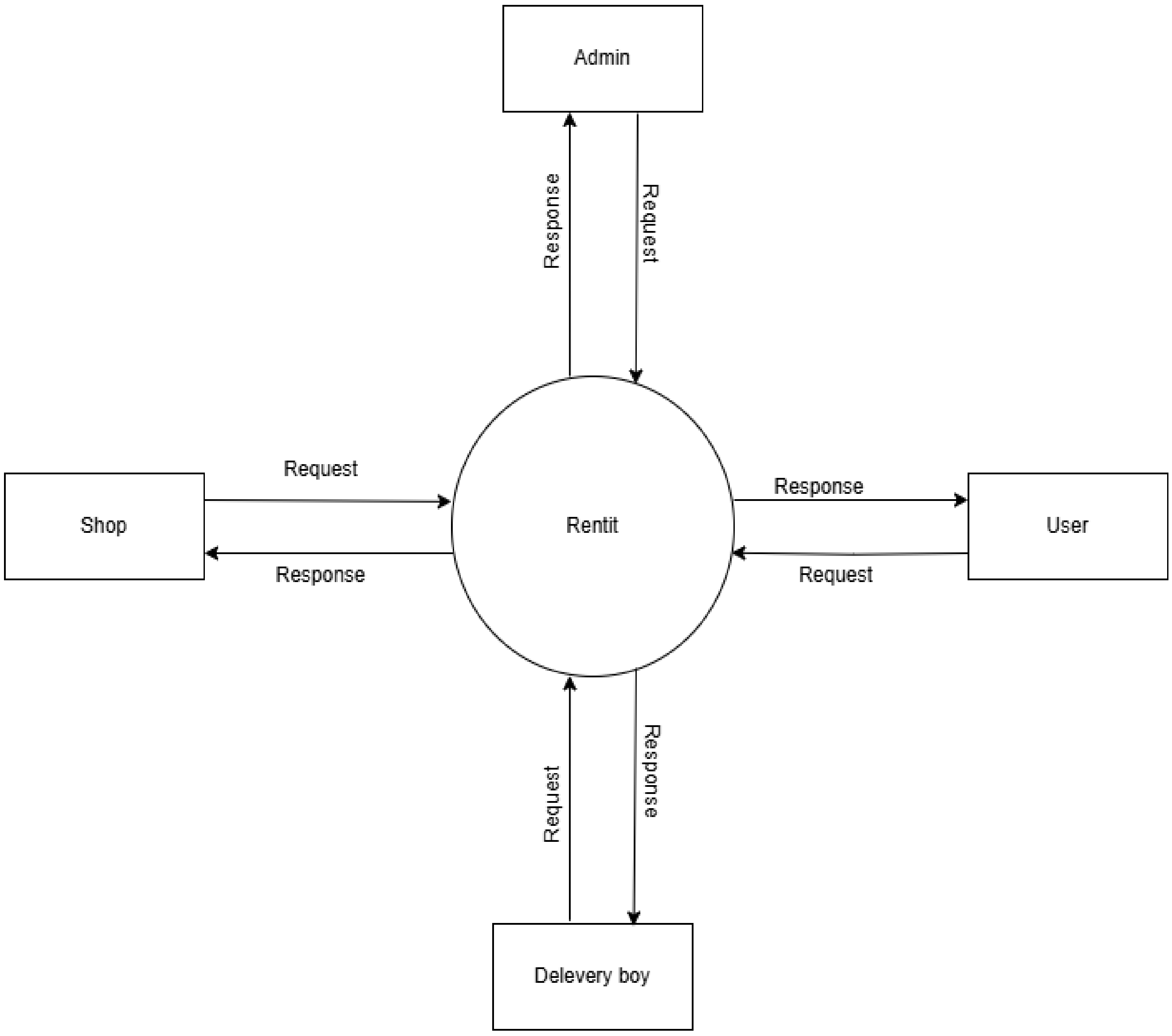


A Process represents transformation where incoming data flows are changed into outgoing data flows.

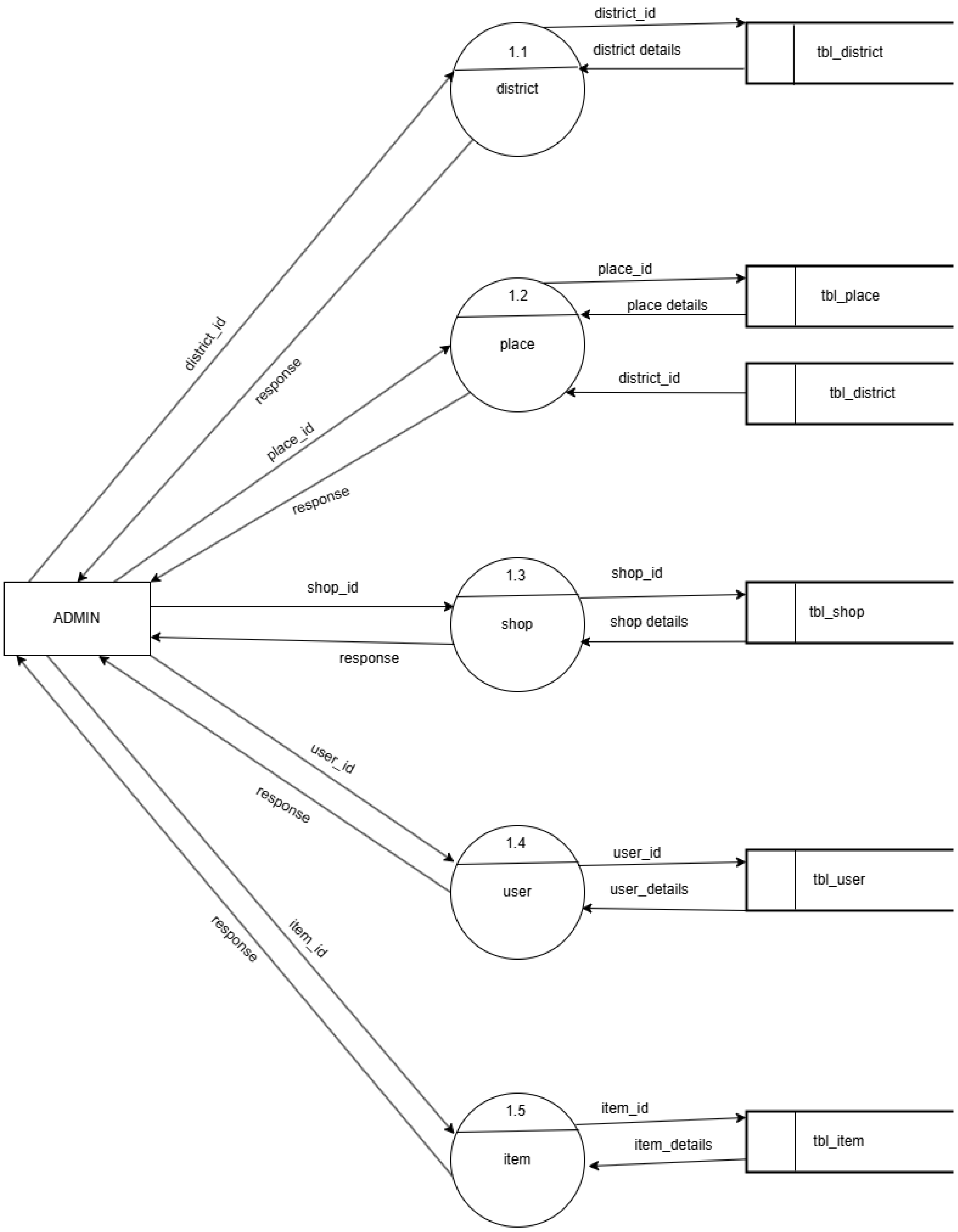
Open rectangle:

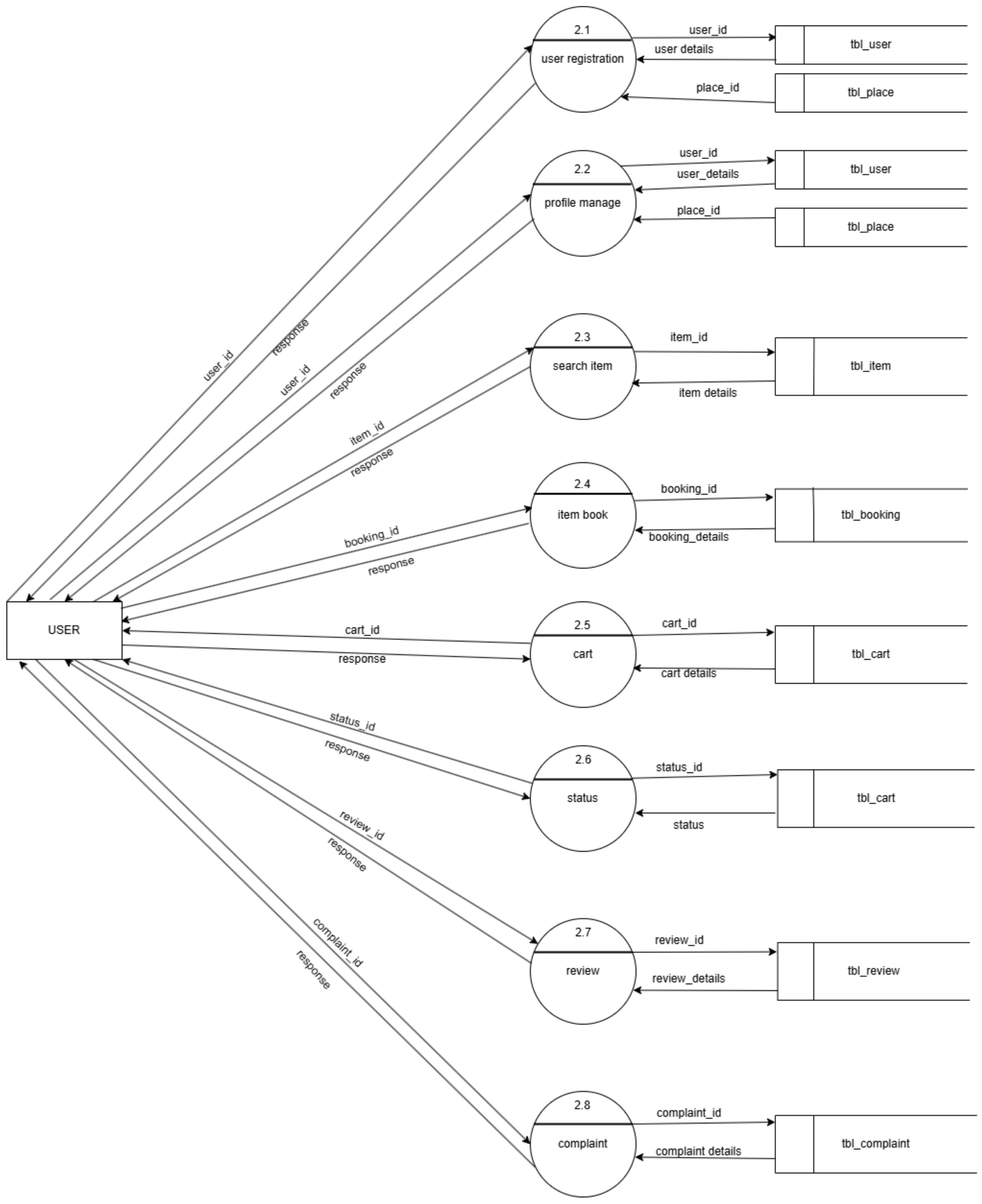


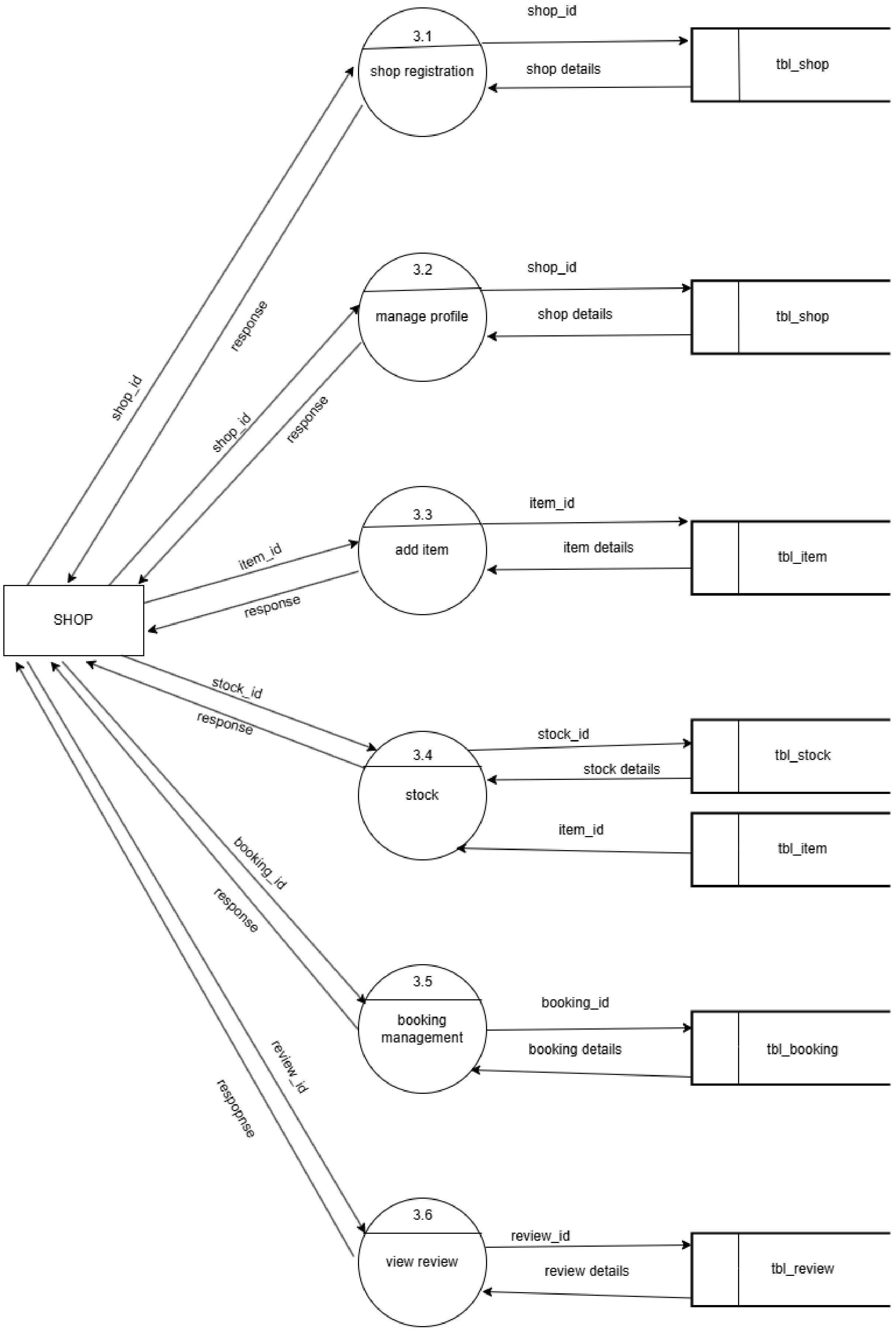
Level 0

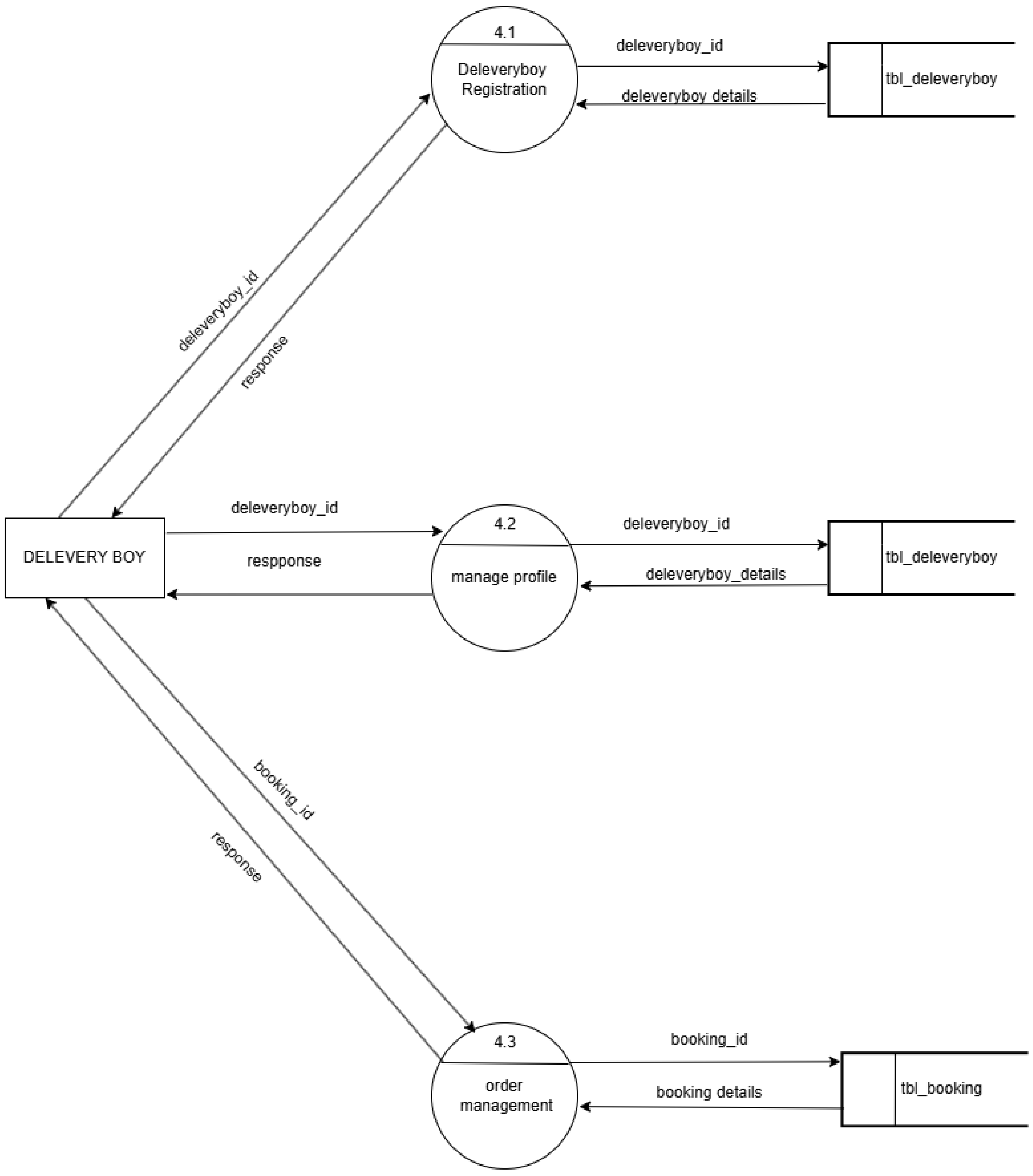

Admin
Response
Request
Shop
Request
Response
Rentit
Response
Request
User
Request
Response
Delevery boy

Level 1


1.1
district
district_id
district details
tbl_district
1.2
place
place_id
place details
district_id
tbl_place
tbl_district
1.3
shop
shop_id
shop details
tbl_shop
1.4
user
user_id
user_details
tbl_user
1.5
item
item_id
item_details
tbl_item
ADMIN
district_id
response
place_id
response
shop_id
response
user_id
response
item_id
response


USER
2.1
user registration
2.2
profile manage
2.3
search item
2.4
item book
2.5
cart
2.6
status
2.7
review
2.8
complaint
tbl_user
tbl_place
tbl_user
tbl_place
tbl_item
tbl_booking
tbl_cart
tbl_cart
tbl_review
tbl_complaint
user_id
response
user_id
response
item_id
response
booking_id
response
cart_id
response
status_id
response
review_id
response
complaint_id
response
user_id
user details
place_id
user_id
user_details
place_id
item_id
item details
booking_id
booking_details
cart_id
cart details
status_id
status
review_id
review_details
complaint_id
complaint details


shop_id
3.1
shop registration
shop details
tbl_shop
response
shop_id
3.2
manage profile
shop_id
shop details
tbl_shop
shop_id
response
3.3
add item
item_id
item details
tbl_item
item_id
response
SHOP
stock_id
response
3.4
stock
stock_id
stock details
tbl_stock
item_id
tbl_item
booking_id
response
3.5
booking management
booking_id
booking details
tbl_booking
review_id
respopnse
3.6
view review
review_id
review details
tbl_review


4.1
Deleveryboy Registration
deleveryboy_id
deleveryboy details
tbl_deleveryboy
deleveryboy_id
response
DELEVERY BOY
deleveryboy_id
respponse
4.2
manage profile
deleveryboy_id
deleveryboy_details
tbl_deleveryboy
booking_id
response
4.3
order management
booking_id
booking details
tbl_booking

3. System Design

3.1 Input Design

Input design is a crucial aspect of the 'RentIt' system, ensuring accurate data entry for optimal performance. The platform employs intuitive forms and structured layouts for both users and shops. Users are provided with clear options to enter personal details, rental preferences, and payment information. Shops are equipped with dedicated interfaces to manage product listings, booking updates, and complaint handling. The system ensures input validation to minimize errors and maintain data integrity. Fields such as phone numbers, email addresses, and dates are validated to ensure correct format and accuracy. Additionally, dropdown menus, checkboxes, and radio buttons are incorporated to improve usability and minimize errors during data entry.

3.2 Output Design

Output design focuses on presenting clear and useful information to the users. The 'RentIt' system displays detailed item descriptions, booking confirmations, payment summaries, and rent history in a user-friendly format. Shops receive detailed reports on inventory status, customer complaints, and booking records. The system's responsive design ensures that output is well-organized and easily accessible on both mobile and web platforms. Visual indicators such as icons, color coding, and alerts are utilized to improve readability and enhance user experience.

3.3 Database Design

The 'RentIt' system leverages Supabase for efficient database management. Key tables such as tbl_user, tbl_shop, tbl_booking, tbl_cart, and tbl_complaint are structured to maintain data consistency. Relationships between these tables ensure smooth data retrieval and accurate record-keeping. For instance, the tbl_booking table tracks booking details linked to both users and shops, while the tbl_cart table manages item selections and order statuses. The database design emphasizes data security, scalability, and streamlined query execution to enhance system performance. Indexing is applied to frequently queried fields like booking_id and item_id to optimize search speed and data retrieval.

TABLE

tbl_district

This table stores the details of districts

SL NO	NAME	DATATYPE	DESCRIPTION	CONSTRAINTS
1	district_id	INT (11)	Stores the district id	PRIMARY KEY
2	district_name	VARCHAR (100)	Stores the district name	NOT NULL

tbl_place

This table stores the details of place

SL NO	NAME	DATATYPE	DESCRIPTION	CONSTRAINTS
1	place_id	INT (11)	Stores the place id	PRIMARY KEY
2	place_name	VARCHAR (100)	Stores the place name	NOT NULL
3	district_id	INT (11)	Stores the place id	FOREIGN KEY

tbl_user

This table contains the details of user.

SL NO	**NAME**	DATATYPE	DESCRIPTION	CONSTRAINT
1	user_id	INT (11)	Stores the user id	PRIMARY KEY
2	user_name	VARCHAR (100)	Stores the user name	NOT NULL

3	user_email	VARCHAR (100)	Stores the user email	NOT NULL
4	user_gender	VARCHAR (100)	Stores the user gender	NOT NULL
5	user_contact	INT (11)	Stores the user contact	NOT NULL
6	user_address	VARCHAR (100)	Stores the user address	NOT NULL
7	place_id	INT (11)	Stores the place id	FOREIGN KEY
8	user_photo	VARCHAR (100)	Stores the user photo	NOT NULL
9	user_proof	VARCHAR (100)	Stores the user proof	NOT NULL
10	user_password	VARCHAR (100)	Stores theuser password	NOT NULL
11	user_vstatus	VARCHAR (100)	Stores the user verification status	NOT NULL

tbl_booking

This table contains booking details**.**

SL NO	NAME	DATATYPE	DESCRIPTION	CONSTRAINT
1	booking_id	INT (11)	used to store booking id	PRIMARY KEY
2	booking_date	VARCHAR (100)	Store the date of booking	NOT NULL
3	user_id	INT (11)	Stores the user id	FOREIGN KEY
4	booking_status	VARCHAR (100)	Store the status of booking	NOT NULL
5	booking_amount	VARCHAR (100)	Store the amount of booking	NOT NULL
6	booking_totalprice	INT (11)	Store the total price of booking	NOT NULL
7	boy_id	INT (11)	Store the id of delivery boy	FOREIGN KEY

tbl_cart

This table contains the cart details.

SL NO	NAME	DATATYPE	DESCRIPTION	CONSTRIANT
1	cart_id	INT (11)	Store the id of branch	PRIMARY KEY
2	cart_qty	INT (11)	Storesthe quantityof cart	NOT NULL
3	item_id	INT(11)	Store the id of tool	FOREIGN KEY

4	booking_id	INT(11)	Store the id of booking	FOREIGN KEY
5	cart_status	INT(11)	Store the status of cart	NOT NULL

tbl_category

This table stores the details of the category

SL NO	NAME	DATATYPE	DESCRIPTION	CONSTRAINTS
1	category_id	INT (11)	Stores the category id	PRIMARY KEY
2	category_name	VARCHAR (100)	Stores the category name	NOT NULL

tbl_subcategory

This table contains the details of subcategory.

SL NO	NAME	DATATYPE	DESCRIPTION	CONSTRAINT
1	subcategory_id	INT (11)	Used to store subcategory id	PRIMARY KEY
2	subcategory_name	VARCHAR (100)	Used to store subcategory name	NOT NULL
3	Category_id	INT (11)	Used to store category id	FOREIGN KEY

tbl_toolshop

This table contains the details of shop.

SL NO	**NAME**	DATATYPE	DESCRIPTION	CONSTRAINT
1	shop_id	INT (11)	Stores the shop id	PRIMARY KEY
2	shop_name	VARCHAR (100)	Stores the shop name	NOT NULL
3	shop_contact	INT (11)	Stores the shop contact	NOT NULL
4	shop_email	VARCHAR (100)	Stores the shop email	NOT NULL
5	shop_address	VARCHAR (100)	Stores the shop address	NOT NULL
6	place_id	INT (11)	Stores the place id	FOREIGN KEY
7	shop_logo	VARCHAR (100)	Stores the shop logo	NOT NULL
8	shop_proof	VARCHAR (100)	Stores the shop proof	NOT NULL
9	shop_password	VARCHAR (100)	Stores the shop password	NOT NULL
10	shop_vstatus	VARCHAR (100)	Stores the shop verification status	NOT NULL

tbl_item

This table contains the details of item

SL NO	NAME	DATATYPE	DESCRIPTION	CONSTRAINT
1	item_id	INT (11)	Stores the item id	PRIMARY KEY
2	item_name	VARCHAR (100)	Stores the name of item	NOT NULL
3	subcategory_id	INT (11)	Stores the subcategory id	FOREIGN KEY
4	company_id	INT (11)	Stores the company id	FOREIGN KEY
5	type_id	INT (11)	Stores the type id	FOREIGN KEY
6	item_photo	VARCHAR (100)	Stores the item photo	NOT NULL
7	item_detail	VARCHAR (100)	Stores the item detail	NOT NULL
8	item_rentprice	INT (11)	Stores the item rent price	NOT NULL
9	itemshop_id	INT (11)	Stores the item shop id	FOREIGN KEY
10	user_id	INT (11)	Stores the user id	FOREIGN KEY

tbl_stock

This table contains the details of stock.

SL NO	NAME	DATATYPE	DESCRIPTION	CONSTRAINT
1	stock_id	INT (11)	Stores stock id	PRIMARY KEY
2	stock_quantity	INT (11)	Stores stock quantity	NOT NULL

3	Stock_date	INT (11)	Stores stock date	NOT NULL
4	item_id	INT (11)	Stores stock id	FOREIGN KEY

tbl_complaint

This table contains the complaint details.

SL NO	NAME	DATA TYPE	DESCRIPTION	CONSTRAINT
1	Complaint_id	INT (11)	Stores the complaint id	PRIMARY KEY
2	Compalint_title	VARCHAR (100)	Stores the complaint title	NOT NULL
3	Compalint_content	VARCHAR (100)	Stores the complaint content	NOT NULL
4	user_id	INT (11)	Stores the user id	FOREIGN KEY
5	shop_id	INT (11)	Stores the toolshop id	FOREIGN KEY
6	Complaint_reply	VARCHAR (100)	Stores the complaint reply	NOT NULL
7	Complaint_status	VARCHAR (100)	Storesthe complaint status	NOT NULL
8	Complainttype_id	INT (11)	Stores the complaint typeid	FOREIGN KEY

tbl_deliveryboy

This table stores delivery boy details.

SL NO	**NAME**	TYPE	DESCRIPTION	CONSTRIANT
1	boy_id	INT (11)	Stores the delivery boy id	PRIMARY KEY
2	boy_name	VARCHAR (100)	Stores the delivery boy name	NOT NULL
3	boy_address	VARCHAR (100)	Used to store delivery boy address	NOT NULL
4	boy_contact	VARCHAR (10)	Used to store delivery boy contact	NOT NULL
5	boy_email	VARCHAR (100)	Used to store delivery boy email	NOT NULL
6	boy_photo	VARCHAR (100)	Stores delivery boy photo	NOT NULL
7	boy_proof	VARCHAR (100)	Stores delivery boy proof	NOT NULL
8	place_id	VARCHAR (100)	Stores the place id	NOT NULL
9	boy_password	VARCHAR (100)	Stores delivery boy password	NOT NULL

4.System Testing & Implementation

4.1 System Testing

Testing plays a vital role in the development of the 'RentIt' project, serving as a quality assurance process to validate the platform's functionality and reliability. The testing team will conduct various levels of testing, including unit testing to verify individual components, integration testing to assess interactions between modules, and user acceptance testing to evaluate the platform from the end users' perspective. Manual and automated testing techniques will be employed to catch defects early, ensure compliance with requirements, and optimize the platform's performance.

System testing is defined as the process by which one detects defects in the software. Any software development organization or team has to perform several processes, and software testing is one among them. It is the final opportunity of any programmer to detect and rectify any defects that may have appeared during the software development stage. Testing is the process of testing a program with the explicit intention of finding errors that make the program fail. In short, testing and quality assurance is a review of software products and related documentation for completion, correctness, reliability, and maintainability.

System testing is the first stage of implementation, which is aimed at ensuring that the system works accurately and efficiently before live operation commences. Testing is vital to the success of the system. System testing makes a logical assumption that all the parts of the system are correct and that the goal will be successfully achieved. A series of tests is performed for the proposed system before the proposed system is ready for user acceptance testing.

The testing steps are:

- Unit Testing
- Integration Testing
- Validation Testing
- Output Testing
- Acceptance Testing

System Testing provides the file assurance that software once validated must be combined with all other system elements. System testing verifies whether all elements have been combined properly, and that overall system function and performance is achieved. FA the integration of modules, the validation test was carried out over the system

It was that all the modules worked well together and met the overall system function and performance. Unit Testing Unit testing is carried out screen-wise, with each screen being identified as an object. Attention is diverted to individual modules, independently to one another to locate errors. This has enabled the detection of errors in coding and logic. Various test cases are prepared. For each module, these test cases are implemented, and it is checked whether the module is executed as per the requirements and outputs the desired result. In this test each service input and output parameters are checked.

Unit Testing

● The module interface was tested to ensure that information properly flows into and out of the program under the test.

● Boundary condition was tested to ensure that the module operates properly at boundaries established to limit or restrict processing.

● All independent paths through the control structures were executed to ensure that all statements in the modules had been executed at least once.

● Error handling paths were also tested.

Integration Testing

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. Unit-tested modules were taken and a single program structure was built that has been dictated by the design. Incremental integration has been adopted here. The modules are tested separately for accuracy and modules are integrated using bottom-up integration i.e., by integrating from moving from the bottom to the top the system is checked and errors found during integration are rectified.

In this testing individual modules were combined and the module-wise Shifting was verified to be right. The entire software was developed and tested in small segments, where errors were easy to locate and rectify. The program builds (groups of modules) were constructed corresponding to the successful testing of user interaction, data manipulation analysis, display processing, and database management.

Validation Testing

Validation testing is done to ensure complete assembly of the error-free software. Validation can be termed successful only if it functions in a manner. Reasonably expected by the student under validation is alpha and beta testing. The student-side validation is done in this testing phase. It is checked whether the data passed to each student is valid or not. Entering incorrect values does validation testing and it is checked whether the errors are being considered. Incorrect values are to be discarded. The errors are corrected. In “FlickPicks" verifications are done correctly. So, there is no chance for users to enter incorrect values. It will give error messages by using different validations. The validation testing is done very clearly and it is error-free.

Output Testing

After performing the validation testing the next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in a specific format. The output format on the screen was found to be correct as the format was designed in the system design phase according to the user’s needs. For the hard copy also, the output comes out as specified requirement by the user. Hence output testing does not result in any correction in the system output. This project is developed based on the user's choice. It is user-friendly. The output format is very clear to the user. Output testing is done on Smart Builders correctly

Acceptance Testing

Acceptance testing involves running a suite of tests on the completed system. Each individual test, known as a Case, exercises a particular operating condition of the user's environment or feature of the system and will result in a pass-fail, or Boolean outcome.

4.2 System Implementation

The implementation is the final stage, and it is an important phase. It involves invalid programming system testing. user training and the operational running of the developed proposed system that constitutes the application subsystems. A major task of preparing for implementation is the education of users which should have taken place much carrier in the project when they were involved in the investigation and design work. During the implementation phase system takes physical shape. To develop a system implemented planning is very essential. The implementation phase of the software development is concerned with translating design specifications into source code. The user tests the developed system, and changes are made according to their needs. Our system has been successfully implemented. Before implementation several tests have been conducted to ensure that no errors are encountered during the operation. The implementation phase ends with an evaluation of the system after placing it into operation for some time. The process of putting the developed system into actual use is called system implementation. This includes all those activities that take place to convert from the old system to the new system. The system can be implemented only after testing is done and is found to be working to specifications. The implementation stage is a systems project in its own right.

The implementation stage involves the following tasks:

● Careful planning

● Investigation of system and constraints

● Design of method to achieve change over

● Evaluation of the changeover method

In the case of this project all the screens are designed first. To make it to be executable, codes are written on each screen and perform the implementation by creating the database and connecting to the server. After that the system checks, whether it performs all the transactions Correctly. Then databases are cleared and made it to be usable to the technicians

5. SECURITY TECHNOLOGIES AND POLICIES

The protection of computer-based resources that includes hardware, software, data procedures and people against unauthorized use or natural. Disaster is known as System Security.

System Security can be divided into four related issues:

● Security

● Integrity

● Privacy

● Confidentiality

SYSTEM SECURITY refers to the technical innovations and procedures applied to the hardware and operation systems to protect against deliberate or accidental damage from a defined threat.

DATA SECURITY is the protection of data from loss, disclosure, modification, and destruction.

PRIVACY defines the rights of the user or organizations to determine what information they are willing to share with or accept from others and how the organization can be protected against unwelcome, unfair, or excessive dissemination of information about it.

CONFIDENTIALITY is a special status given to sensitive information in a database to minimize the possible invasion of privacy. It is an attribute of information that characterizes its needs for protection.

SECURITY IN SOFTWARE System security refers to various validations on data in the form of checks and controls to prevent the system from failing. It is always important to ensure that only valid data is entered, and only valid operations are performed on the systems

The system employs two types of checks and controls:

CLIENT-SIDE VALIDATION Various client-side validations are used to ensure on the client side that only valid data is entered. Client-side validation saves server time and loads to handle invalid data. Some checks imposed are:

● Forms cannot be submitted without filling up the mandatory data so that manual mistakes of submitting empty mandatory fields can be sorted out on the client side to save the server time and load.

● Tab-indexes are set according to the need and take into account the ease of the user while working with the system.

SERVER-SIDE VALIDATION Some checks cannot be applied on the client side. Server-side checks are necessary to save the system from failing and intimating the user that some invalid operation has been performed, or the performed operation is restricted.

Some of the server-side checks imposed are:

● The server-side constraint has been imposed to check for the validity of the primary key and foreign key. A primary key value cannot be duplicated. Any attempt to duplicate the primary value results in a message intimating the user about those values through the forms using foreign keys can be updated only with the existing foreign key values.

● The user is intimating through appropriate messages about the successful operations or exceptions occurring on the server side.

● Various Access Control Mechanisms have been built so that one user may not agitate upon another. Access permissions to various types of users are controlled according to the organizational structure. Only permitted users can log on to the system and can have access according to their category. User names, passwords, and permissions are controlled over the server side.

● Using server-side validation, constraints on several restricted operations are imposed.

6. MAINTENANCE

Software maintenance is the modification of a software product and delivery to correct faults, and improve performance or other attributes. Maintenance is the ease with which a program can be corrected if any error is encountered, adapted if its environment changes or enhanced if the customer desires a change in requirement. Maintenance follows conversation to extend that changes are necessary to maintain satisfactory operations relative to changes in the user's environment.

Maintenance often includes minor enhancements or corrections to problems that surface in the system's operation. Maintenance is also done based on fixing the problems reported, changing the interface with other software or hardware enhancing the software.

CATEGORIES OF MAINTENANCE

Corrective Maintenance

Corrective maintenance is the most commonly used maintenance approach, but it is easy to see its limitations. When equipment fails, it often leads to downtime in production and sometimes damages other parts. In most cases, this is expensive. Also, if the equipment needs to be replaced, the cost of replacing it alone can be substantial. The reliability of systems maintained by this type of maintenance is unknown and cannot be measured. Corrective maintenance is possible since the consequences of failure or worn out are not significant and the cost of this maintenance is not great.

Adaptive Maintenance

Modification of a software product performed after delivery to keep a are product usable m a changed or changing environment. Adaptive maintenance includes any work initiated as a consequence of moving the software to a different hardware or software platform. It is a change driven by the need to accommodate modifications in the environment of a software system. The environment in this context refers to the totality of all conditions and influences which act from outside upon the system. A change to the whole or part of this environment will Warrant a corresponding modification of the software.

Perfective Maintenance

Modification of a software product alters delivery to improve performance or maintainability. This term is used to describe changes undertaken to expand the existing requirements of the system. A successful piece of software lends to be subjected to the succession of changes resulting in an increase in user requirements. This is based on the premise that as the software becomes useful, the user experiments with new cases beyond the Scope for which it was initially developed. Vxpansi01 n requirements can take the form of enhancement of existing system functionality and improvement in computational efficiency.

Preventive Maintenance

Preventive maintenance is a schedule of planned maintenance actions aimed at the prevention of breakdowns and failures. The primary goal of preventive maintenance is to prevent the failure of equipment before it occurs. It is designed to preserve and enhance equipment reliability by replacing worn components before they actually fail. Preventive maintenance activities include equipment checks, and partial or complete overhauls at specified periods.

Long-term benefits of preventive maintenance include:

● Improved system reliability

● Decreased cost of replacement

● Decreased system downtime

7. SCOPE FOR FUTURE ENHANCEMENT

The 'RentIt' project has significant potential for future enhancements to improve its functionality and user experience. One key enhancement could involve expanding the platform's product categories to include additional rental items such as event supplies, vehicles, or specialized equipment. By broadening the scope of available products, 'RentIt' can cater to a wider range of customer needs.

Another potential enhancement is the introduction of a dynamic pricing model that allows shops to adjust rental rates based on demand, availability, and seasonal trends. This feature would enable shops to optimize pricing strategies and improve revenue generation.

Additional enhancements may include advanced performance analytics powered by AI to help shop owners gain deeper insights into booking trends, customer preferences, and inventory management. Implementing an AI chatbot for customer assistance can provide instant support and improve user engagement. Furthermore, adding offline access with push notifications can enhance the platform's accessibility, ensuring users can access important information even without an active internet connection. With these enhancements, 'RentIt' can continue to evolve and remain competitive in the growing rental market.

8. CONCLUSION

The 'RentIt' project is designed to address the increasing demand for rental services by offering a feature-rich and user-centric platform. By integrating secure transactions, efficient booking management, and comprehensive customer support, the platform ensures a seamless experience for both users and shop owners. The project's focus on scalability, performance, and enhanced features positions it as a reliable solution for the rental industry.

With continuous improvements and strategic enhancements, 'RentIt' can adapt to evolving customer needs and technological advancements. By implementing future upgrades such as AI-powered analytics, chatbot support, and offline accessibility, 'RentIt' is poised to become a comprehensive and indispensable platform in the rental services sector.

9. BIBLIOGRAPHY

1. Jalotte, Pankaj. "SOFTWARE ENGINEERING", Third Edition, Spring Publishers, Narosa Publishing House.

2. Flutter – Build apps for any screen - https://flutter.dev/ 3. The official repository for Dart and Flutter packages - https://pub.dev/

4. W3Schools - https://www.w3schools.com

5. Figma – UI/UX Design Tool - https://www.figma.com/

6. Supabase – Open Source Firebase Alternative - https://supabase.com/

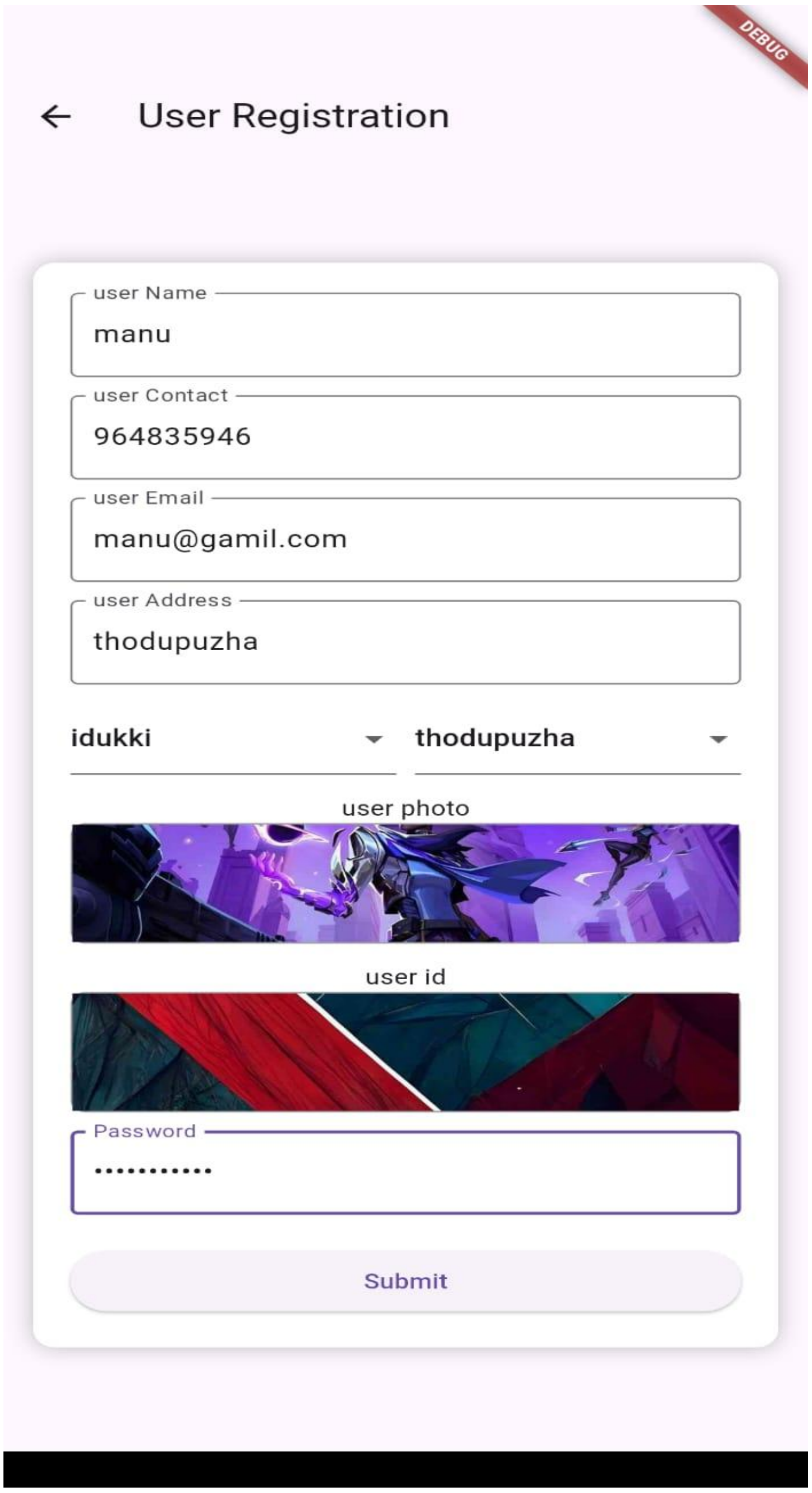
7. Firebase – Backend Services for Apps - https://firebase.google.com/

8. PostgreSQL – Open Source Database - https://www.postgresql.org

10. APPENDIX

10.1 Screenshots

User registration:



User Registration

user Name
manu

user Contact
964835946

user Email
manu@gamil.com

user Address
thodupuzha

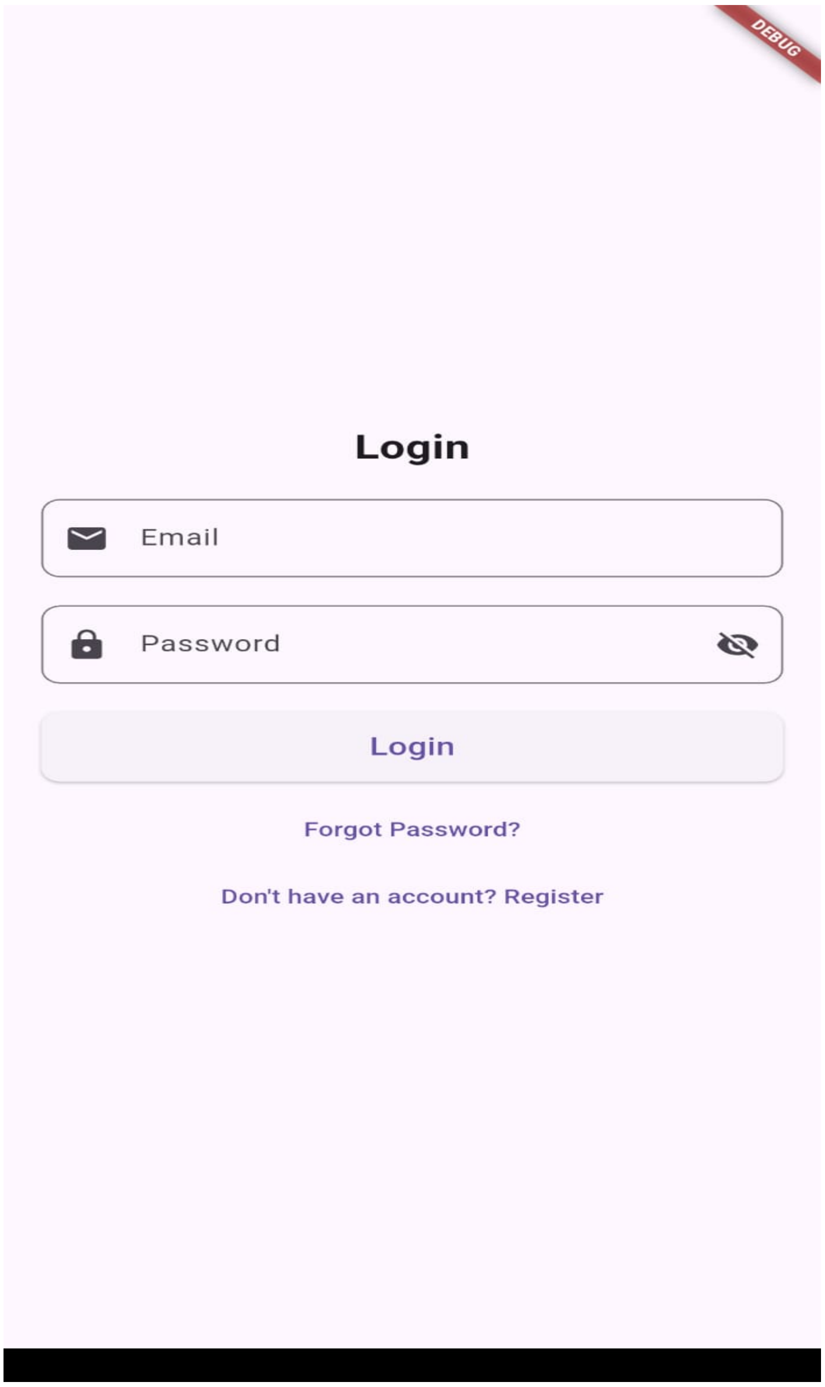
idukki
thodupuzha

user photo

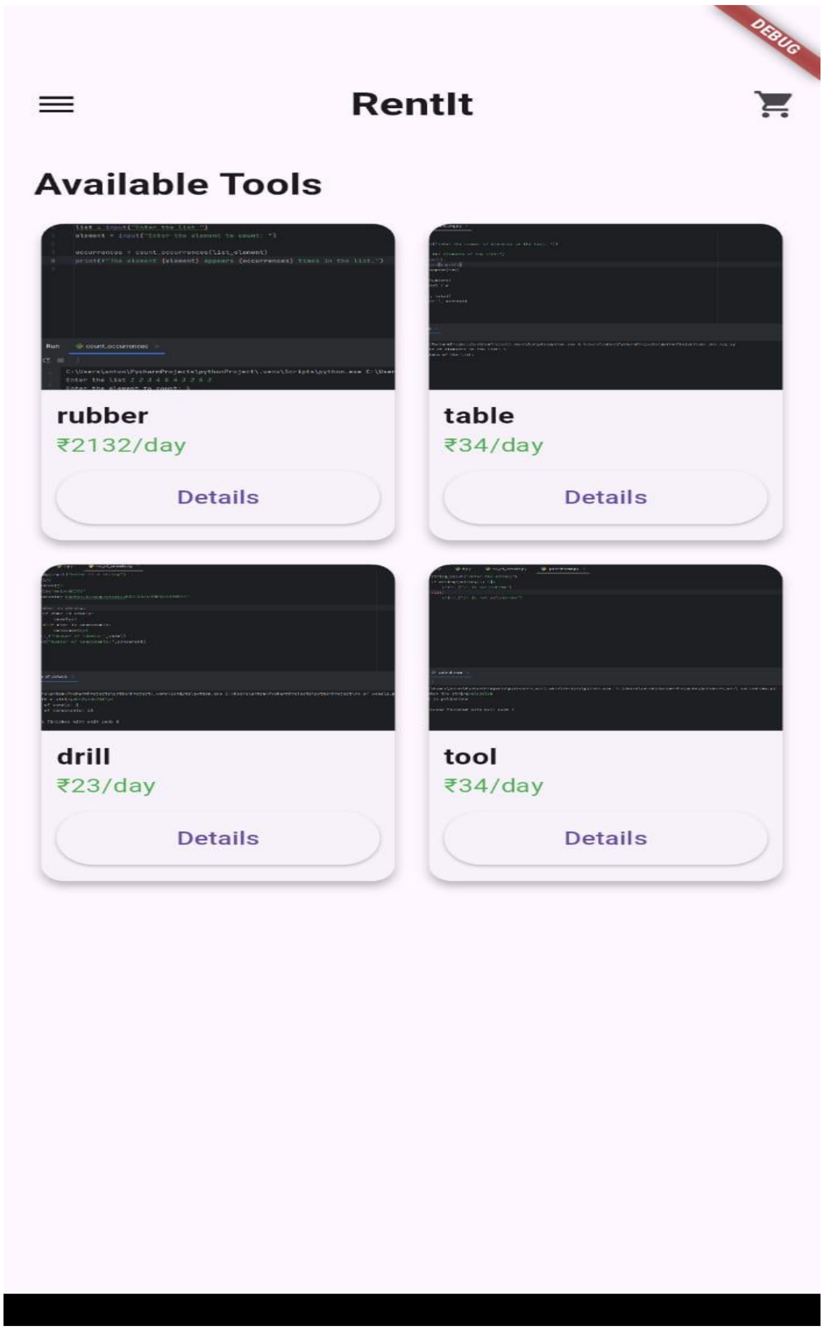
user id

Password
•••••••••••

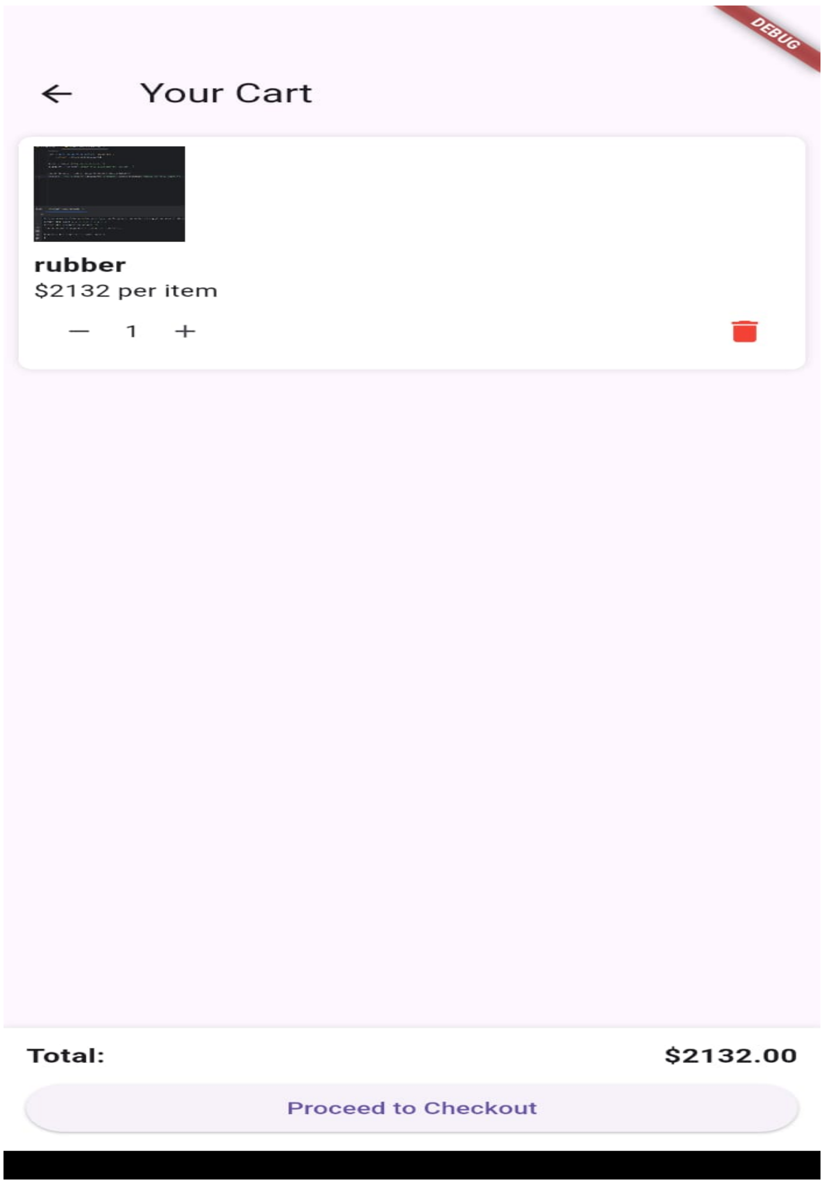
User login:


DEBUG
Login
Email
Password
Login
Forgot Password?
Don't have an account? Register

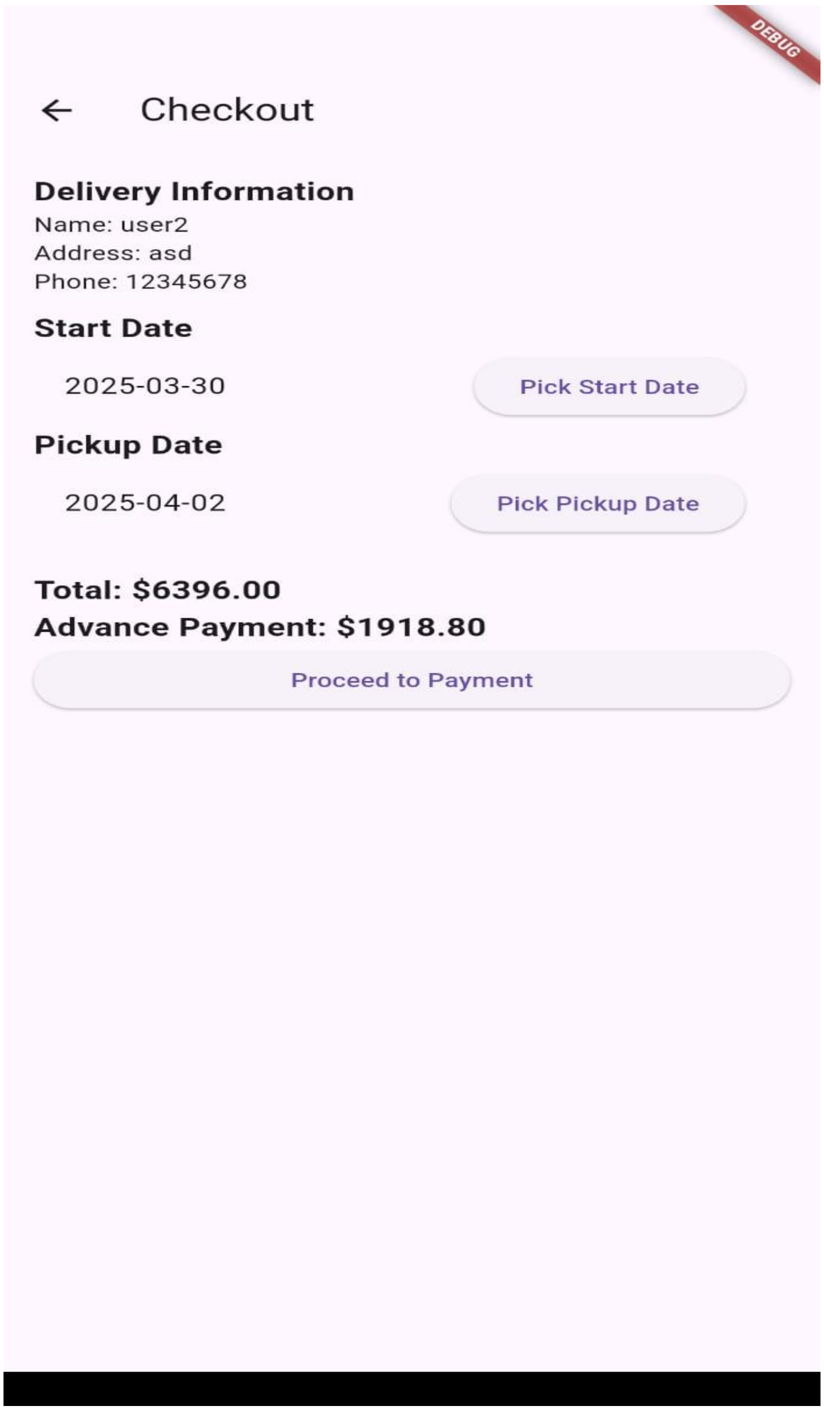
User home page:



Cart:


DEBUG
Your Cart
rubber
$2132 per item
1
Total:
$2132.00
Proceed to Checkout

Checkout page:



Checkout

Delivery Information

Name: user2

Address: asd

Phone: 12345678

Start Date

2025-03-30

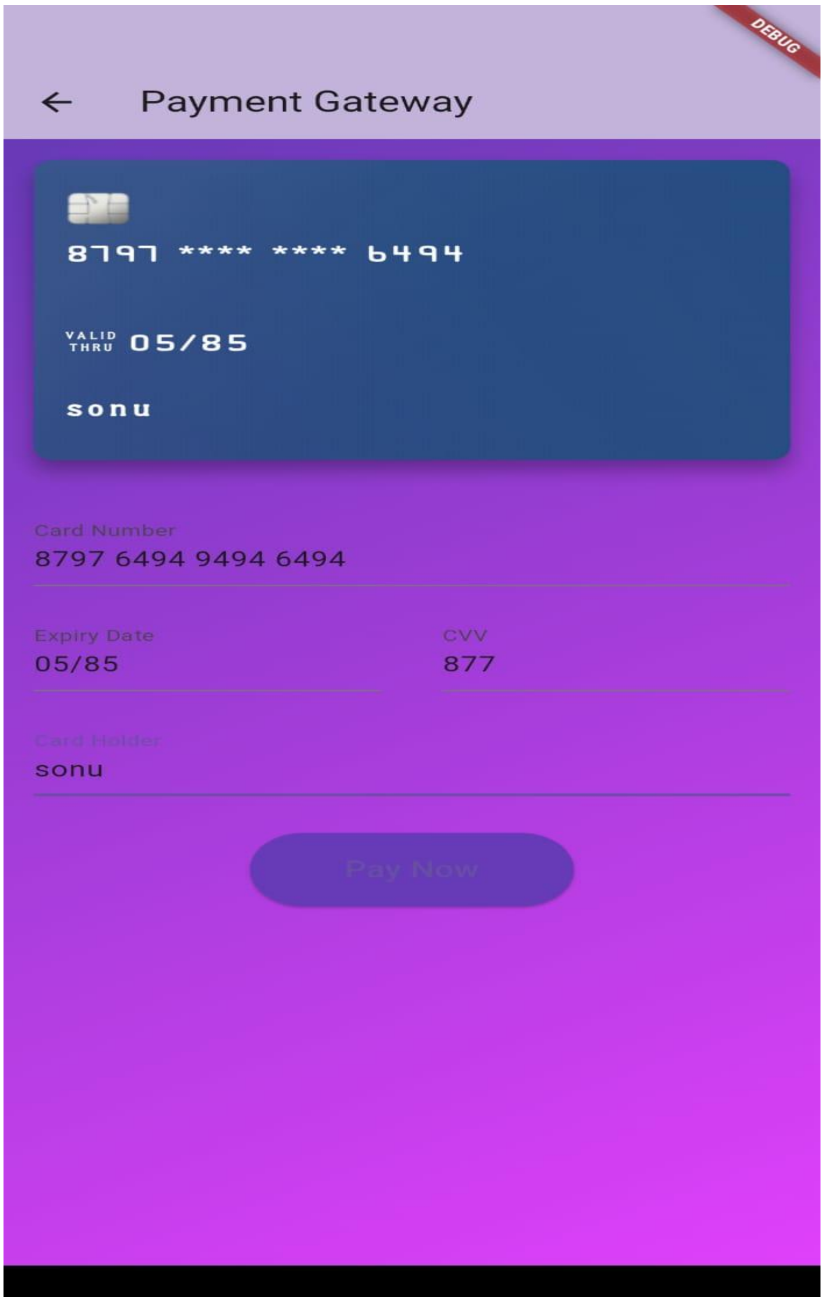
Pickup Date

2025-04-02

Total: $6396.00

Advance Payment: $1918.80

Payment gateway:


Payment Gateway

8797 **** **** 6494

VALID THRU 05/85

sonu

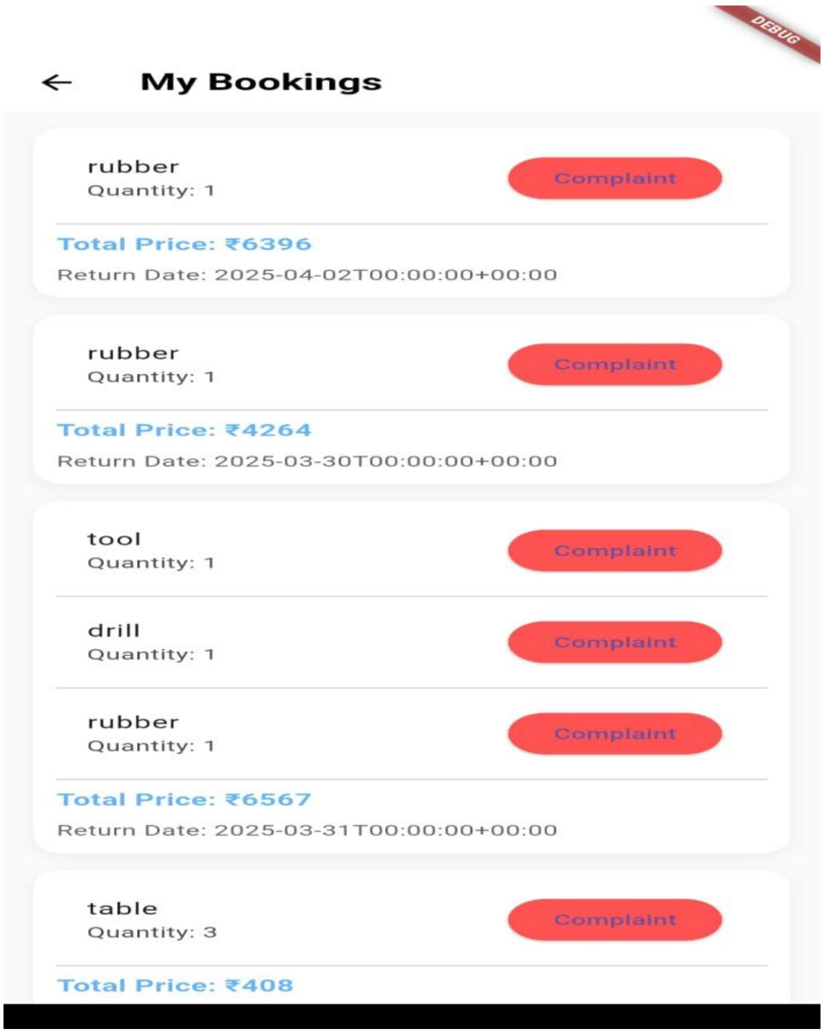
Card Number
8797 6494 9494 6494

Expiry Date
05/85

CVV
877

Card Holder
sonu

My booking:



DEBUG

My Bookings

rubber
Quantity: 1
Complaint
Total Price: ₹6396
Return Date: 2025-04-02T00:00:00+00:00

rubber
Quantity: 1
Complaint
Total Price: ₹4264
Return Date: 2025-03-30T00:00:00+00:00

tool
Quantity: 1
Complaint

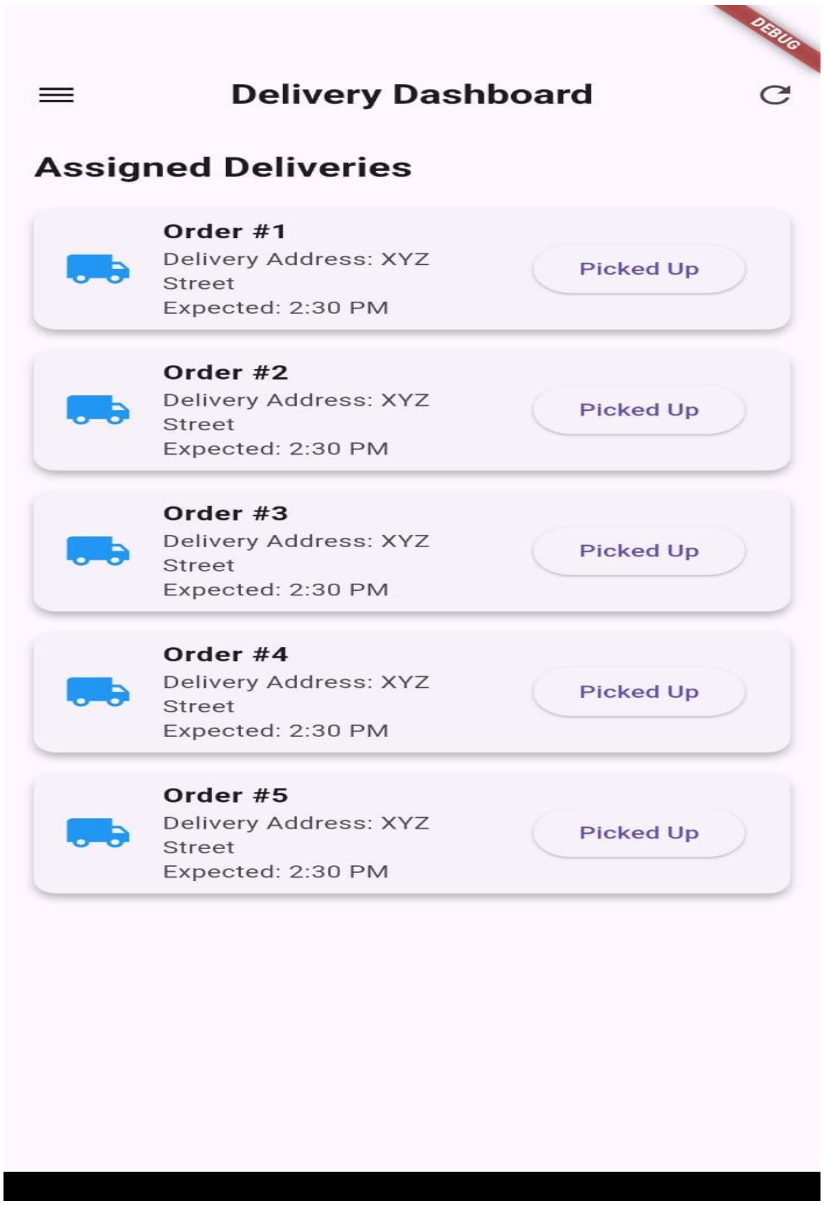
drill
Quantity: 1
Complaint

rubber
Quantity: 1
Complaint
Total Price: ₹6567
Return Date: 2025-03-31T00:00:00+00:00

table
Quantity: 3
Complaint
Total Price: ₹408

Delivery boy

Delivery boy dashboard:



Delivery Dashboard

Assigned Deliveries

Order #1
Delivery Address: XYZ Street
Expected: 2:30 PM
Picked Up

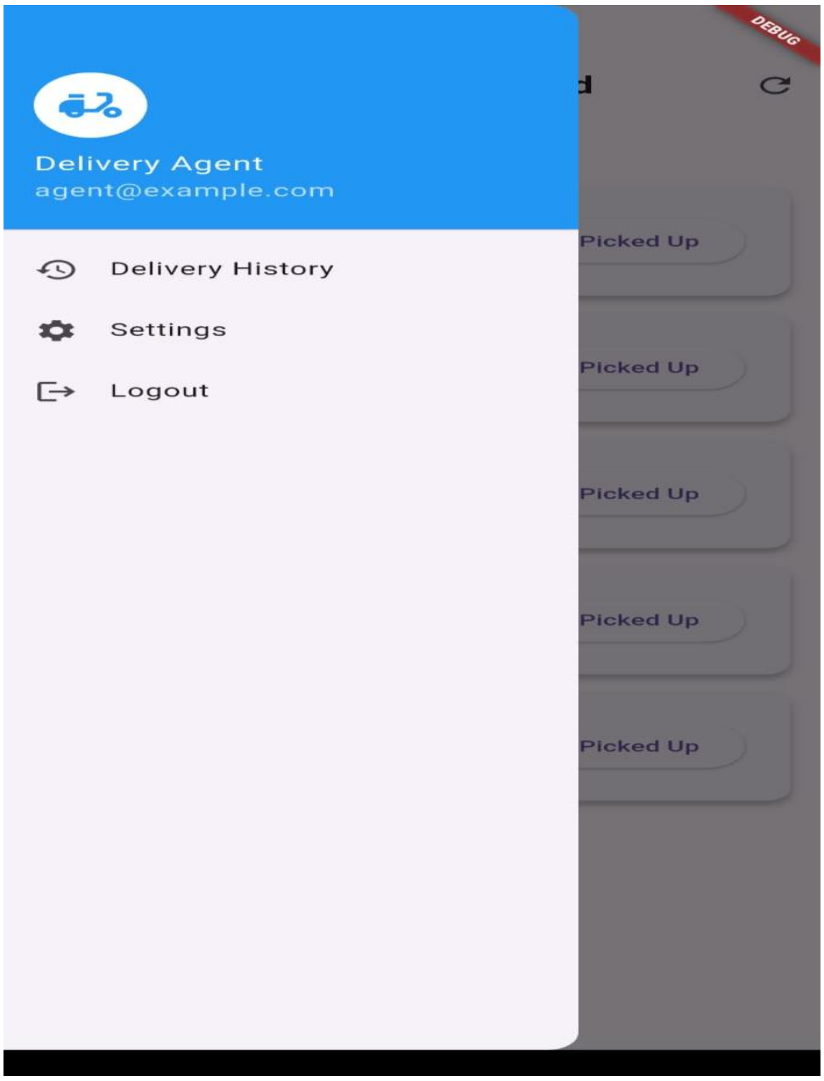
Order #2
Delivery Address: XYZ Street
Expected: 2:30 PM
Picked Up

Order #3
Delivery Address: XYZ Street
Expected: 2:30 PM
Picked Up

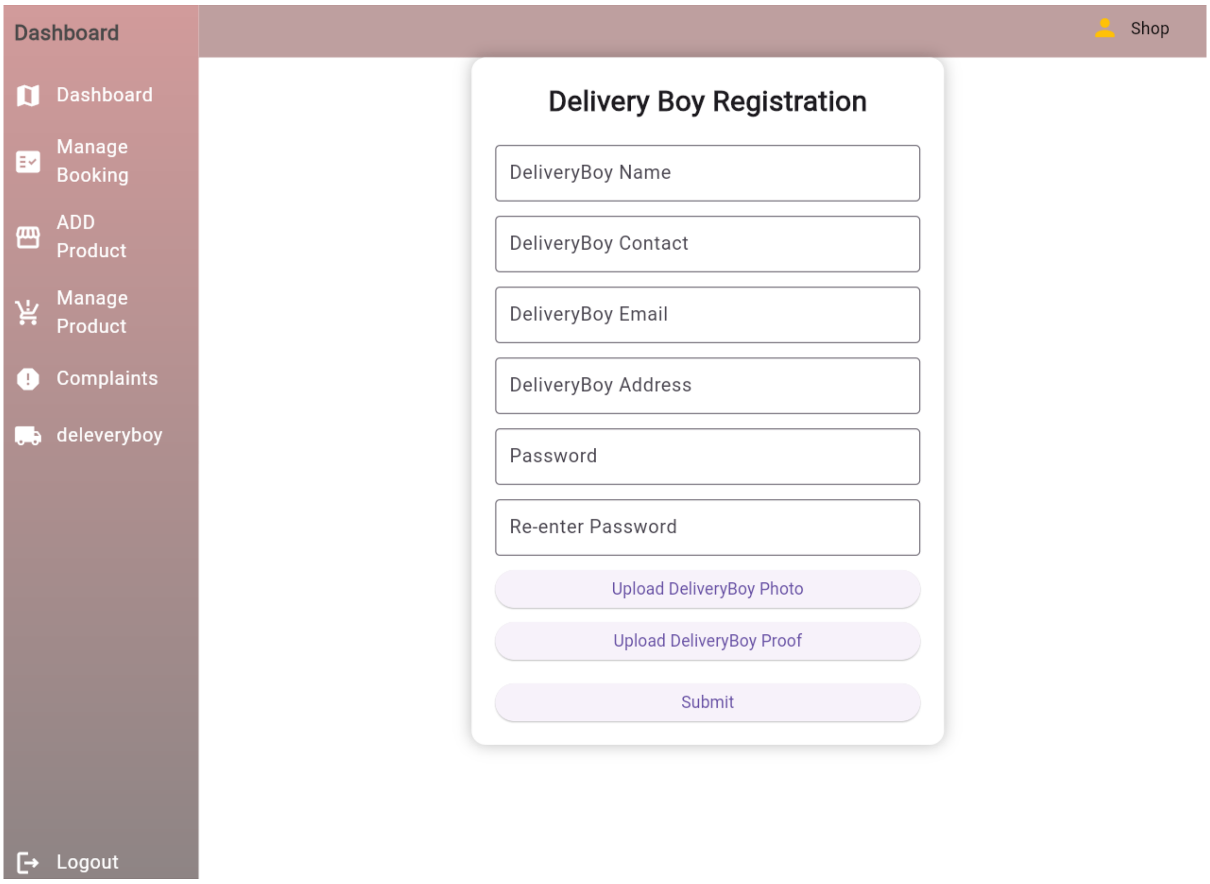
Order #4
Delivery Address: XYZ Street
Expected: 2:30 PM
Picked Up

Order #5
Delivery Address: XYZ Street
Expected: 2:30 PM
Picked Up

Drawer page:


DEBUG
Delivery Agent
agent@example.com
Delivery History
Settings
Logout
Picked Up
Picked Up
Picked Up
Picked Up
Picked Up

Deliver boy registration:



Delivery Boy Registration

DeliveryBoy Name

DeliveryBoy Contact

DeliveryBoy Email

DeliveryBoy Address

Password

Re-enter Password

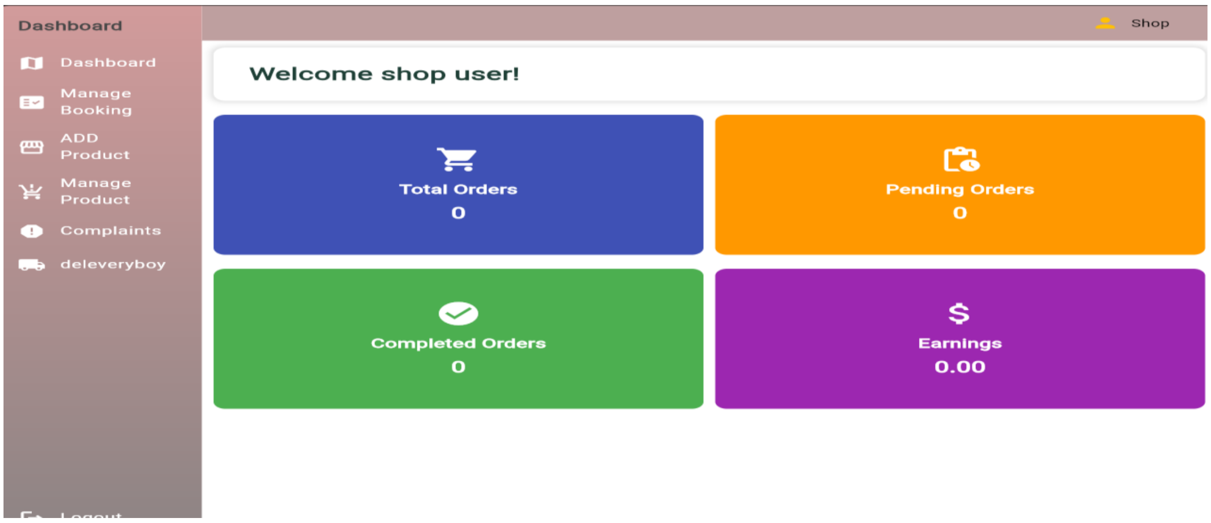
Upload DeliveryBoy Photo

Upload DeliveryBoy Proof

Submit

Shop

Shop dashboard:



Welcome shop user!

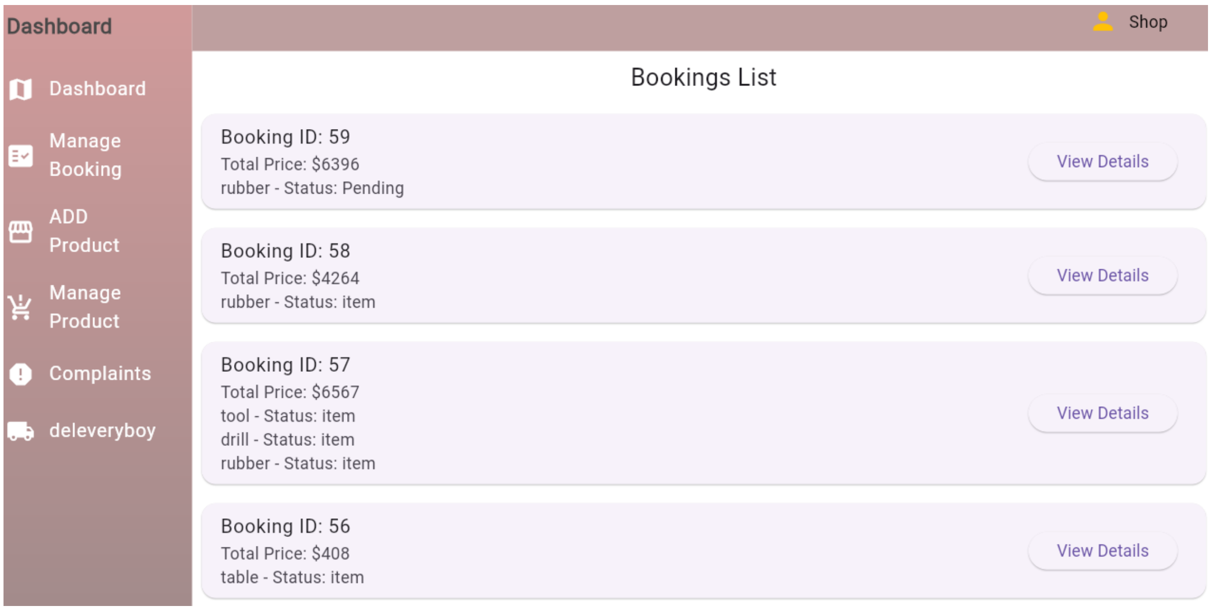
Total Orders 0

Pending Orders 0

Completed Orders 0

Earnings 0.00

Manage booking:



Bookings List

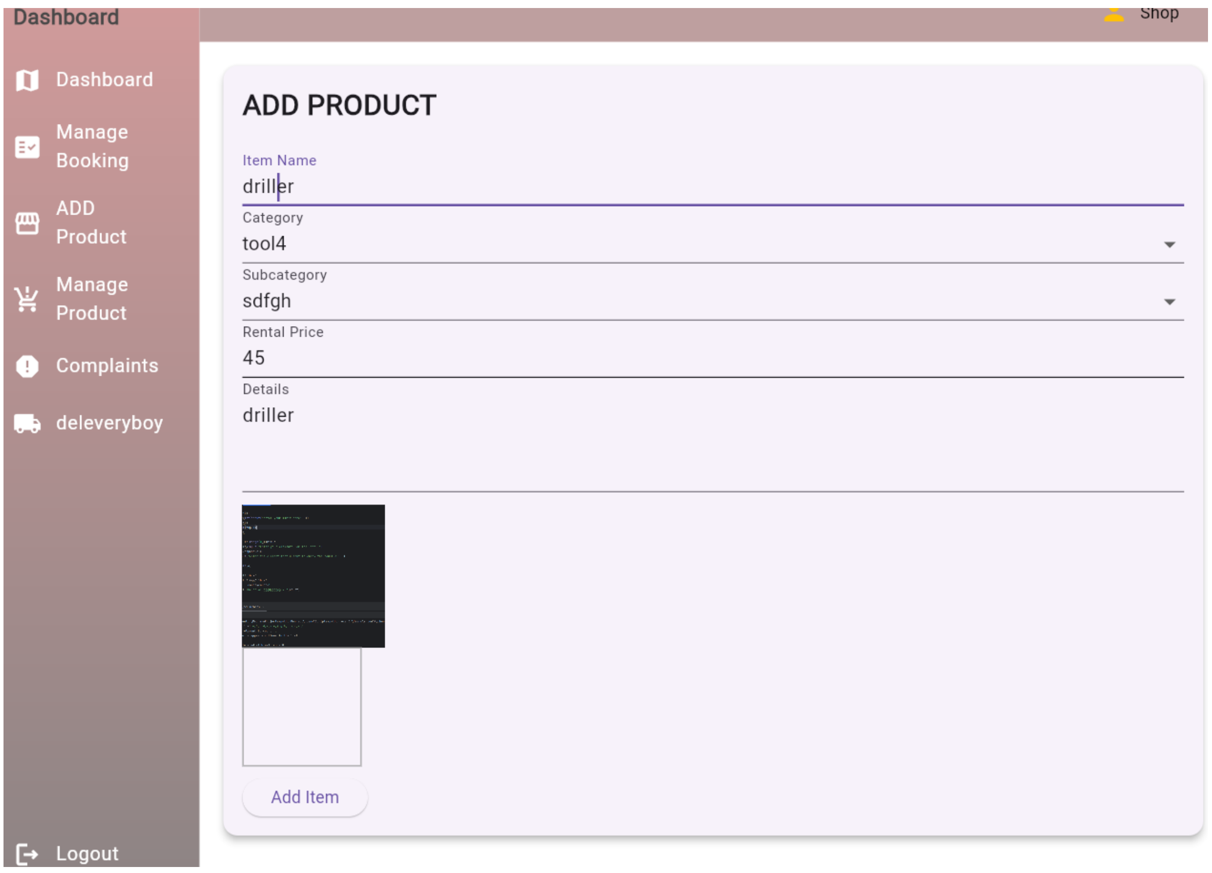
Booking ID: 59
Total Price: $6396
rubber - Status: Pending
View Details

Booking ID: 58
Total Price: $4264
rubber - Status: item
View Details

Booking ID: 57
Total Price: $6567
tool - Status: item
drill - Status: item
rubber - Status: item
View Details

Booking ID: 56
Total Price: $408
table - Status: item
View Details

Add product:

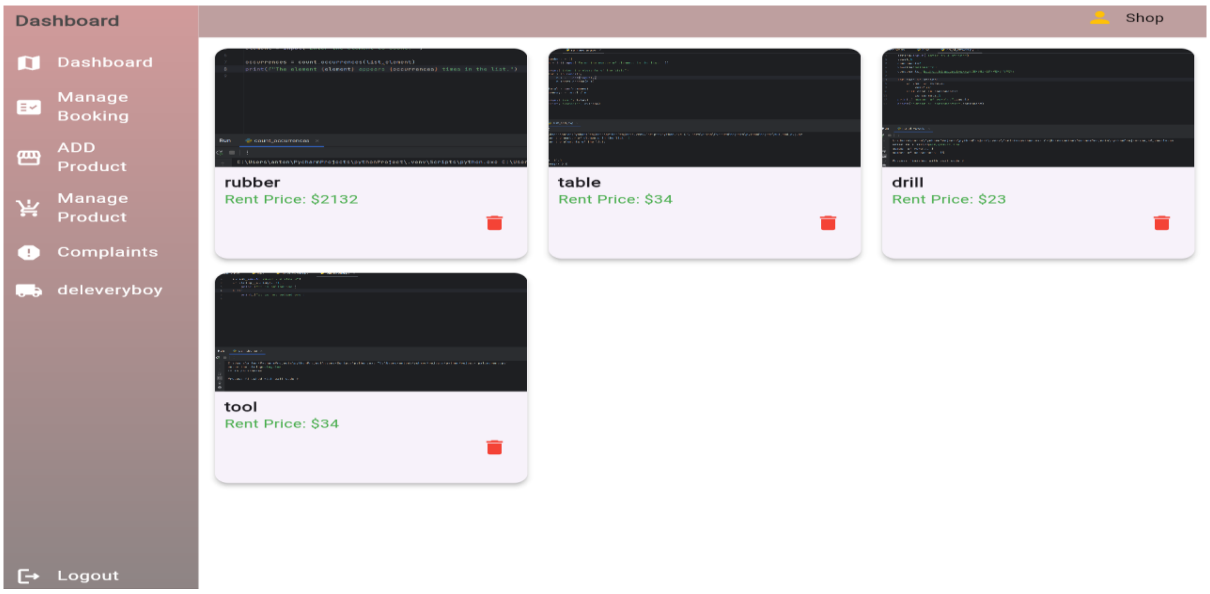


ADD PRODUCT

Item Name: driller
Category: tool4
Subcategory: sdfgh
Rental Price: 45
Details: driller

Add Item

Manage product:



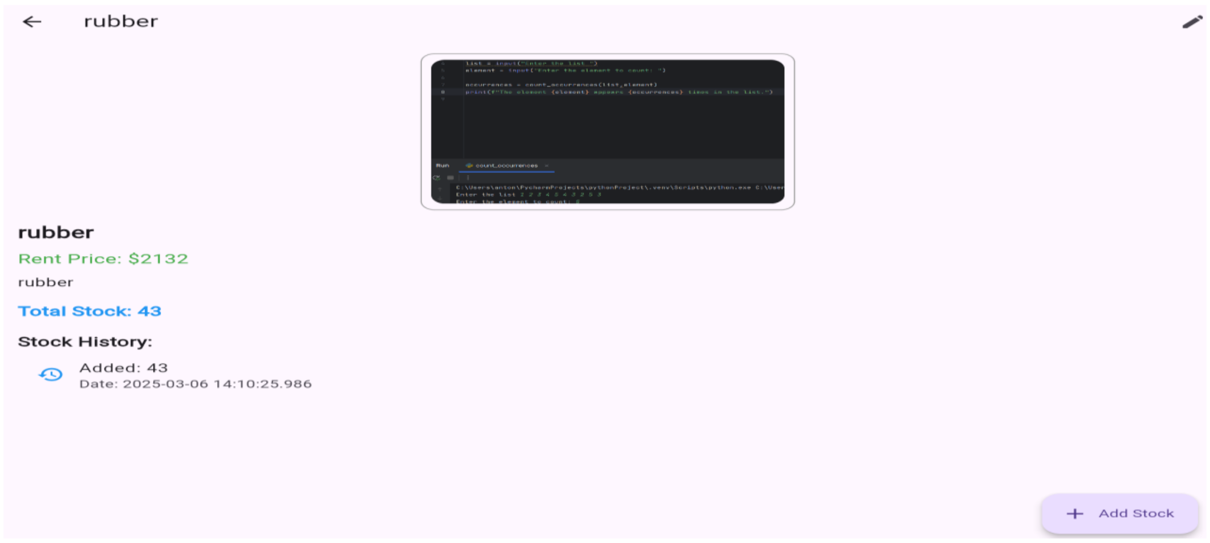
rubber
Rent Price: $2132

table
Rent Price: $34

drill
Rent Price: $23

tool
Rent Price: $34

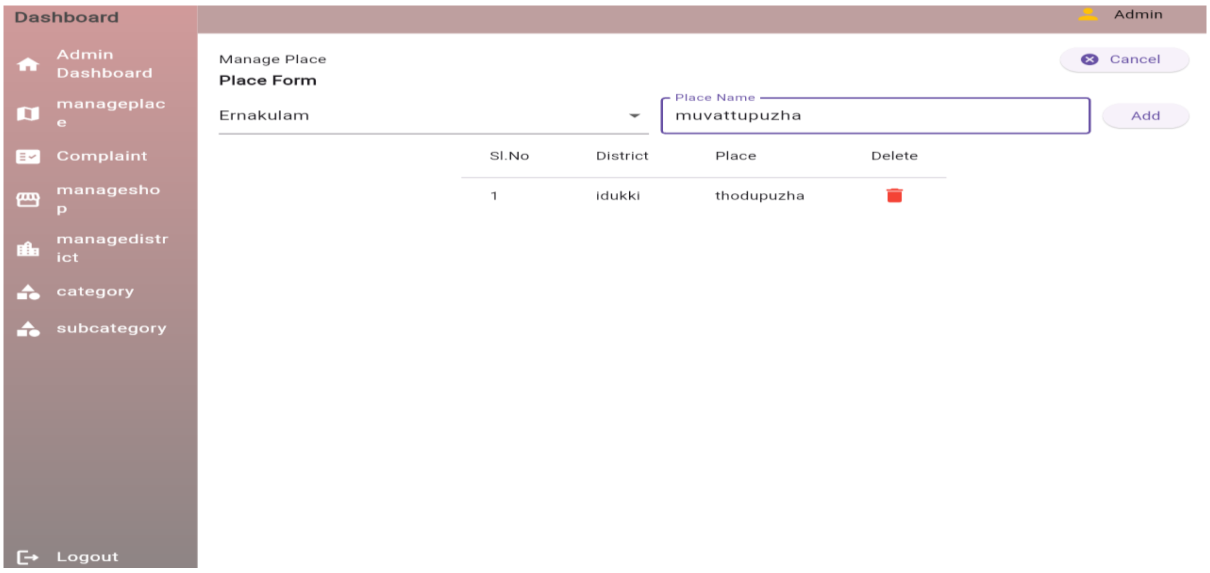
Product details and add stocks:



rubber
Rent Price: $2132
rubber
Total Stock: 43
Stock History:
Added: 43
Date: 2025-03-06 14:10:25.986

Admin:

Manage place

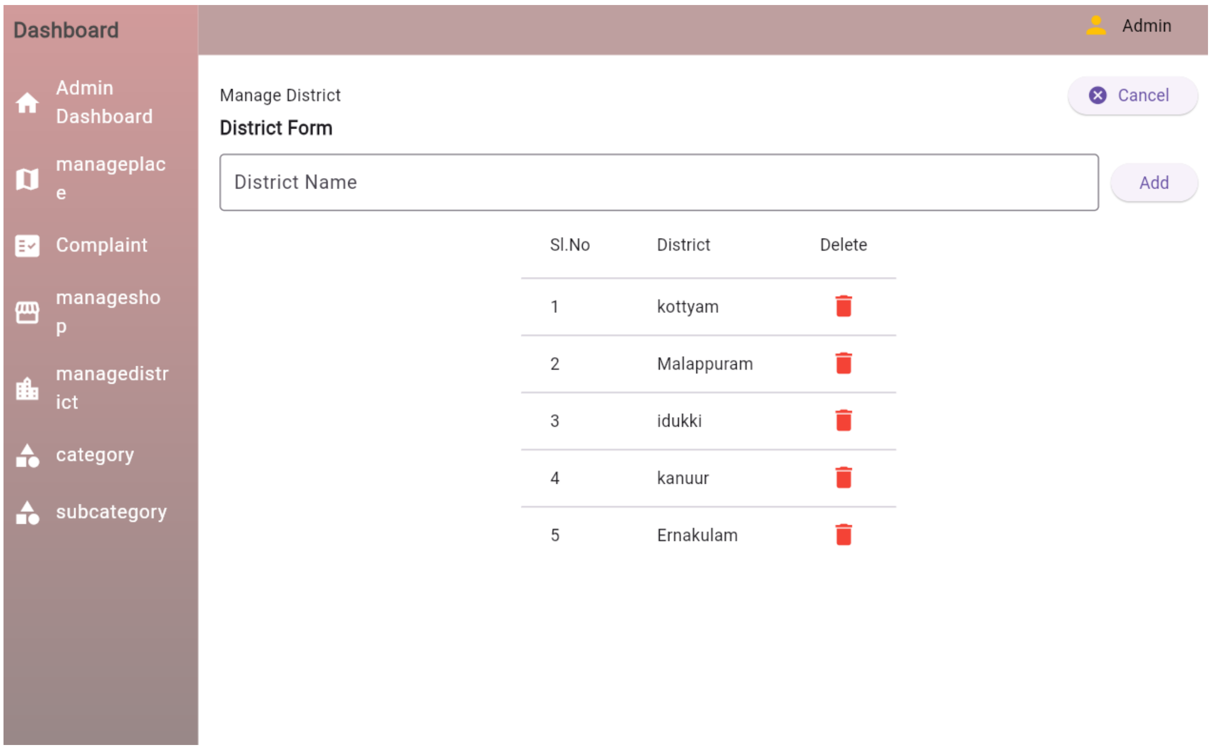


Manage Place
Place Form

Ernakulam
Place Name: muvattupuzha

Sl.No	District	Place	Delete
1	idukki	thodupuzha	🗑

Manage district:

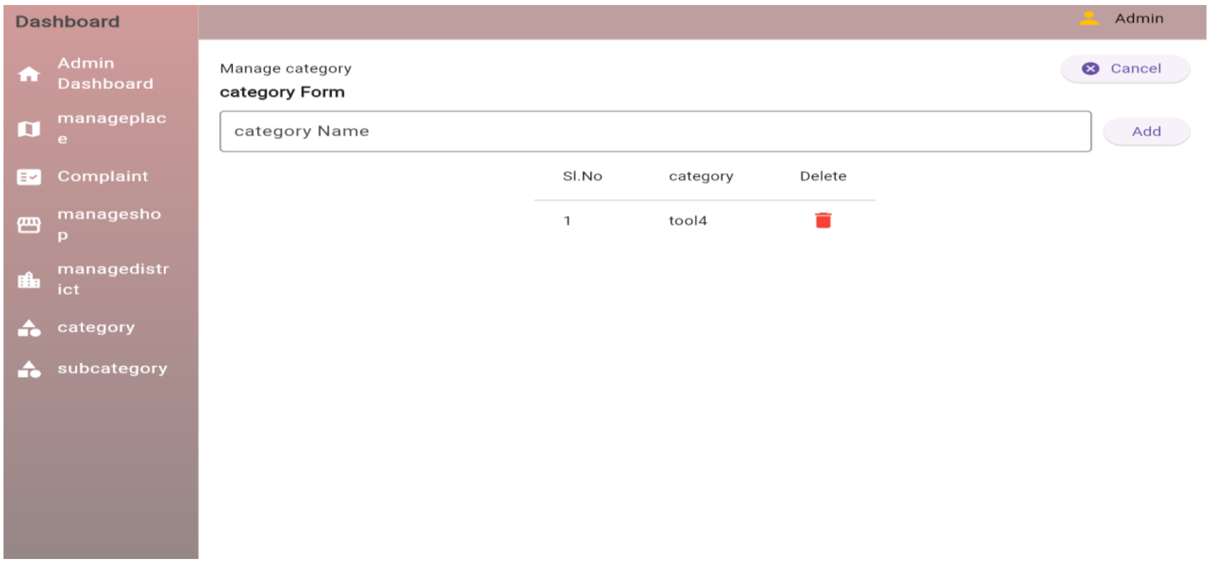


Manage District
District Form

District Name

Sl.No	District	Delete
1	kottyam	🗑
2	Malappuram	🗑
3	idukki	🗑
4	kanuur	🗑
5	Ernakulam	🗑

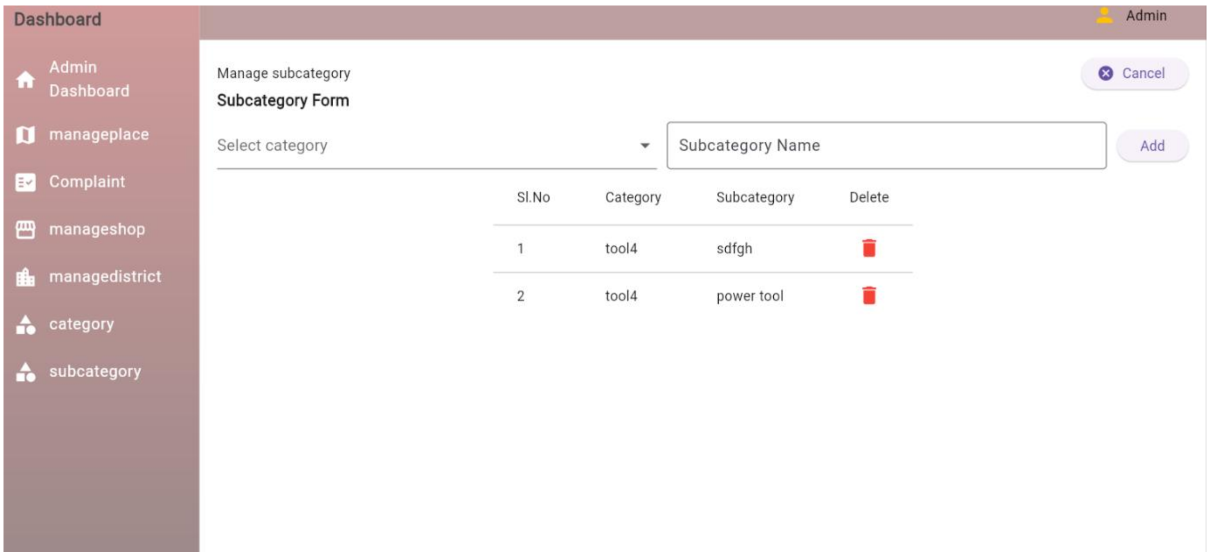
Manage category:



Manage category
category Form
category Name

Sl.No	category	Delete
1	tool4	🗑

Manage subcategory:



Manage subcategory
Subcategory Form
Select category
Subcategory Name

Sl.No	Category	Subcategory	Delete
1	tool4	sdfgh	🗑
2	tool4	power tool	🗑

10.2 Code

10.2.1 main.dart – user

```
import 'package:flutter/material.dart';

import 'package:supabase_flutter/supabase_flutter.dart';

import 'package:user/screen/login.dart';



Future<void> main() async {

  await Supabase.initialize(

   url: 'https://arixvhqgapwaxuycshaa.supabase.co',

anonKey:'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIs
InJlZiI6ImFyaXh2aHFnYXB3YXh1eWNzaGFhIiwicm9sZSI6ImFub24iLCJpYXQiOjE3Mz
QzNDYzNjUsImV4cCI6MjA0OTkyMjM2NX0.dRr9mK8ug9UeIif_0UskxKuXsNh8pDoBT
ST4ShPzTnA',

  );

  runApp(const MainApp());

}



final supabase = Supabase.instance.client;

class MainApp extends StatelessWidget {

  const MainApp({super.key});
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home:UserLoginPage() ,
    );
  }
}
```

Login.dart

```
import 'package:flutter/material.dart';
import 'package:user/main.dart';
import 'package:user/screen/userRegistration.dart';
import 'package:user/screen/userhomepage.dart'; // Fixed import by adding .dart extension

class UserLoginPage extends StatefulWidget {
  const UserLoginPage({super.key});



  @override

  _UserLoginPageState createState() => _UserLoginPageState();

}



class _UserLoginPageState extends State<UserLoginPage> {

  final TextEditingController _emailController = TextEditingController();

  final TextEditingController _passwordController = TextEditingController();



  bool _obscurePassword = true;



  Future<void> login() async {

    try {



    await supabase.auth.signInWithPassword(

        password: _passwordController.text,
        email: _emailController.text);

      Navigator.pushReplacement(

        context,

        MaterialPageRoute(

          builder: (context) => UserHomePage(),

        ));

    }

     catch (e) {

      print("Error: $e");

    ScaffoldMessenger.of(context).showSnackBar(

      SnackBar(content: Text("Login failed: ${e.toString()}")),

    );

   }

  }


  @override

  Widget build(BuildContext context) {
```

```
return Scaffold(
  body: Center(
    child: Padding(
      padding: const EdgeInsets.all(20.0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text(
            "Login",
            style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
          ),
          SizedBox(height: 20),
          TextField(
            controller: _emailController,
            decoration: InputDecoration(
              labelText: "Email",
              prefixIcon: Icon(Icons.email),
                                    border:    OutlineInputBorder(borderRadius:
BorderRadius.circular(10)),
                  ),
                ),
                SizedBox(height: 20),
                TextField(
                  controller: _passwordController,
                  obscureText: _obscurePassword,
                  decoration: InputDecoration(
                    labelText: "Password",
                    prefixIcon: Icon(Icons.lock),
                    suffixIcon: IconButton(
                      icon: Icon(
                        _obscurePassword ? Icons.visibility_off : Icons.visibility,
                      ),
                      onPressed: () {
                        setState(() {
                          _obscurePassword = !_obscurePassword;
                  });
                 },
                ),
                            border:   OutlineInputBorder(borderRadius:
BorderRadius.circular(10)),
               ),
             ),
             SizedBox(height: 20),
             ElevatedButton(
              onPressed: () {
               print("Button");
               login();
              },
              style: ElevatedButton.styleFrom(
               minimumSize: Size(double.infinity, 50),
               shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(10),
               ),
               ),
               child: Text("Login", style: TextStyle(fontSize: 18)),
             ),
             SizedBox(height: 10),
             TextButton(
               onPressed: () {},
               child: Text("Forgot Password?"),
             ),
             TextButton(
               onPressed: () {
                 Navigator.push(
                       context,
                       MaterialPageRoute(
                         builder: (context) => const Userregistration(),
                       ),
                     );
               },
              child: Text("Don't have an account? Register"),
             ),
            ],
          ),
         ),
       ),
     );
   }
 }
```

User_registration.dart

```
import 'package:flutter/material.dart';
import 'package:file_picker/file_picker.dart';
import 'dart:io';
import 'package:supabase_flutter/supabase_flutter.dart';
import 'package:user/screen/login.dart';
```

```
class Userregistration extends StatefulWidget {
  const Userregistration({super.key});


  @override
  State<Userregistration> createState() => _userregistrationState();
}


class _userregistrationState extends State<Userregistration> {
      final TextEditingController _userNameController =
TextEditingController();
      final TextEditingController _userContactController =
TextEditingController();
      final TextEditingController _userEmailController =
TextEditingController();
      final TextEditingController _userAddressController =
TextEditingController();
      final TextEditingController _passwordController =
TextEditingController();
```

```
        String? selectedDist;

        String? selectedPlace;

        List<Map<String, dynamic>> _distList = [];

        List<Map<String, dynamic>> _placeList = [];

        PlatformFile? _userphoto;

        PlatformFile? _userid;



        Future<void> _pickImage(bool isLogo) async {

          FilePickerResult? result = await FilePicker.platform.pickFiles(type:
FileType.image);

         if (result != null) {

          setState(() {

           if (isLogo) {

            _userphoto = result.files.first;

           } else {

            _userid = result.files.first;

           }
```

```
    });

   }

  }



  Future<void> fetchDist() async {

   try {

                    final       response       =       await
Supabase.instance.client.from('tbl_district').select();

    setState(() {

     _distList = List<Map<String, dynamic>>.from(response);

    });

   } catch (e) {

    print("Error fetching districts: $e");

   }

  }



  Future<void> fetchPlace(String id) async {

   try {
```

```
                                      final        response        =        await
Supabase.instance.client.from('tbl_place').select().eq('district_id', id);
        setState(() {
          _placeList = List<Map<String, dynamic>>.from(response);
        });
      } catch (e) {
        print("Error fetching places: $e");
      }
    }


    Future<void> register() async {
      try {
        final auth = await Supabase.instance.client.auth.signUp(
          password: _passwordController.text,
          email: _userEmailController.text,
        );
        final uid = auth.user?.id;
        if (uid != null && uid.isNotEmpty) {
```

```
      await storeData(uid);

     }

   } catch (e) {

    print("Error in authentication: $e");

   }

 }



 Future<void> storeData(String uid) async {

   try {

    await Supabase.instance.client.from('tbl_user').insert({

      'user_id': uid,

      'user_name': _userNameController.text,

      'user_contact': _userContactController.text,

      'user_address': _userAddressController.text,

      'user_email': _userEmailController.text,

      'user_password': _passwordController.text,

      'place_id': selectedPlace,
     });

   } catch (e) {

    print("Error inserting data: $e");

   }

 }



 @override

 void initState() {

   super.initState();

   fetchDist();

 }



 @override

 Widget build(BuildContext context) {

   return Scaffold(

    appBar: AppBar(title: Text('User Registration')),

    body: Center(
 child: Container(
   width: 400, // Adjust width to make it compact
   padding: EdgeInsets.all(20),
   decoration: BoxDecoration(
    color: Colors.white,
    borderRadius: BorderRadius.circular(12),
    boxShadow: [BoxShadow(color: Colors.black26, blurRadius: 10)],
   ),
   child: SingleChildScrollView(
    child: Column(
      children: [
        TextFormField(
          controller: _userNameController,
            decoration: InputDecoration(labelText: 'user Name', border:
 OutlineInputBorder()),
        ),
        SizedBox(height: 12),
        TextFormField(
          controller: _userContactController,
          decoration: InputDecoration(labelText: 'user Contact', border: OutlineInputBorder()),
          keyboardType: TextInputType.phone,
        ),
        SizedBox(height: 12),
        TextFormField(
          controller: _userEmailController,
            decoration: InputDecoration(labelText: 'user Email', border: OutlineInputBorder()),
          keyboardType: TextInputType.emailAddress,
        ),
        SizedBox(height: 12),
        TextFormField(
          controller: _userAddressController,
          decoration: InputDecoration(labelText: 'user Address', border: OutlineInputBorder()),
        ),
SizedBox(height: 12),

Row(

  children: [

    Expanded(

      child: DropdownButtonFormField<String>(

        value: selectedDist,

        hint: Text("Select District"),

        onChanged: (newValue) {

          setState(() {

            selectedDist = newValue;

            fetchPlace(newValue!);

          });

        },

        items: _distList.map((district) {

          return DropdownMenuItem<String>(

            value: district['district_id'].toString(),

            child: Text(district['district_name']),
          );

        }).toList(),

      ),

    ),

    SizedBox(width: 10),

    Expanded(

      child: DropdownButtonFormField<String>(

        value: selectedPlace,

        hint: Text("Select Place"),

        onChanged: (newValue) {

          setState(() {

            selectedPlace = newValue;

          });

        },

        items: _placeList.map((place) {

          return DropdownMenuItem<String>(

            value: place['place_id'].toString(),
                    child: Text(place['place_name']),

                   );

                 }).toList(),

               ),

             ),

           ],

         ),

         SizedBox(height: 12),

         Text('user photo'),

         GestureDetector(

           onTap: () => _pickImage(true),

           child: Container(

             width: double.infinity,

             height: 80,

                      decoration: BoxDecoration(border: Border.all(color:
Colors.grey), borderRadius: BorderRadius.circular(8)),
                                 child: _userphoto == null ? Center(child:
Icon(Icons.add_a_photo, size: 40)) : Image.file(File(_userphoto!.path!), fit:
BoxFit.cover),
           ),
         ),
         SizedBox(height: 12),
         Text('user id'),
         GestureDetector(
           onTap: () => _pickImage(false),
           child: Container(
             width: double.infinity,
             height: 80,
                      decoration: BoxDecoration(border: Border.all(color:
Colors.grey), borderRadius: BorderRadius.circular(8)),
             child: _userid == null ? Center(child: Icon(Icons.add_a_photo,
size: 40)) : Image.file(File(_userid!.path!), fit: BoxFit.cover),
           ),
         ),
         SizedBox(height: 12),
         TextFormField(
           controller: _passwordController,
               decoration: InputDecoration(labelText: 'Password', border:
OutlineInputBorder()),
           obscureText: true,
         ),
         SizedBox(height: 20),
         SizedBox(
           width: double.infinity,
           child: ElevatedButton(
             onPressed: () {
               register();
                Navigator.push(
                       context,
                       MaterialPageRoute(
                               builder: (context) => UserLoginPage (), // Fixed
constructor reference
```

```
                    ),
                   );
              },
              child: Text('Submit'),
             ),
            ),
           ],
          ),
         ),
        ),
       ),
      );
     }
    }
```

Homepage.dart

```
import 'package:flutter/material.dart';

import 'package:supabase_flutter/supabase_flutter.dart';

import 'package:user/screen/cart.dart';

import 'package:user/screen/mybookings.dart';

import 'package:user/screen/login.dart';

import 'package:user/screen/productpage.dart';

import 'package:user/screen/settingsPage.dart';



class UserHomePage extends StatefulWidget {

  const UserHomePage({super.key});



  @override

  _UserHomePageState createState() => _UserHomePageState();

}
```

```
class _UserHomePageState extends State<UserHomePage> {

 final SupabaseClient supabase = Supabase.instance.client;

 List<Map<String, dynamic>> items = [];

 bool isLoading = true;



 @override

 void initState() {

  super.initState();

  fetchItems();

 }



 Future<void> fetchItems() async {

  try {

   final response = await supabase.from('tbl_item').select();

   setState(() {

    items = List<Map<String, dynamic>>.from(response);

    isLoading = false;
    });
   } catch (e) {
    print('Error fetching items: $e');
    setState(() {
     isLoading = false;
    });
   }
  }

  @override
  Widget build(BuildContext context) {
   return Scaffold(
    appBar: AppBar(
        title: Text('RentIt', style: TextStyle(fontSize: 24, fontWeight:
FontWeight.bold)),
     centerTitle: true,
     actions: [
      IconButton(
                icon: Icon(Icons.shopping_cart),
                onPressed: () {
                  Navigator.push(context, MaterialPageRoute(builder: (context) =>
CartPage()));
                },
              ),
            ],
          ),
          drawer: Drawer(
            child: ListView(
              padding: EdgeInsets.zero,
              children: [
                DrawerHeader(
                  decoration: BoxDecoration(color: Colors.blue),
                  child: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      CircleAvatar(
                        radius: 30,
                        backgroundColor: Colors.white,
                        child: Icon(Icons.person, size: 40, color: Colors.blue),
                      ),
                      SizedBox(height: 10),
                    Text('User Name', style: TextStyle(color: Colors.white, fontSize:
18)),
                        Text('user@example.com',   style:   TextStyle(color:
Colors.white70)),
                    ],
                  ),
                ),
                ListTile(
                  leading: Icon(Icons.home),
                  title: Text('Home'),
                  onTap: () {},
                ),
                ListTile(
                  leading: Icon(Icons.list),

                  title: Text('My Bookings'),

                  onTap: () {

                      Navigator.push(context, MaterialPageRoute(builder: (context)
=>Mybookings()));

                  },

                ),

                ListTile(

                  leading: Icon(Icons.settings),

                  title: Text('Settings'),

                  onTap: () {

                      Navigator.push(context, MaterialPageRoute(builder: (context)
=> SettingsPage()));

                  },

                ),

                ListTile(

                  leading: Icon(Icons.logout),

                  title: Text('Logout'),
                  onTap: () {
                    Navigator.pushAndRemoveUntil(
                      context,
                      MaterialPageRoute(builder: (context) => UserLoginPage()),
                      (route) => false,
                    );
                  },
                ),
              ],
            ),
          ),
          body: Padding(
            padding: EdgeInsets.all(16.0),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                  Text('Available Tools', style: TextStyle(fontSize: 22, fontWeight:
FontWeight.bold)),
                SizedBox(height: 10),
                Expanded(
                  child: isLoading
                      ? Center(child: CircularProgressIndicator())
                      : GridView.builder(
                        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
                          crossAxisCount: 2,
                          crossAxisSpacing: 10,
                          mainAxisSpacing: 10,
                          childAspectRatio: 0.8,
                        ),
                        itemCount: items.length,
                        itemBuilder: (context, index) {
                          final item = items[index];
                          return Card(
                            elevation: 4,
                            shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(12)),
child: Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
    Expanded(
      child: Container(
        decoration: BoxDecoration(
          borderRadius: BorderRadius.vertical(top: Radius.circular(12)),
          image: DecorationImage(
                image: NetworkImage(item['item_photo']),
                fit: BoxFit.cover,
          ),
        ),
      ),
    ),
    Padding(
      padding: EdgeInsets.all(8.0),
      child: Column(
                              crossAxisAlignment: CrossAxisAlignment.start,
                              children: [
                                Text(item['item_name'], style: TextStyle(fontSize:
16, fontWeight: FontWeight.bold)),
                                     Text('₹${item['item_rentprice']}/day', style:
TextStyle(fontSize: 14, color: Colors.green)),
                               SizedBox(height: 5),
                               SizedBox(
                                width: double.infinity,
                                child: ElevatedButton(
                                 onPressed: () {
                                  Navigator.push(
                                   context,
                                   MaterialPageRoute(
                                      builder: (context) => ProductPage(itemId:
item['item_id']),
                                   ),
                                  );
```

```
                                    },
                                    child: Text('Details'),
                                  ),
                                ),
                              ],
                            ),
                          ),
                        ],
                      ),
                    );
                  },
                ),
              ),
            ],
          ),
        ),
      );
```

```
 }

}
```

10.3.1 main.dart – SHOP

```
import 'package:flutter/material.dart';

import 'package:shop/screen/login.dart';

import 'package:shop/screen/shopHome.dart';

import 'package:supabase_flutter/supabase_flutter.dart';


Future<void> main() async {

  await Supabase.initialize(

    url: 'https://arixvhqgapwaxuycshaa.supabase.co',

                                                                  anonKey:
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJl
ZiI6ImFyaXh2aHFnYXB3YXh1eWNzaGFhIiwicm9sZSI6ImFub24iLCJpYXQ
iOjE3MzQzNDYzNjUsImV4cCI6MjA0OTkyMjM2NX0.dRr9mK8ug9UeIif_0
UskxKuXsNh8pDoBTST4ShPzTnA',
 );

 runApp(const MainApp());

}



final supabase = Supabase.instance.client;



class MainApp extends StatelessWidget {

 const MainApp({super.key});


 @override

 Widget build(BuildContext context) {

  return MaterialApp(

   debugShowCheckedModeBanner: false,

   home: const AuthWrapper(),

  );

 }

}
```

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

  @override
  Widget build(BuildContext context) {
    final session = supabase.auth.currentSession;

    if (session != null) {
      // If session exists, navigate to Home
      return const Shophome();
    } else {
      // If no session, navigate to Login
      return const ShopLogin();
    }
  }
}
```

shopHome.dart

```
import 'package:flutter/material.dart';

import 'package:shop/component/appbar.dart';

import 'package:shop/component/sidebar.dart';

import 'package:shop/screen/complaint.dart';

import 'package:shop/screen/dashboard.dart';

import 'package:shop/screen/manageBooking.dart';

import 'package:shop/screen/addProduct.dart';

import 'package:shop/screen/manageProduct.dart';




class Shophome extends StatefulWidget {

  const Shophome({super.key});


  @override

  State<Shophome> createState() => _ShopHomeState();
```

```
}



class _ShopHomeState extends State<Shophome> {

  int _selectedIndex = 0;



  final List<Widget> _pages = [

    Dashboard(),

    ManageBookingsPage(),

    addProduct(),

    ManageProductsPage(),

    ComplaintPage(),



    const Center(child: Text('Settings Content')),

  ];



  void onSidebarItemTapped(int index) {

    setState(() {
      _selectedIndex = index;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        backgroundColor: Color(0xFFFFFFFF),
        body: Row(
          children: [
            Expanded(
                flex: 1,
                child: SideBar(
                  onItemSelected: onSidebarItemTapped,
                )),
            Expanded(
              flex: 5,
              child: Column(
                children: [
                  Appbar1(),
                  _pages[_selectedIndex],
                ],
              ),
            )
          ],
        ));
  }
}
```

Addproduct.dart

```
import 'dart:io';
import 'dart:typed_data';
import 'package:flutter/material.dart';
import 'package:file_picker/file_picker.dart';
import 'package:shop/main.dart';

import 'package:supabase_flutter/supabase_flutter.dart';

import 'package:intl/intl.dart';



class addProduct extends StatefulWidget {

  const addProduct({super.key});



  @override

  _addProductState createState() => _addProductState();

}



class _addProductState extends State<addProduct> {

  final TextEditingController _nameController = TextEditingController();

  final TextEditingController _priceController = TextEditingController();

  final TextEditingController _detailsController = TextEditingController();



  String? _selectedCategory;
```

```
String? _selectedSubcategory;



List<Map<String, dynamic>> categories = [];

List<Map<String, dynamic>> subcategories = [];



@override

void initState() {

  super.initState();

  _fetchCategories();

}



PlatformFile? pickedImage;



Future<void> handleImagePick() async {

  FilePickerResult? result = await FilePicker.platform.pickFiles(

    allowMultiple: false, // Only single file upload

  );
```

```
    if (result != null) {
      setState(() {
        pickedImage = result.files.first;
      });
    }
  }


  Future<String?> photoUpload() async {
    try {
      final bucketName = 'shop'; // Replace with your bucket name
                                   final              timestamp              =
DateFormat('yyyyMMdd_HHmmss').format(DateTime.now());
      final fileExtension =
          pickedImage!.name.split('.').last; // Get the file extension
      final fileName =
          "${timestamp}.${fileExtension}"; // New file name with timestamp
      final filePath = fileName;
      await supabase.storage.from(bucketName).uploadBinary(
        filePath,
        pickedImage!.bytes!, // Use file.bytes for Flutter Web
          );

      final publicUrl =
          supabase.storage.from(bucketName).getPublicUrl(filePath);
      return publicUrl;
    } catch (e) {
      print("Error photo upload: $e");
      return null;
    }
  }

  Future<void> _fetchCategories() async {
    final response =
        await Supabase.instance.client.from('tbl_category').select();
    if (mounted) {
      setState(() {
        categories = List<Map<String, dynamic>>.from(response);
      });
    }
  }

  Future<void> _fetchSubcategories(int categoryId) async {
    final response = await Supabase.instance.client
        .from('tbl_subcategory')
        .select()
        .eq('category_id', categoryId);
    if (mounted) {
      setState(() {
        subcategories = List<Map<String, dynamic>>.from(response);
      });
    }
```

```
}



Future<void> _submitProduct() async {

 try {

   String? url = await photoUpload();



   final response = await Supabase.instance.client.from('tbl_item').insert({

     'item_name': _nameController.text,

     'category_id': int.parse(_selectedCategory!),

     'subcategory_id': int.parse(_selectedSubcategory!),

     'item_rentprice': double.parse(_priceController.text),

     'item_detail': _detailsController.text,

     'item_photo': url,

   });



   print("Product added successfully: $response");
```

```
        // Clear fields after insertion

        setState(() {

          _nameController.clear();

          _priceController.clear();

          _detailsController.clear();

          pickedImage = null;

          _selectedCategory = null;

          _selectedSubcategory = null;

        });

      } catch (e) {

        print("Error inserting product: $e");

      }

    }




      @override

      Widget build(BuildContext context) {

        return Padding(
        padding: const EdgeInsets.all(16.0),

        child: Card(

          elevation: 5,

                        shape:    RoundedRectangleBorder(borderRadius:
BorderRadius.circular(12)),

          child: Padding(

            padding: const EdgeInsets.all(16.0),

            child: SingleChildScrollView(

              child: Column(

                crossAxisAlignment: CrossAxisAlignment.start,

                children: [

                  const Text(

                    "ADD PRODUCT",

                    style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),

                  ),

                  const SizedBox(height: 20),

                  TextField(

                    controller: _nameController,
                    decoration: const InputDecoration(labelText: "Item Name"),
                  ),
                  DropdownButtonFormField(
                    value: _selectedCategory,
                    items: categories.map((category) {
                      return DropdownMenuItem(
                        value: category['category_id'].toString(),
                        child: Text(category['category_name']),
                      );
                    }).toList(),
                    onChanged: (value) {
                      if (value != null) {
                        setState(() {
                          _selectedCategory = value;
                          _fetchSubcategories(int.parse(value));
                          _selectedSubcategory = null;
                        });
                      }
                    },
                    decoration: const InputDecoration(labelText: "Category"),
                  ),
                  DropdownButtonFormField(
                    value: _selectedSubcategory,
                    items: subcategories.map((sub) {
                      return DropdownMenuItem(
                        value: sub['subcategory_id'].toString(),
                        child: Text(sub['subcategory_name']),
                      );
                    }).toList(),
                    onChanged: (value) {
                      if (value != null) {
                        setState(() {
                          _selectedSubcategory = value;
                        });
                      }
                    },
                    decoration: const InputDecoration(labelText: "Subcategory"),
                  ),
                  TextField(
                    controller: _priceController,
                    keyboardType: TextInputType.number,
                    decoration: const InputDecoration(labelText: "Rental Price"),
                  ),
                  TextField(
                    controller: _detailsController,
                    decoration: const InputDecoration(labelText: "Details"),
                    maxLines: 3,
                  ),
                  const SizedBox(height: 10),
                  SizedBox(
                    height: 120,
              width: 120,

              child: pickedImage == null

                ? GestureDetector(

                    onTap: handleImagePick,

                    child: Icon(

                      Icons.add_a_photo,

                      color: Color(0xFF0277BD),

                      size: 50,

                    ),

                  )

                : GestureDetector(

                    onTap: handleImagePick,

                    child: ClipRRect(

                      child: pickedImage!.bytes != null

                          ? Image.memory(

                              Uint8List.fromList(

                                  pickedImage!.bytes!), // For web
                              fit: BoxFit.cover,
                            )
                          : Image.file(
                              File(pickedImage!
                                  .path!), // For mobile/desktop
                              fit: BoxFit.cover,
                            ),
                    ),
                  ),
                ),
                Container(
              width: 100,
              height: 100,
              decoration: BoxDecoration(
                    border: Border.all(color: Colors.grey),
              ),
                ),
            const SizedBox(height: 10),

            ElevatedButton(

              onPressed: _submitProduct,

              child: const Text("Add Item"),

            ),

          ],

        ),

      ),

    ),

  ),

 );

}

}
```

Manange product.dart

```
import 'package:flutter/material.dart';

import 'package:shop/screen/product_details.dart';

import 'package:supabase_flutter/supabase_flutter.dart';
```

```
class ManageProductsPage extends StatefulWidget {
  const ManageProductsPage({super.key});


  @override
          State<ManageProductsPage>          createState()          =>
_ManageProductsPageState();
}


class _ManageProductsPageState extends State<ManageProductsPage> {
  List<Map<String, dynamic>> products = [];
  bool isLoading = true;


  @override
  void initState() {
    super.initState();
    _fetchProducts();
  }
```

```
Future<void> _fetchProducts() async {
  try {
    final response =
        await Supabase.instance.client.from('tbl_item').select();
    setState(() {
      products = List<Map<String, dynamic>>.from(response);
      isLoading = false;
    });
  } catch (e) {
    print("Error fetching products: $e");
    setState(() {
      isLoading = false;
    });
  }
}
```

```
Future<void> _deleteProduct(int productId) async {

try {

  final supabase = Supabase.instance.client;



  // Delete related stock entries first

  await supabase.from('tbl_stock').delete().eq('item_id', productId);



  // Now delete the product

  await supabase.from('tbl_item').delete().eq('item_id', productId);



  _fetchProducts(); // Refresh after delete

  ScaffoldMessenger.of(context).showSnackBar(

    const SnackBar(

      content: Text("Product deleted successfully!"),

      backgroundColor: Colors.red,

    ),

  );
```

```
    } catch (e) {
      print("Error deleting product: $e");
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text("Error deleting product: $e"),
          backgroundColor: Colors.red,
        ),
      );
    }
  }

    @override
    Widget build(BuildContext context) {
      return isLoading
          ? const Center(child: CircularProgressIndicator())
          : products.isEmpty
              ? const Center(child: Text("No products available"))
          : Padding(
            padding: const EdgeInsets.all(10.0),
            child: SingleChildScrollView(
              child: Column(
                children: [
                  GridView.builder(
                    physics: const NeverScrollableScrollPhysics(),
                    shrinkWrap: true,
                                                    gridDelegate:     const
SliverGridDelegateWithFixedCrossAxisCount(
                      crossAxisCount: 3,
                      crossAxisSpacing: 10,
                      mainAxisSpacing: 10,
                      childAspectRatio: 1,
                    ),
                    itemCount: products.length,
                    itemBuilder: (context, index) {
                      final product = products[index];
```

```
return GestureDetector(
  onTap: () {
      Navigator.push(context, MaterialPageRoute(builder:
(context) => ProductDetails(product: product),));
  },
  child: Card(
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(12),
    ),
    elevation: 4,
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        SizedBox(
          height: 150,
          width: double.infinity,
          child: ClipRRect(
                          borderRadius: const BorderRadius.vertical(top:
Radius.circular(12)),
                        child: Image.network(
                                      product['item_photo'] ??
'https://via.placeholder.com/150',
                          fit: BoxFit.cover,
                          errorBuilder: (context, error, stackTrace) {
                                      return const Center(child:
Icon(Icons.broken_image, size: 50));
                          },
                        ),
                      ),
                    ),
                    Padding(
                      padding: const EdgeInsets.all(8.0),
                      child: Column(
                        crossAxisAlignment: CrossAxisAlignment.start,
                        children: [
                                                    Text(
                                                      product['item_name'] ?? "No Name",
                                                                      style: const TextStyle(fontWeight:
FontWeight.bold, fontSize: 16),
                                                    ),
                                                    Text(
                                                        "Rent Price: \$${product['item_rentprice'] ??
'0.00'}",
                                                              style: const TextStyle(color: Colors.green,
fontSize: 14),
                                                    ),
                                                    Row(
                                                      mainAxisAlignment: MainAxisAlignment.end,
                                                      children: [
                                                        IconButton(
                                                      icon: const Icon(Icons.delete, color: Colors.red),
                                                                                  onPressed: () =>
_deleteProduct(product['item_id']),
                              ),
                            ],
                          ),
                        ],
                      ),
                    ),
                  ],
                ),
              ),
            );
          },
        ),
      ],
    ),
  ),
);
}
```