# 1. INTRODUCTION

Tinytots is an innovative platform designed to assist parents and caregivers in managing the daily needs and activities of their children through personalized care plans and high-quality, child-centric services. The system consists of a mobile application for parents and staff and a web-based interface for daycare administrators, ensuring a seamless experience across all stakeholders.

The Tinytots mobile application empowers parents to select daycare plans tailored to their child’s specific needs, whether for full-day care, after-school programs, or occasional drop-ins. By integrating key features such as activity customization, dietary preferences, and real-time updates, Tinytots prioritizes both child development and parental peace of mind. The app also incorporates an allergy condition filter, allowing parents to ensure their child’s safety and well-being while in daycare.

On the administrative side, the web-based interface enables daycare owners to manage operations efficiently. The platform facilitates child enrollment, staff scheduling, event planning, and secure payment processing, ensuring smooth and optimized daycare services. Daycare owners can monitor their facilities seamlessly, while staff can manage daily activities, attendance, and child progress reports efficiently.

Tinytots is more than just a daycare management system; it is a commitment to supporting child development and family well-being through technology. By offering a user-friendly mobile app for parents and a powerful web interface for daycare management, Tinytots revolutionizes the childcare industry, making quality daycare services more accessible, transparent, and reliable for everyone.

## 1.1 OBJECTIVE OF THE PROJECT

TinyTots is a cutting-edge mobile application tailored exclusively for baby daycare centers, offering a comprehensive solution to streamline operations, enhance communication, and prioritize the well-being and development of children. Designed as an integrated platform for administrators, staff, and parents, TinyTots ensures efficient management of daily activities, fostering a seamless experience for all stakeholders. Key features include child profile management, attendance tracking, daily activity logs, and customizable meal planning, enabling daycare centers to operate smoothly and efficiently. Parents benefit from real-time updates on their children's activities, meals, and progress, promoting transparency and trust. The app also includes event calendars, and learning progress, creating a safe, organized, and nurturing environment for children. By combining advanced technology with a child-centric approach, TinyTots not only supports the developmental needs of children but also addresses the operational challenges of daycare centers, making it an indispensable tool for modern childcare management.

# 2. SYSTEM ANALYSIS

System analysis is a step-by-step process used to identify and develop or acquire the software needed to control the processing of specific applications. System analysis is a continuing activity in the stages of the systems development. System analysis is the process of gathering and interpreting facts, diagnosing problems, and using the facts to improve the system. The outputs from the organization are traced through the various processing that the input phases through in the organization. This involves gathering information and using structured tools for analysis. A detailed study of this process must be made by various techniques like interviews; questionnaires etc.

## 2.1 EXISTING SYSTEM

The existing system of daycare management often relies heavily on manual processes, which can be time-consuming, error-prone, and inefficient. Daycare centers typically use paper-based records for child profiles, attendance tracking, and daily activity logs, making it difficult to organize and retrieve information quickly. Meal planning and dietary requirements are managed manually, increasing the risk of oversight, especially for children with allergies or specific needs. Staff schedules and payroll are often handled using spreadsheets or physical logs, which can result in scheduling conflicts or calculation errors. Additionally, tracking children's developmental progress and generating reports is a tedious task, requiring significant administrative effort. These manual processes not only create operational inefficiencies but also limit transparency for parents, who may feel disconnected from their child's daily activities. Overall, the lack of automation in traditional daycare systems hinders productivity, communication, and the ability to provide a consistently high standard of care.

### DISADVANTAGES

- **Time-Consuming Processes**: Manual record-keeping takes too much time and reduces staff efficiency.
- **Error-Prone Operations**: Paper-based systems lead to mistakes in attendance, meals, and payroll.
- **Inefficient Information Retrieval**: Finding child records and logs is slow without digital organization.
- **Scheduling Conflicts:** Staff schedules on paper or spreadsheets often cause miscommunication.

- **Lack of Automation**: Without automation, operations and communication are less efficient.
- **Difficulty in Reporting:** Creating reports takes a lot of effort and often gets delayed.
- **Operational Inefficiencies**: Manual work slows down daycare activities.
- **Inconsistent Care Standards**: Without a structured system, care quality may vary.
- **Limited Scalability**: As the daycare grows, manual systems become harder to manage.

## 2.2 PROPOSED SYSTEM

The proposed system is a modern mobile application designed to revolutionize daycare center operations by providing platforms for administrators, staff, and parents. This app streamlines daily tasks such as child profile management, attendance tracking, and activity scheduling, ensuring efficient and organized daycare operations. Parents can access real-time updates on their child's activities, meals, and developmental progress, fostering transparency and trust. Additionally, it offers learning progress reports to track children's developmental milestones. By integrating these tools, the app creates a safe, nurturing, and well-structured environment for children while simplifying administrative tasks for daycare centers. This innovative solution not only enhances the quality of childcare but also improves operational efficiency, making it an essential tool for modern daycare management.

**ADVANTAGES**

- **Time-Saving**: Automates record-keeping, attendance tracking, and scheduling, reducing manual effort.
- **Error Reduction**: Minimizes mistakes in attendance, meal planning, and payments.
- **Easy Information Access**: Provides quick access to child profiles, attendance logs, and activity records.
- **Better Meal Management**: Tracks dietary needs accurately, reducing the risk of errors.
- **Smooth Staff Scheduling**: Prevents scheduling conflicts and improves coordination.
- **Simplified Developmental Tracking**: Offers structured monitoring of children's progress.
- **Automated Operations**: Streamlines daycare management for better efficiency.
- **Improved Daycare Efficiency**: Eliminates bottlenecks, making daycare operations smoother.

## 2.3 SYSTEM REQUIREMENT SPECIFICATION

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 also minimizes the development cost. A good SRS defines how an application will interact with system hardware, other programs and human users in a wide variety of real worked situations.

**Problem to be Solved**

This system solves the manual errors happening in a daycare.

**Customer requirements**

- The system should be fast.
- User friendly.
- Maintaining the security of customer's data.
- Efficiency in data retrieval and appointment management

**What does the developer need to know?**

Must know the existing system and its drawbacks.
Must know what will be needed in the proposed system.

**Business Requirements**

The system should be feasible both for the developer and the client. It should be effective and should be able to be completed in time. The developer should be responsible for developing the system, installing the software, updating the software whenever necessary, and conducting any user training that may be needed for using the system.

**User Requirements**

The user requirement(s) specification is a document usually that specifies the requirements the user expects from software to be constructed in a software project.

The administrator has overall control of the system.

- Provide customer details.
- Faster processing

**Functional Requirements**

Functional requirements define what a system is supposed to do. The system should perform the following functionalities.

- Login – Login of admin, staff, and parent.
- Admission– Parents can register with the system and register for their child’s admission
- Edit profile– Users can update their profile.
- View status – Users can view the status of their activities in the app.
- Approve – The admin can approve the admission.
- Logout – System users can log out from the app.

### 2.3.1 HARDWARE SPECIFICATIONS

| | |
|---|---|
| Processor | : Intel core i5 or higher processor |
| Speed | : 2.50GHz or Higher |
| System bus | : 64 bits |
| Memory | : 16 GB RAM or Higher |
| Hard disk | : 40 GB or Higher |
| Monitor | : 15.6” FHD |
| Keyboard | : 104 Keys |
| Pointing Device | : Two or Three Button Mouse. |

### 2.3.2 SOFTWARE SPECIFICATIONS

| | |
|---|---|
| Operating System | : Windows 10 |
| Front End | : Flutter (Dart) |
| Back End | : Supabase |
| Development Tools | : Android Studio, Visual Studio Code |
| Platform | : Android, Web |
| Browser Compatibility | : Google Chrome |

### 2.3.3 FRONT END

**FLUTTER**

Flutter is an open-source UI software development kit created by Google. It is used to develop cross-platform applications from a single codebase, allowing developers to build high-performance, visually appealing apps for mobile, web, and desktop.

- Flutter uses the Dart programming language, which is optimized for fast app development.
- It provides a rich set of pre-designed widgets that help create responsive and attractive user interfaces.
- Flutter’s hot reload feature allows developers to see real-time updates without restarting the application.
- It ensures a smooth user experience with a high frame rate and native-like performance.

**Common Uses of Flutter**

- Flutter is used for building mobile applications that work on both Android and iOS.
- It enables web and desktop development using the same codebase.
- It is ideal for applications requiring rich UI elements and animations.
- It integrates seamlessly with Supabase for backend services such as authentication, cloud storage, and real-time databases.

**Characteristics of Flutter**

Flutter's advantages include:

- Cross-Platform Development – Develop once, deploy anywhere.
- Fast Development – Hot reload accelerates the development process.
- Beautiful UI – Uses Material Design and Cupertino widgets.
- High Performance – Delivers near-native performance.
- Strong Community Support – Backed by Google and a vast developer community

### 2.3.4 BACK END

**SUPABASE**

Supabase is an open-source backend-as-a-service (BaaS) platform that provides a scalable and powerful alternative to Firebase. It is built on PostgreSQL and offers real-time capabilities, authentication, and cloud storage.

- Supabase Authentication allows secure user login using email, Google, and other OAuth providers.

- Supabase Database is a managed PostgreSQL database with real-time capabilities.
- Supabase Storage enables file and media storage with easy access control.
- Supabase Functions provide serverless computing for custom backend logic.
- Supabase API allows seamless integration with Flutter applications

## 2.4 FEASIBILITY ANALYSIS

A feasibility study is an evaluation and analysis of the potential of the proposed project which is based on extensive investigation and research to give full comfort to the decision makers. Feasibility studies aim to objectively and rationally uncover the strengths and weaknesses of existing business of proposed venture, opportunities and threats as presented by the environment, the resources required to carry through, and ultimately the process for success. In its simplest terms, the two criteria to judge feasibility are cost required and value to attain. As such, a well-designed feasibility study should provide a historical background of the business or project, a description of the product or service, accounting statements, details of the operations and management, marketing research and policies, financial data, legal requirements, and tax obligations.

The four aspects of the feasibility study are:

- Technical feasibility
- Economic feasibility
- Operational feasibility
- Behavioural feasibility

**Technical Feasibility**

The technical feasibility centers on the existing system and to what extent it can support the proposed addition. The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. The minimum requirements of the system are met by the average user. The developer system has á modest technical requirements as only minimal or null changes are required for implementing the system.

Normally associated with the technical feasibility includes:

- Development risk
- Resource availability
- Technology

The proposed system can work without any additional hardware or software support other than the computer system and networks. So, I analyzed that the proposed system is much more technically feasible than other systems when compared with the benefits of the new system.

### Economic Feasibility

Economic feasibility analysis is also known as cost/benefit analysis. The purpose of the economic feasibility assessment is to determine the positive economic benefits to the organization that the proposed system will provide. The proposed system reduces the operating cost in terms of time by automating the process. This system is economically feasible.

### Operational Feasibility

Operational feasibility is a measure of how well a proposed system solves problems and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.

### Behavioral Feasibility

People are inherently resistant to changes, and computer is known for facilitating the changes.

An estimate should be made to how strongly the users react toward the e development of the system. The proposed system consumes less time. Thus, the people are made to engage in some other important work.

## 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 the flowchart as it shows the data flow instead of the control flow of the program. A data flow diagram can also be used for the visualization of data processing (structured design).

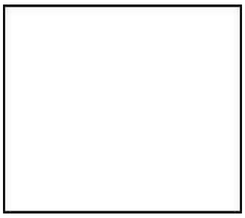
Data flow diagrams were invented by Larry Constantine, the original developer of structured design based on Martin and Estrin's "data flow graph" model of computation.

Data flow diagrams (DFDs) are one of the three essential perspectives of Structured System Analysis and Design Method SSADM. The sponsor of a project and the end users will 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, and 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 draw comparisons to implement a more efficient system. Data flow diagrams can be used to provide the end user with a physical idea of where the data they input ultimately has an effect on the structure of the whole system from order to dispatch to report. How any system is developed can be determined through a data flow diagram.

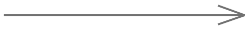
Developing a data flow diagram helps in identifying the transaction data in the data model. There are different notations to draw data flow diagrams, defining different visual representations for processes, data stores, data flow, and external entities. The first step is to draw a data flow diagram (DFD). A DFD also known as a "bubble chart" has the purpose of clarifying system requirements and identifying major transformations that will become programs in system design. So, it is the starting point of the design phase that functionally decomposes the requirements specification down to the lowest level of details DFD consists of a series of bubbles joined by lines. The bubbles represent data transformation and the Iines represent data flow in the system.

**DFD Symbols: -**

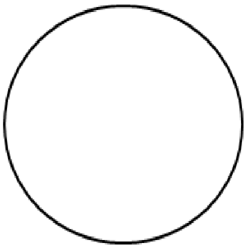
- Square- Defines the source or destination of the system.



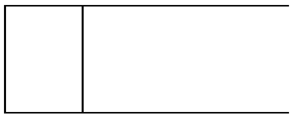
- Data flow - Identifies data flow Circle.



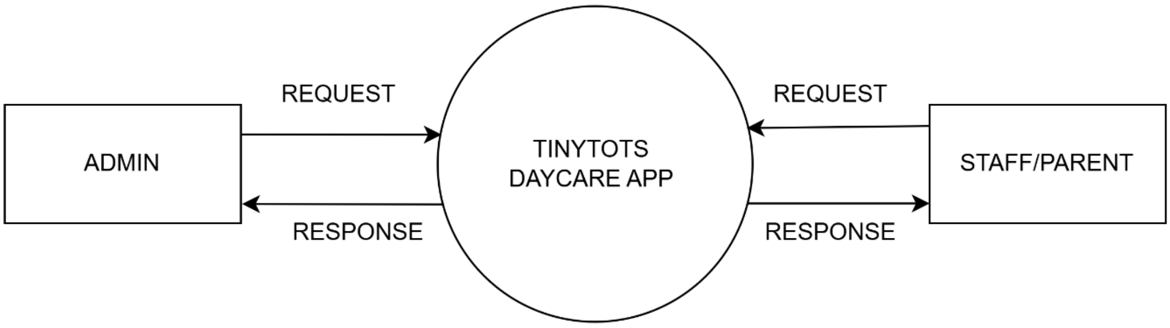
- Bubble - Represents a process that transforms incoming data to outgoing data.



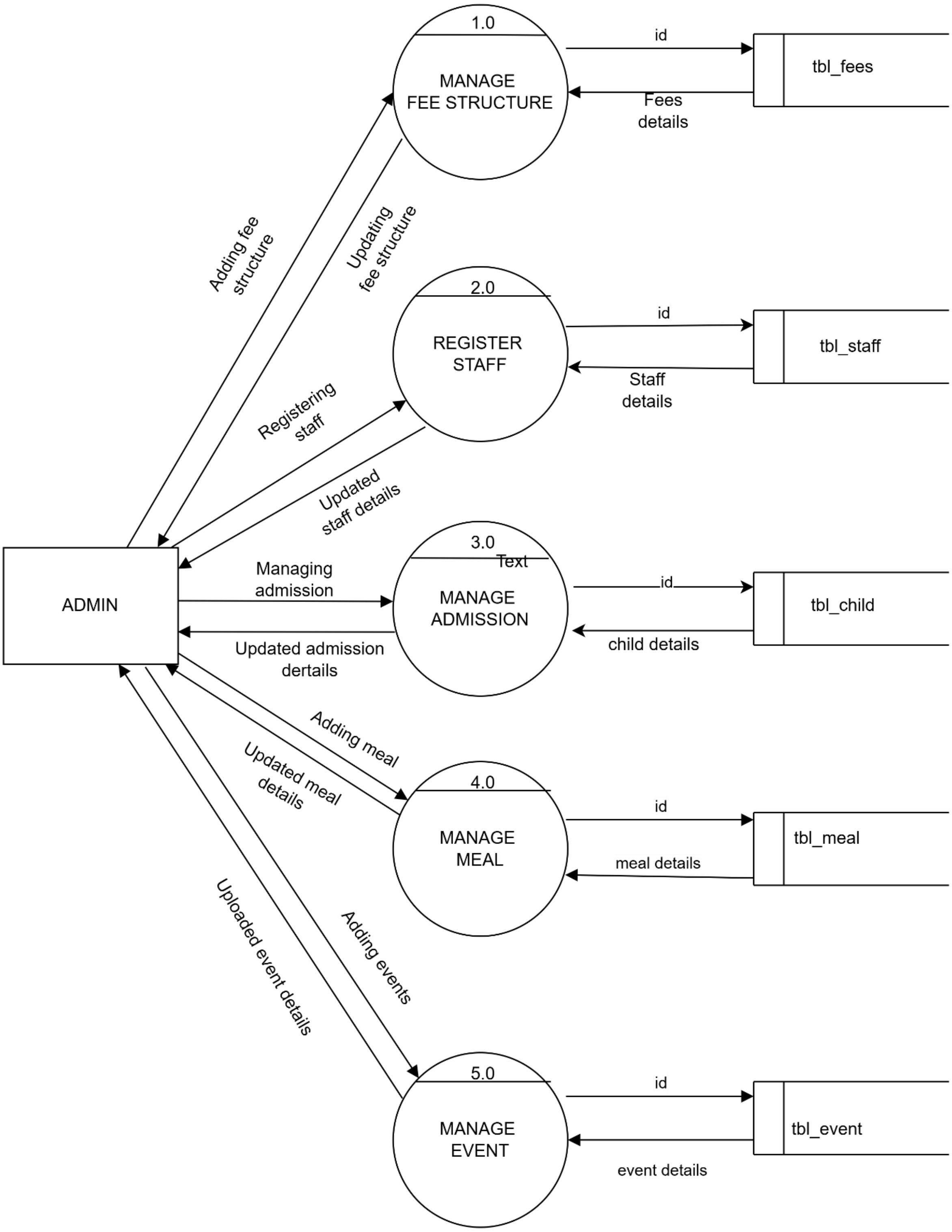
- Open rectangle- Data store

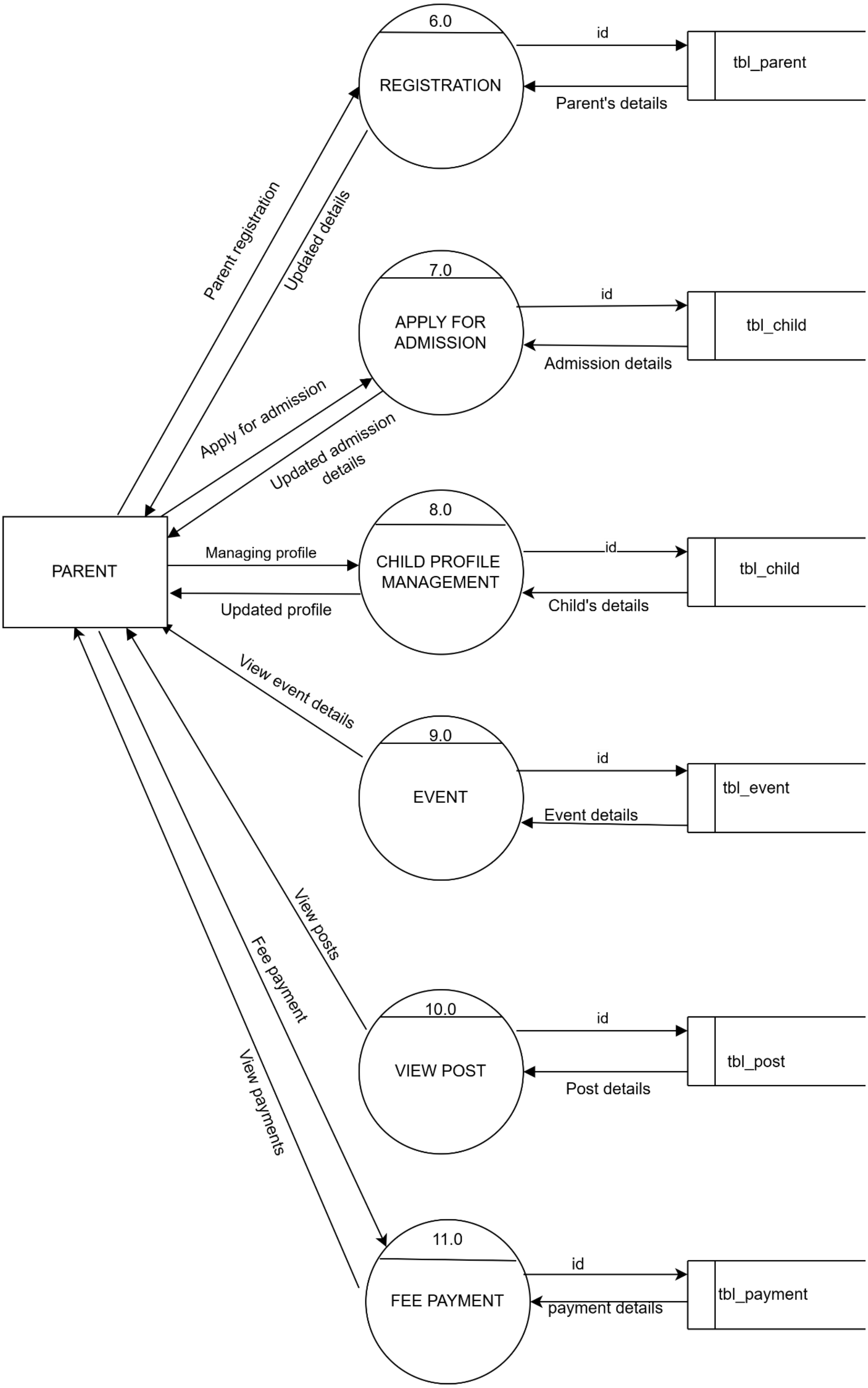


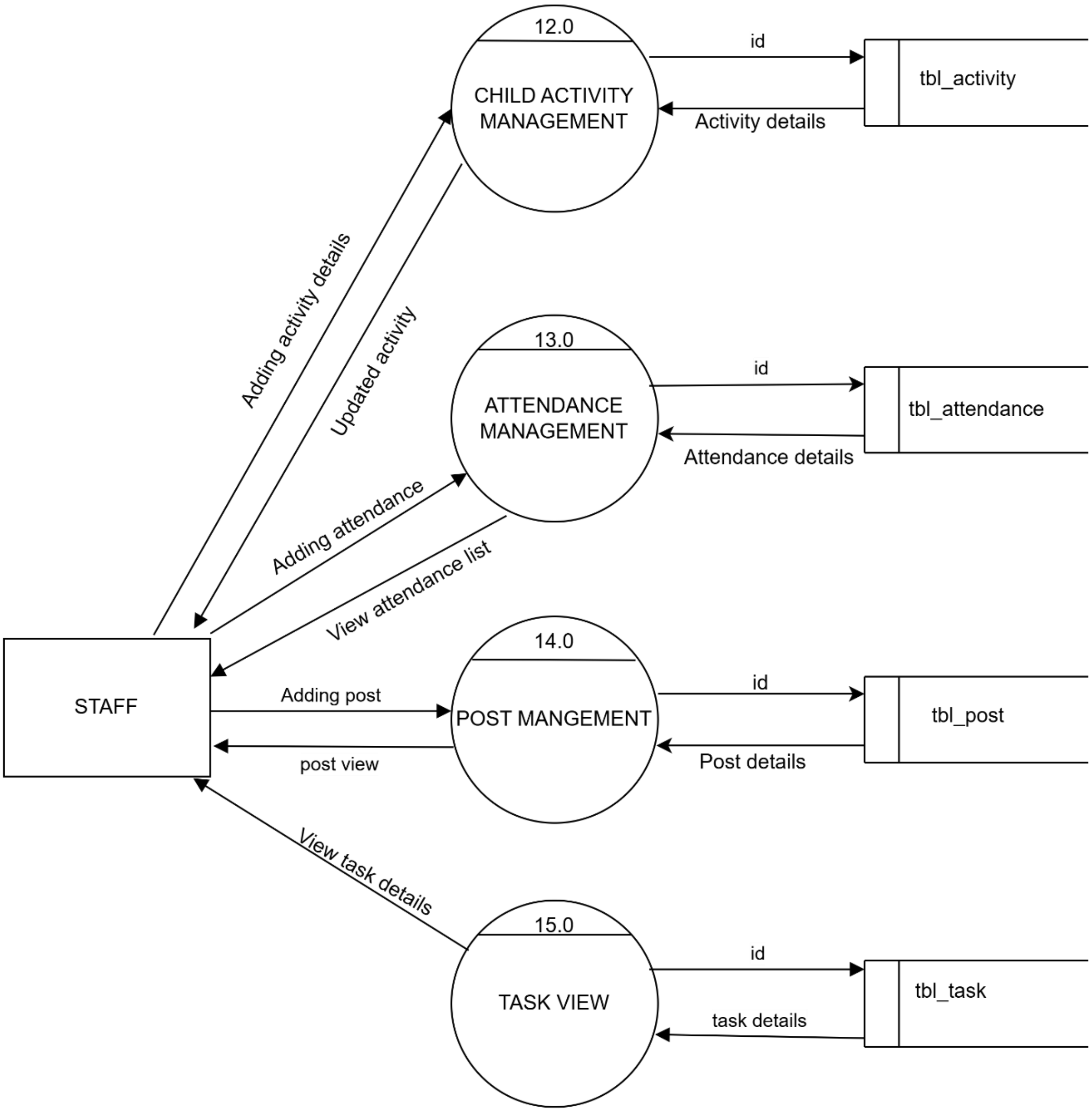
# LEVEL 0


ADMIN
REQUEST
RESPONSE
TINYTOTS
DAYCARE APP
REQUEST
RESPONSE
STAFF/PARENT

# LEVEL 1


1.0
MANAGE FEE STRUCTURE
id
tbl_fees
Fees details
Adding fee structure
Updating fee structure
2.0
REGISTER STAFF
id
tbl_staff
Staff details
Registering staff
Updated staff details
3.0
Text
MANAGE ADMISSION
id
tbl_child
child details
ADMIN
Managing admission
Updated admission dertails
Adding meal
Updated meal details
4.0
MANAGE MEAL
id
tbl_meal
meal details
Uploaded event details
Adding events
5.0
MANAGE EVENT
id
tbl_event
event details


6.0
REGISTRATION
id
tbl_parent
Parent's details
Parent registration
Updated details
7.0
APPLY FOR ADMISSION
id
tbl_child
Admission details
Apply for admission
Updated admission details
PARENT
Managing profile
Updated profile
8.0
CHILD PROFILE MANAGEMENT
id
tbl_child
Child's details
View event details
9.0
EVENT
id
tbl_event
Event details
View posts
Fee payment
View payments
10.0
VIEW POST
id
tbl_post
Post details
11.0
FEE PAYMENT
id
tbl_payment
payment details


12.0
CHILD ACTIVITY MANAGEMENT
id
tbl_activity
Activity details
Adding activity details
Updated activity
13.0
ATTENDANCE MANAGEMENT
id
tbl_attendance
Attendance details
Adding attendance
View attendance list
14.0
POST MANGEMENT
id
tbl_post
Post details
STAFF
Adding post
post view
View task details
15.0
TASK VIEW
id
tbl_task
task details

# 3. SYSTEM DESIGN

## 3.1 INPUT DESIGN

The quality of the system input determines the quality of the system output. Input specification describes the way data enter the system for processing. Input design features can ensure the reliability of the system and produce results from accurate data, or they can result in the production or erroneous information. The input design also determines whether the user can interact efficiently with the system.

In our system almost all inputs are being taken from the databases. To provide adequate inputs we have to select necessary values from the databases and arrange it to the appropriate controls.

**ADMIN**

Admin is the one who controls the system. The admin can access the page using a username and password. Admin can register staff, approve admission, add events, and meal management.

**STAFF**

Staff is the one who updates attendance, activities, and posts. Staff can access the system using the provided password and email.

**PARENT**

Parents can register children, can access profiles using email and password.

## 3.2 OUTPUT DESIGN

One of the important features of an information system for users is the output produced. Output is the information delivered to users through the information system. Without the quality of the output, the entire system appears to be unnecessary and users will avoid using it. Users generally merit the system solely by its output. In order to create the most useful output possible. One works closely with the user through an interactive process. Until the result is considered to be satisfactory.

## 3.3 TABLE DESIGN

The data design transforms the information domain model created during analysis into the data structures that will be required to implement the software. The data objects and relationships defined in the entity relationship diagram and the detailed data content depicted in the data dictionary provide the basis for the data design activity.

The overall objective in the development of database technology has been to treat data as an organizational resource and as an integrated whole. Database Management System allows data to be protected and organized separately from other resources. A database is an integrated collection of data. This is the difference between logical and physical data.

The organization of data in the database aims to achieve three major objectives:

- Data integration
- Data integrity
- Data independence

The databases are implemented using a DBMS package. Each DBMS has unique characteristics and general techniques for database design. There are 6 major steps in the design process. The first 5 steps are usually done on paper and finally, the design is implemented.

- Identify the table and relationships.
- Identify the data that is needed for each table and relationship.
- Resolve the relationship.
- Verify the design.
- Implement the design.

The database uses tables for storage. A table also contains records, which is a set of fields.

All records, in a table have the same set of fields with different information. Uses 3 tables.

Each table contains key fields that establish relationships in the database and how the records are stored. There are primary key fields that uniquely identify a record in a table. There are also fields that contain the primary key from another table called foreign keys.

## Table Design

1. Table name: tbl_admin

Primary key: admin_id

Description: Stores the details of admin.

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| id | UUID | -- | Unique id of admin |
| admin_name | TEXT | -- | Name of admin |
| admin_email | TEXT | -- | Email id of admin |
| admin_pwd | TEXT | -- | Password of admin |

2.Table name: tbl_staff

Primary key: staff_id

Description: Stores the details of staff.

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| id | UUID | -- | Unique id of staff |
| staff_name | TEXT | -- | Name of staff |
| staff_email | TEXT | -- | Email id of staff |
| staff_contact | TEXT | -- | Contact of staff |
| staff_pwd | TEXT | -- | password |
| staff_photo | TEXT | -- | Staff photo |
| staff_address | TEXT | -- | Address of staff |
| status | INT | -- | Show active or inactive |

3. Table name: tbl_attendance

Primary key: attendance_id

Description: Stores the details of  attendance

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| id | INT | -- | Unique identifier for each attendance record. |
| child_id | INT | -- | References the child ID in the childTable. |
| date | DATE | -- | The date of the attendance record. |
| check_in | TEXT | -- | The time of checked in. |
| check_out | TEXT | -- | The time of checked out. |
| staff_id | UUID | -- | References the staff ID in the Staff Table |
| role | TEXT | -- | Determines the role, whether it is child or staff |

4. Table name: tbl_parent

Primary key: parent _id

Description: Stores the details of parent

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| id | UUID | -- | Unique id of parent |
| parent_name | TEXT | -- | Parent’s name |
| parent_email | TEXT | -- | Email |
| parent_contact | TEXT | -- | Contact |
| parent_address | TEXT | -- | Address |
| parent_photo | TEXT | -- | Photo |
| parent_pwd | TEXT | -- | Password |
| parent_proof | TEXT | -- | Proof |

5. Table name: tbl_event

Primary key: id

Description: Stores the details of the event

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| id | INT | -- | Unique id of event |
| event_name | TEXT | -- | Name of event |
| event_date | TEXT | -- | Date of event |
| event_details | TEXT | -- | Event details |
| event_photo | TEXT | -- | Event photo |

6.TableName:tbl_meal
PrimaryKey:id
Description: Logs dietary updates, meal consumption, and meals served to children

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| Id | INT | -- | Id of meal record |
| meal_name | TEXT | -- | Description of meal served |
| meal_day | TEXT | -- | Meal day |

7. Table name: tbl_payment

Primary key: id

Description: Stores the details of the payment

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| id | INT | -- | Unique id of payment |
| child_id | INT | -- | Unique id of child |
| amount_due | TEXT | -- | Due amount |
| due_date | TEXT | -- | Due date |
| status | INT | -- | Payment status |
| payment_method | TEXT | -- | Payment method |
| payment_date | TEXT | -- | Date of payment |

8. Table name: tbl_activity

Primary key: id

Description: Stores the details of the activities

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| id | INT | -- | Unique identifier for each activity record. |
| child_id | INT | -- | Unique id of child |
| feeding_details | TEXT | -- | Details of the child's feeding for the day. |
| nap_schedule | TEXT | -- | Information on the child's nap times. |
| play_time_activities | TEXT | -- | Activities the child engaged in during playtime. |
| learning_activities | TEXT | -- | Educational or learning activities performed. |
| staff_id | UUID | -- | Unique id of staff |

9.TableName:tbl_post

PrimaryKey:id

Description: stores details of shared media

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| id | INT | -- | Unique identifier for each media entry. |
| staff_id | UUID | -- | ID of the staff sharing the media. |
| post_title | TEXT | -- | Title of the post |
| post_file | TEXT | -- | Type of media (e.g., photo, video) |
| created_at | TEXT | -- | Date and time the media was shared |

10.TableName:tbl_task

PrimaryKey:id

Description: store details of  staff schedule

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| id | INT | -- | Unique identifier for each schedule entry. |
| staff_id | UUID | -- | ID of the staff member assigned the task. |
| task | TEXT | -- | Detailed description of the assigned task. |
| created_at | TIMESTAMP | -- | Date the task is assigned. |
| end_date | TEXT | -- | Scheduled end time for the task. |
| status | INT | -- | Status of the task (e.g., Pending, Completed, In Progress). |

11.Table name:tbl_participant

Primary key:id

Description:store participant details

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| id | Int | -- | Participant id |
| child_id | Int | -- | Id of child |
| status | int | -- | Status of participant |
| event_id | Int | -- | Id of event |

12. Table name: tbl_fees

Primary key: id

Description: store fees details

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| id | INT | -- | Id of fees |
| fee_amount | TEXT | -- | Amount |
| fee_details | TEXT | -- | details |
| fee_name | TEXT | -- | Name |

13. Table name: tbl_like

Primary key: id

Description: stores the likes

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| id | INT | -- | Id of likes |
| post_id | INT | -- | Id of post |
| staff_id | UUID | -- | Id of staff |
| parent_id | UUID | -- | Id of parent |

14. Table name: tbl_child

Primary key: id

Description: Stores the details of child

| FIELD NAME | DATA TYPE | SIZE | DESCRIPTION |
|---|---|---|---|
| id | INT | -- | Unique id of child |
| name | TEXT | -- | Chil’s name |
| age | TEXT | -- | Age |
| gender | TEXT | -- | Gender |
| dob | DATE | -- | Date of birth |
| status | INT | -- | Admission status |
| parent_id | UUID | -- | Unique id of parent |
| fee_type | INT | -- | Type of fee |
| allergy | TEXT | -- | Allergy details |
| documents | TEXT | -- | Documents |
| doj | TEXT | -- | Date of join |
| photo | TEXT | -- | Photo of child |

# 4. SYSTEM IMPLEMENTATION AND TESTING

## 4.1 SYSTEM TESTING

Testing is the process of examining the software to compare the actual behavior with that of the excepted behavior. The major goal of software testing is to demonstrate that faults are not present. In order to achieve this goal, the tester executes the program with the intent of finding errors. Though testing cannot show the absence of errors by not showing their presence it is considered that these are not present.

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. 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 system 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 if all the parts of the system are correct the goal will be successfully achieved. A series of tests are performed for the proposed system before the proposed system is ready for user acceptance testing. The testing steps are,

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

System Testing provides the file assurance that software once validated must combined with all other system elements. System testing verifies whether all elements have been combined properly and that overall system function and performance are 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.

### 1. 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.

In unit testing

- Module interface was tested to ensure that information properly flows into and out of the program under 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 have been executed at least once.
- Error handling paths were also tested.

### 2. 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 too.th tn. using bottom-up integration i.e., by integrating from moving from the bottom to the top of 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 (group of modules) were constructed corresponding to the successful testing of user interaction, data manipulation analysis, display processing, and database management.

### 3. 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 the validation testing and it is checked whether the errors are being considered. Incorrect values are to be discarded. The errors are rectified.

In “TinyTots " 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.

**4. 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.

**5. Acceptance testing**

Acceptance involves running a suite of tests on the completed system. Each individual test, known as a Case, exercises a particular operating condition of the 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 in preparing for implementation is the education of users. which should really have taken place much carrier in the project when they were belong involved in the investigation and design work. During the implementation phase system takes physical shape. In order 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 into operation for a period of 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 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 changeover.
- 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 performs the implementation by creating the database and connecting to the server. After that the system is Checked, whether it performs all the transactions correctly. Then databases are cleared and made it to be usable to the technicians.

Implementation Plans

The following are the steps involved in the implementation plan of " TinyTots daycare system":

- Test system with sample data
- Detection and correction of errors
- Make the necessary changes in the system.
- Check the existing system.
- Installation of hardware and software utilities
- Training and involvement of user personnel

# 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.

**SYSTEM INTEGRITY** refers to the power functioning of hardware and programs, appropriate physical security and safety against external threats such as eavesdropping and wiretapping.

**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 need for protection.

**SECURITY IN SOFTWARE** System security refers to various validations of 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 system.

The system employees 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 load 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 fields that are mandatory 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 inform the user that some invalid operation has been performed or the performed operation is restricted. Some of the server-side checks imposed are:

- 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 replicate the primary value results in a message intimating the user about those values through the forms using foreign keys, which can be updated only with the existing foreign key values.
- User is intimating through appropriate messages about the successful operations or exceptions occurring at 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 on 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 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 wearing out are not significant and the cost of this maintenance is not great.

### 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 our requirements. This is based on the premise that as the software becomes useful, the user experiments with new cases beyond the of

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.

### Adaptive Maintenance

Modification of a software product performed after delivery to keep a product usable, changed or changing environment. Adaptive maintenance includes any work initiated because 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.

### 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 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

In the future, the TinyTots daycare app can be enhanced by integrating advanced technologies such as Artificial Intelligence (AI) and the Internet of Things (IoT) to further streamline operations and improve childcare quality. AI-powered features could include predictive analytics for identifying developmental delays or behavioral patterns, enabling early intervention. IoT devices, such as smart wearables for children, could monitor health metrics like temperature or activity levels in real-time, ensuring immediate response to any concerns. Additionally, the app could incorporate augmented reality (AR) for interactive learning experiences, making education more engaging for children. A blockchain-based system could be introduced to enhance data security and transparency for sensitive information like medical records or payment details. Furthermore, multilingual support and accessibility features could make the app inclusive for diverse users. By leveraging these innovations, TinyTots can evolve into a comprehensive, tech-driven platform that not only simplifies daycare management but also enriches the overall experience for children, parents, and staff.

# 8. CONCLUSION

In conclusion, the TinyTots daycare app represents a transformative solution for modern childcare management, addressing the inefficiencies of traditional systems while fostering transparency, communication, and operational excellence. By automating administrative tasks, providing real-time updates, and enhancing parental engagement, the app creates a seamless experience for all stakeholders. Future enhancements, such as AI-driven analytics, IoT integration, AR-based learning, and blockchain security, promise to elevate the app's capabilities further, ensuring a safer, more engaging, and data-driven environment for children. These advancements will not only streamline daycare operations but also enrich the quality of care and education provided. TinyTots is poised to become an indispensable tool for daycare centers, empowering them to deliver consistent, high-quality childcare while adapting to the evolving needs of families and educators in a technology-driven world.

## 9. BIBLIOGRAPHY

**BOOKS**:

- Marco L. Napoli, *Beginning Flutter: A Hands-On Guide to App Development*
- Eric Windmill, *Flutter in Action*
- Marco L. Napoli, *Mastering Dart: Advanced Techniques for Modern Applications*
- Slobodan Stojanović, *Serverless Applications with Node.js*
- Shyam Seshadri, *Building Scalable Apps with Firebase*
- Elias M. Awad, *System Analysis and Design*

**Websites:**

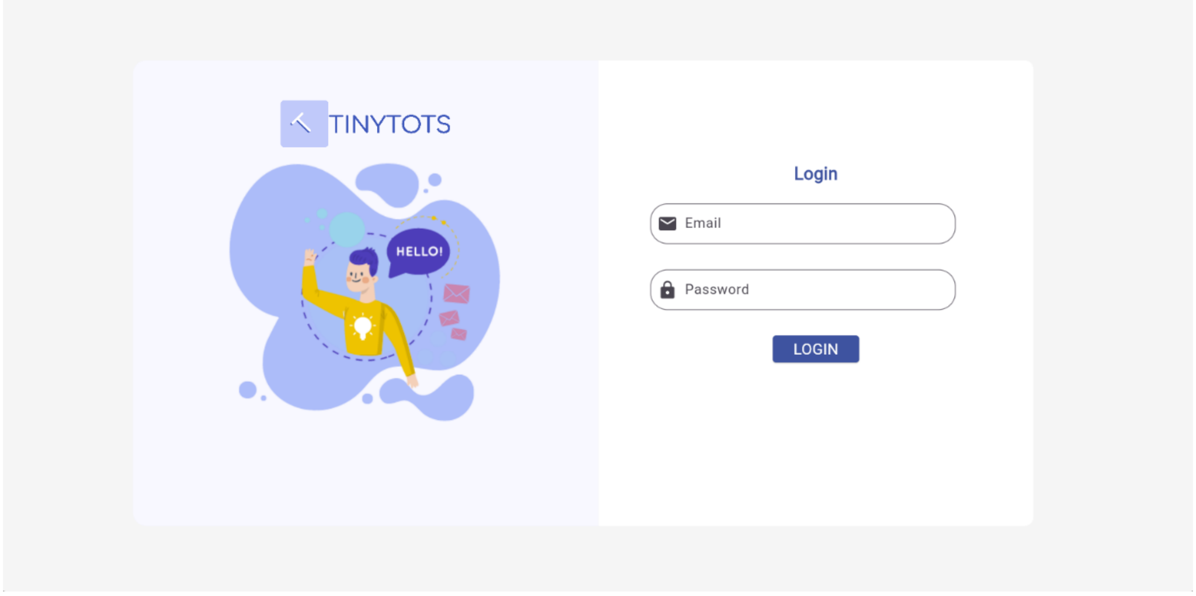
- https://flutter.dev
- https://dart.dev
- https://firebase.google.com
- https://supabase.com
- https://flutterawesome.com
- https://developer.android.com
- https://medium.com
- https://stackoverflow.com
- https://pub.dev

# 10. APPENDIX

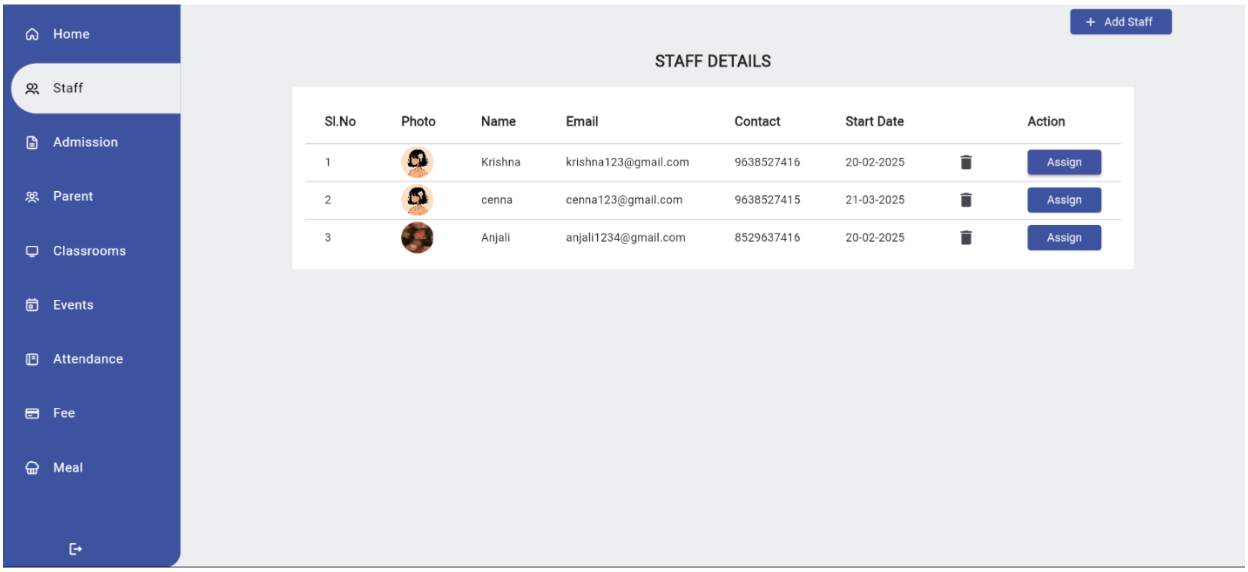
## 10.1 SCREENSHOTS

### ADMIN

### LOGIN



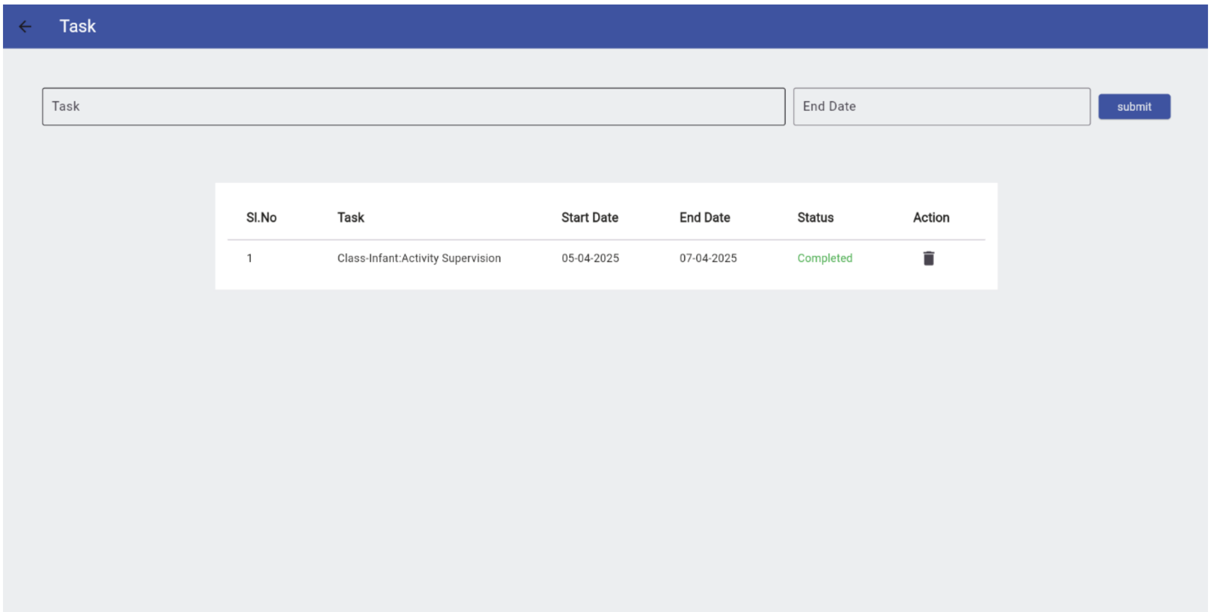
### REGISTERING STAFF



STAFF DETAILS

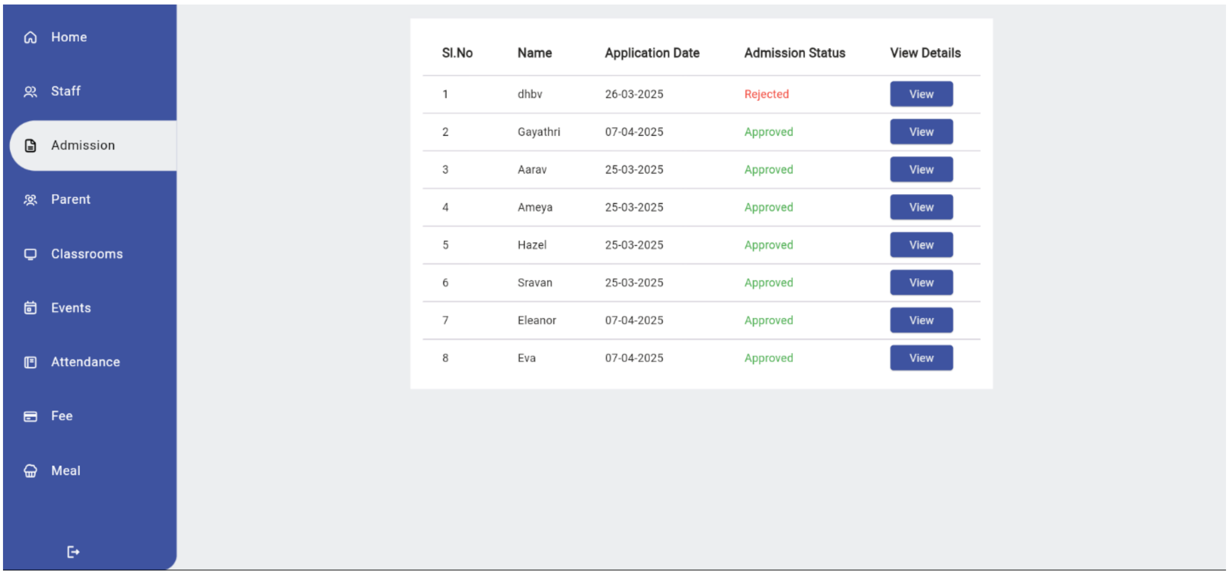
| Sl.No | Photo | Name | Email | Contact | Start Date | | Action |
|---|---|---|---|---|---|---|---|
| 1 | | Krishna | krishna123@gmail.com | 9638527416 | 20-02-2025 | 🗑 | Assign |
| 2 | | cenna | cenna123@gmail.com | 9638527415 | 21-03-2025 | 🗑 | Assign |
| 3 | | Anjali | anjali1234@gmail.com | 8529637416 | 20-02-2025 | 🗑 | Assign |

## ASSIGNING TASK



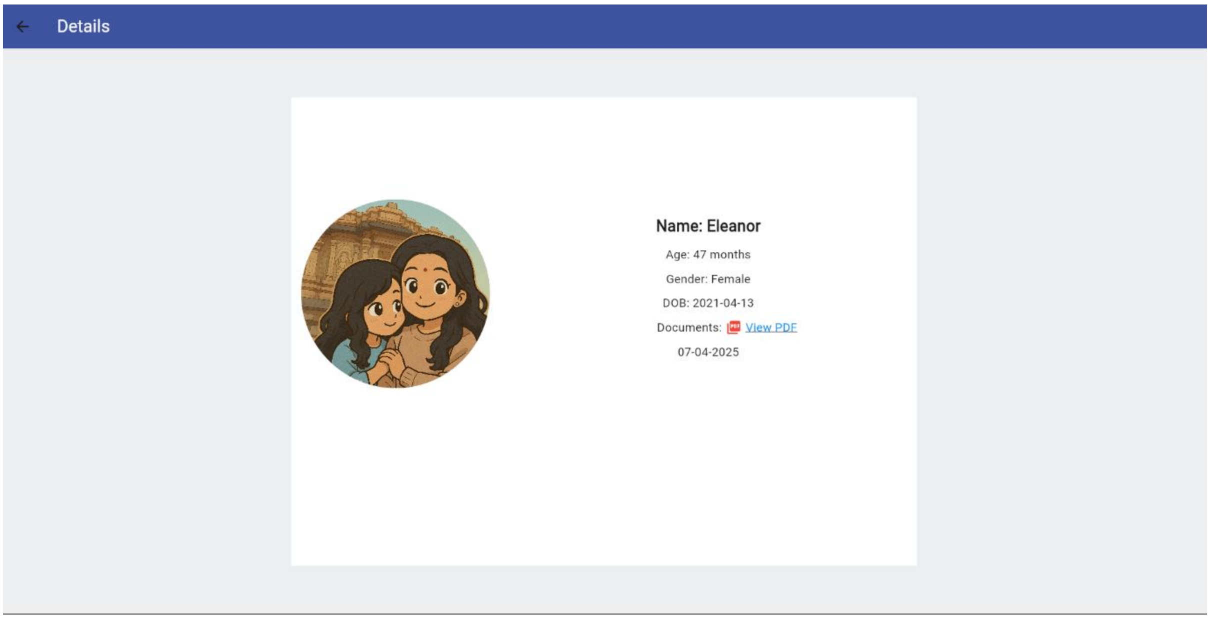
| Sl.No | Task | Start Date | End Date | Status | Action |
|---|---|---|---|---|---|
| 1 | Class-Infant:Activity Supervision | 05-04-2025 | 07-04-2025 | Completed | 🗑 |

## ADMISSION APPROVAL



| Sl.No | Name | Application Date | Admission Status | View Details |
|---|---|---|---|---|
| 1 | dhbv | 26-03-2025 | Rejected | View |
| 2 | Gayathri | 07-04-2025 | Approved | View |
| 3 | Aarav | 25-03-2025 | Approved | View |
| 4 | Ameya | 25-03-2025 | Approved | View |
| 5 | Hazel | 25-03-2025 | Approved | View |
| 6 | Sravan | 25-03-2025 | Approved | View |
| 7 | Eleanor | 07-04-2025 | Approved | View |
| 8 | Eva | 07-04-2025 | Approved | View |

## DETAILS OF CHILD



Name: Eleanor

Age: 47 months

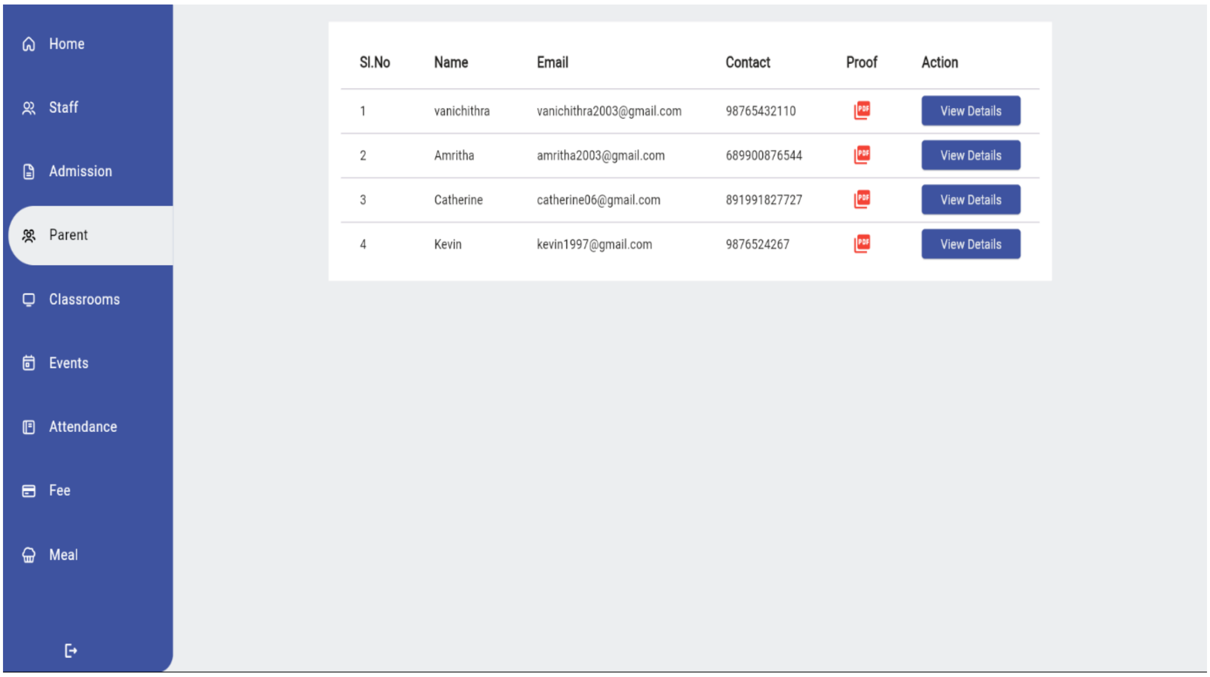
Gender: Female

DOB: 2021-04-13

Documents: View PDF

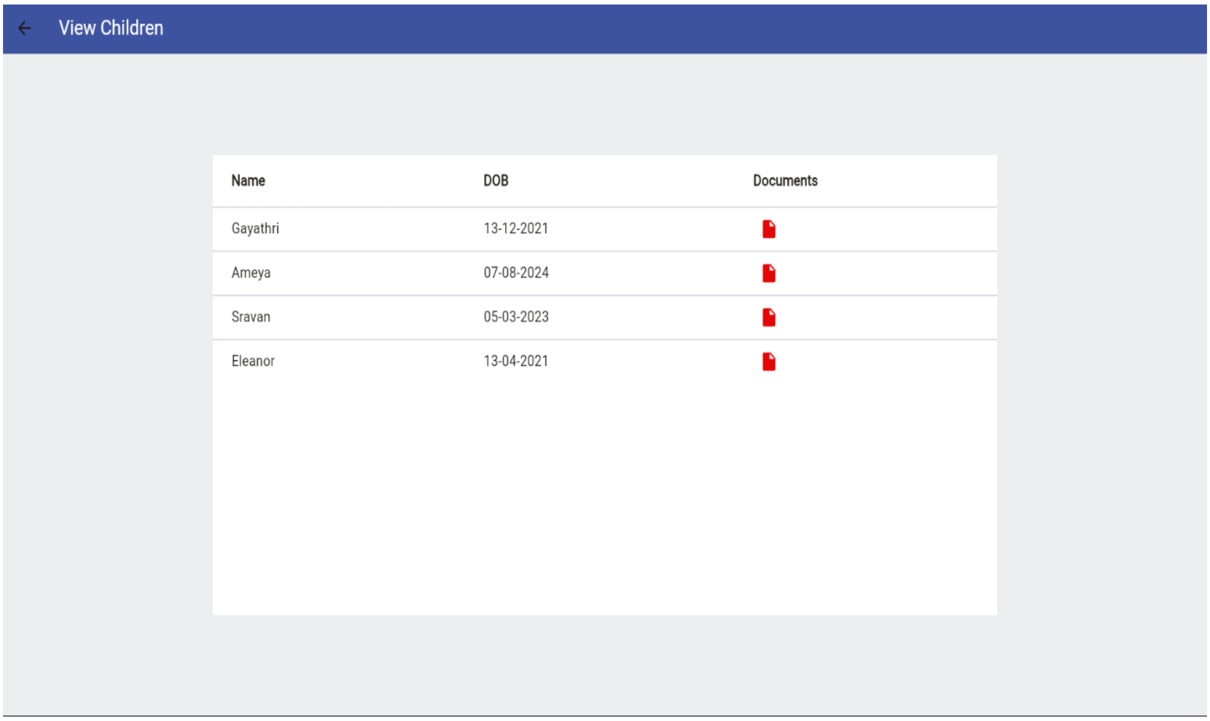
07-04-2025

## LIST OF PARENTS



| Sl.No | Name | Email | Contact | Proof | Action |
|---|---|---|---|---|---|
| 1 | vanichithra | vanichithra2003@gmail.com | 98765432110 | PDF | View Details |
| 2 | Amritha | amritha2003@gmail.com | 689900876544 | PDF | View Details |
| 3 | Catherine | catherine06@gmail.com | 891991827727 | PDF | View Details |
| 4 | Kevin | kevin1997@gmail.com | 9876524267 | PDF | View Details |

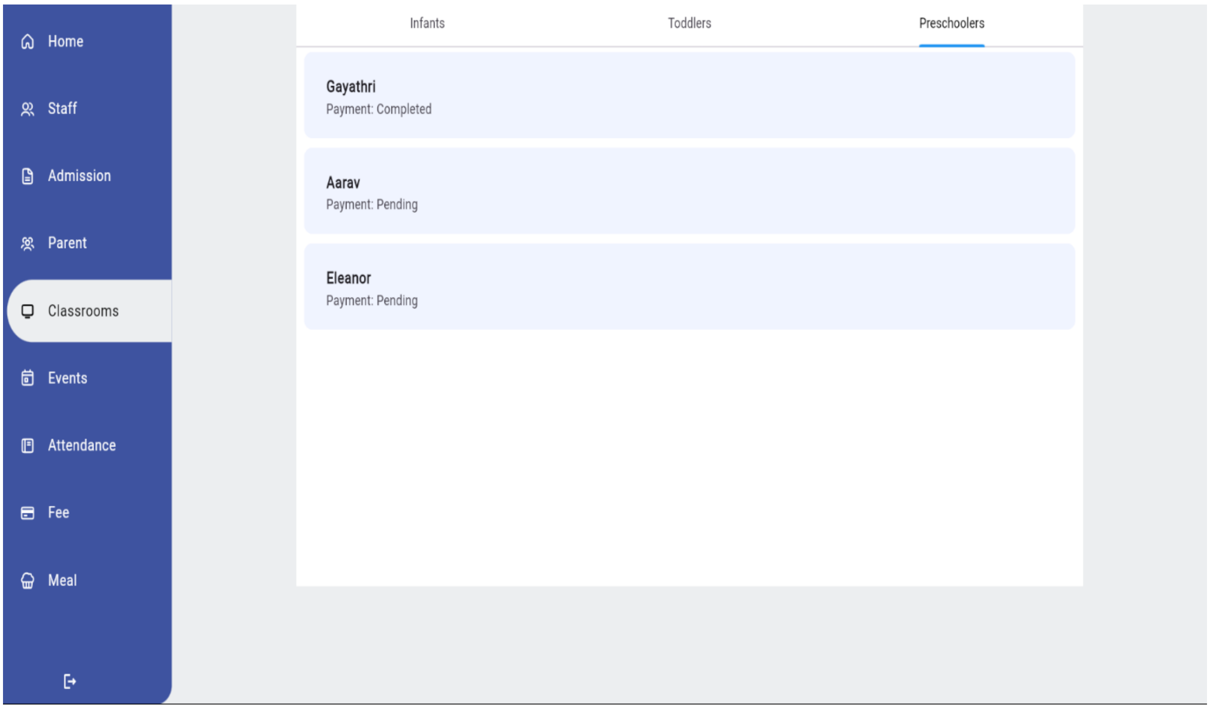
## CHILDREN OF PARTICULAR PARENT



View Children

| Name | DOB | Documents |
|---|---|---|
| Gayathri | 13-12-2021 | ▮ |
| Ameya | 07-08-2024 | ▮ |
| Sravan | 05-03-2023 | ▮ |
| Eleanor | 13-04-2021 | ▮ |

## CLASSROOMS

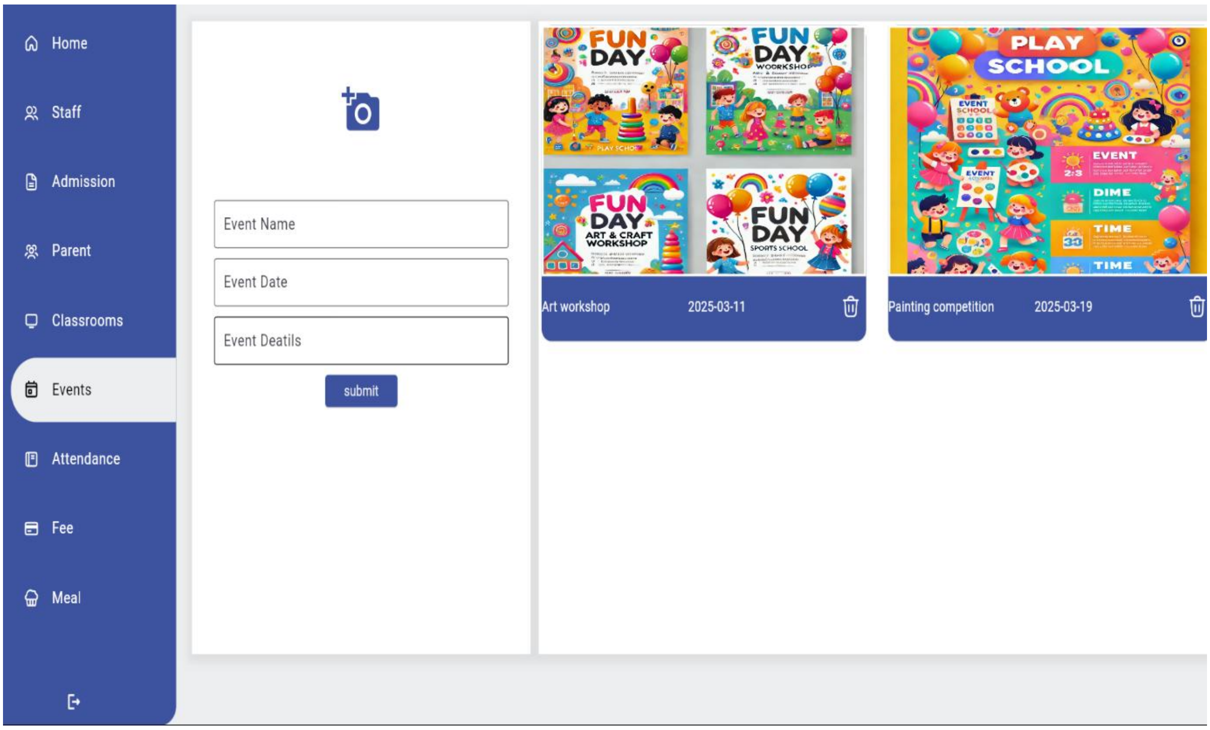


Gayathri
Payment: Completed

Aarav
Payment: Pending

Eleanor
Payment: Pending

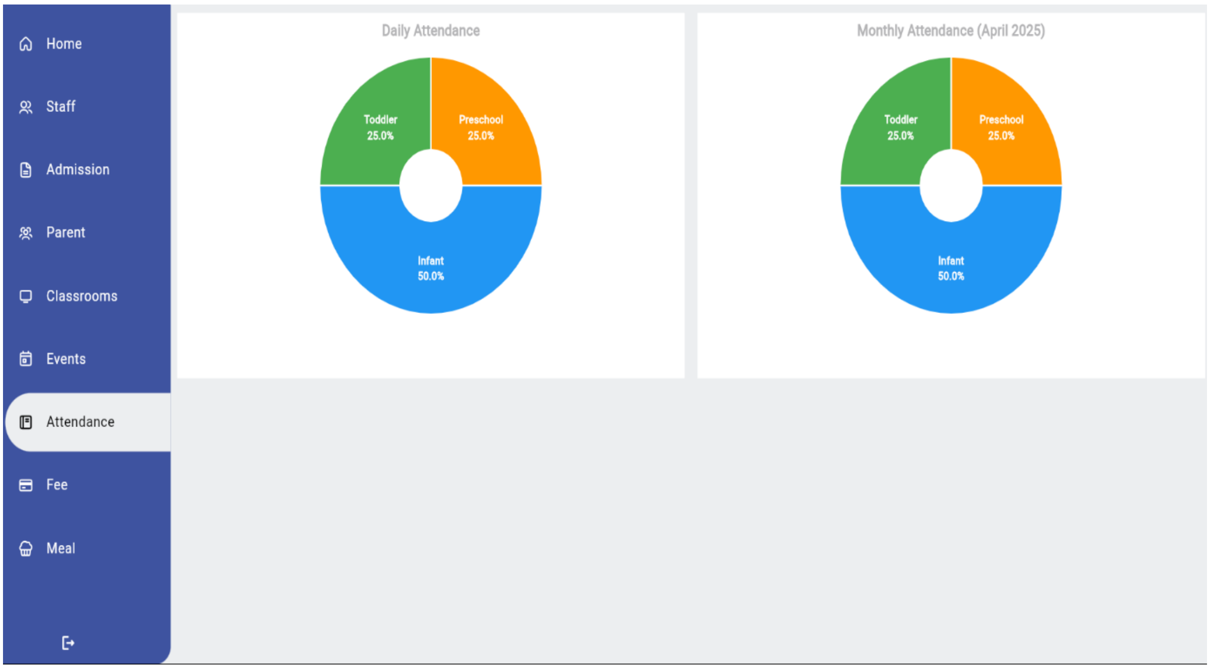
## EVENTS



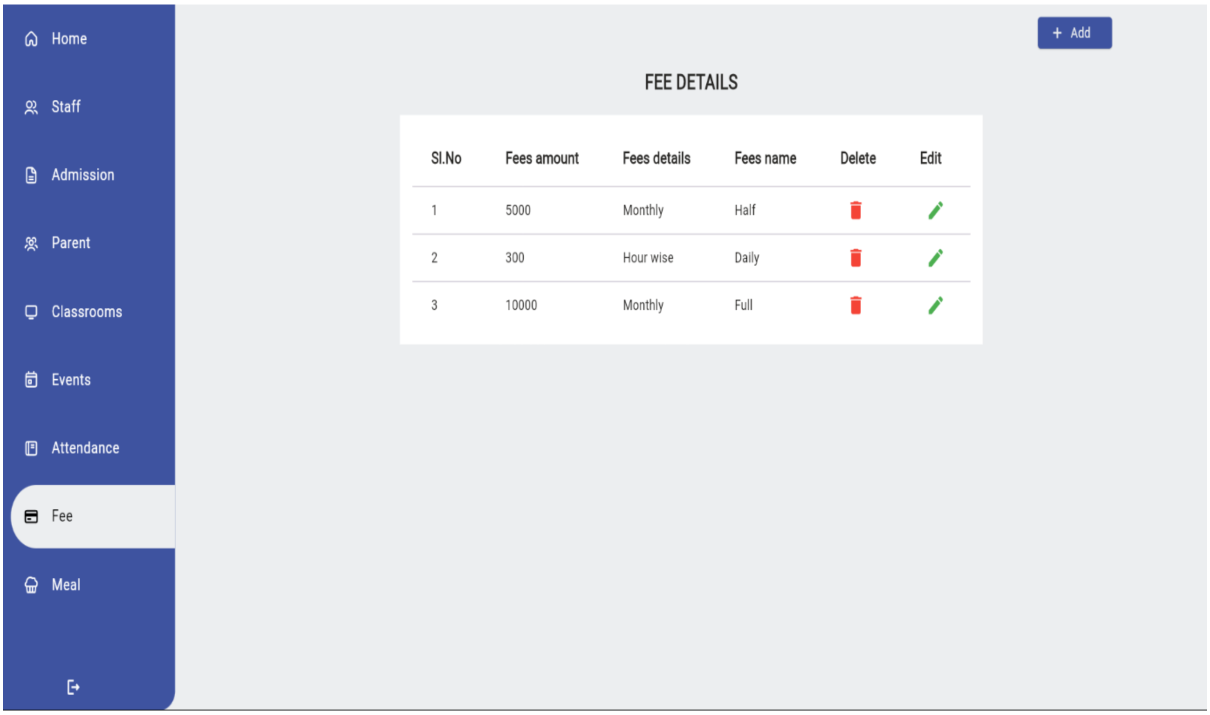
Art workshop 2025-03-11

Painting competition 2025-03-19

## ATTENDANCE



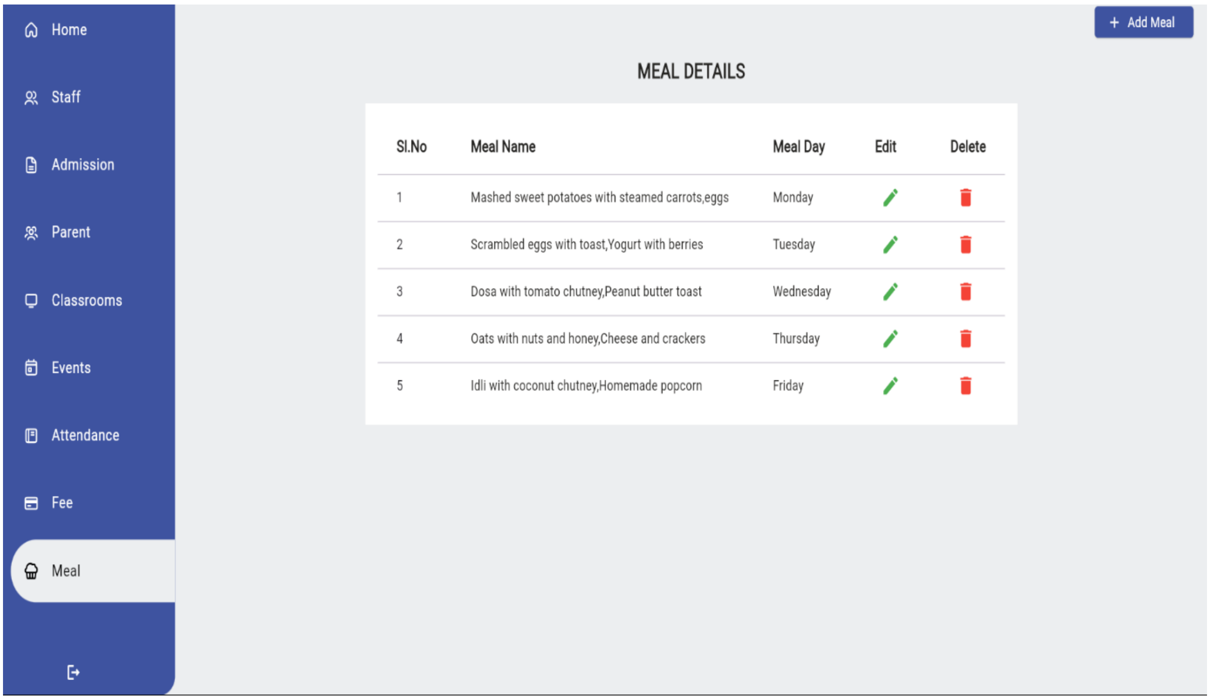
## FEE STRUCTURE



FEE DETAILS

| Sl.No | Fees amount | Fees details | Fees name | Delete | Edit |
|---|---|---|---|---|---|
| 1 | 5000 | Monthly | Half | 🗑 | ✎ |
| 2 | 300 | Hour wise | Daily | 🗑 | ✎ |
| 3 | 10000 | Monthly | Full | 🗑 | ✎ |

## MEAL DETAILS

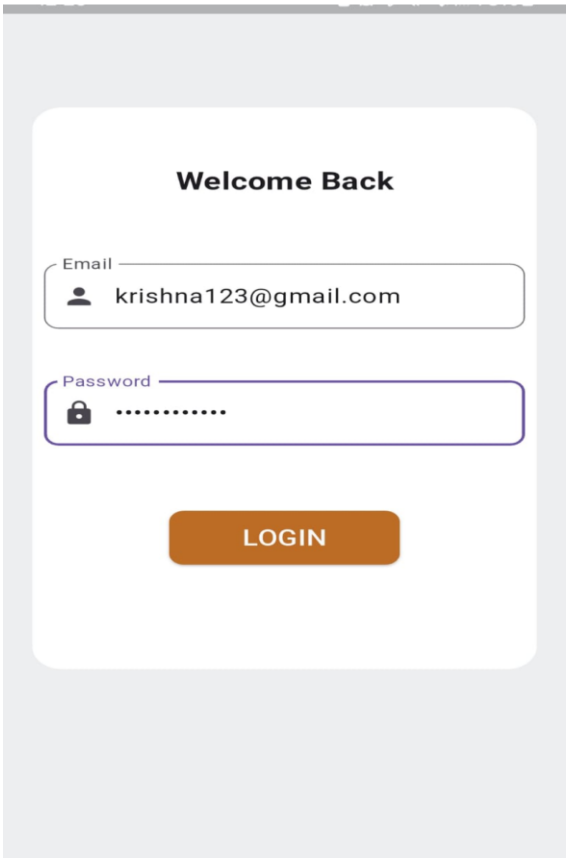


MEAL DETAILS

| Sl.No | Meal Name | Meal Day | Edit | Delete |
|---|---|---|---|---|
| 1 | Mashed sweet potatoes with steamed carrots,eggs | Monday | ✎ | 🗑 |
| 2 | Scrambled eggs with toast,Yogurt with berries | Tuesday | ✎ | 🗑 |
| 3 | Dosa with tomato chutney,Peanut butter toast | Wednesday | ✎ | 🗑 |
| 4 | Oats with nuts and honey,Cheese and crackers | Thursday | ✎ | 🗑 |
| 5 | Idli with coconut chutney,Homemade popcorn | Friday | ✎ | 🗑 |

## STAFF

### LOGIN

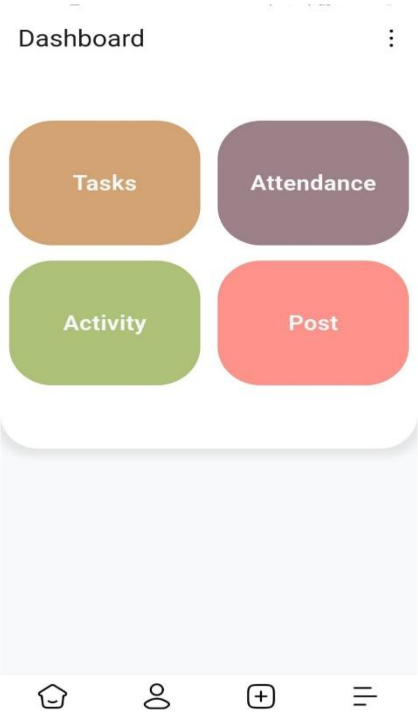


Welcome Back

Email
krishna123@gmail.com

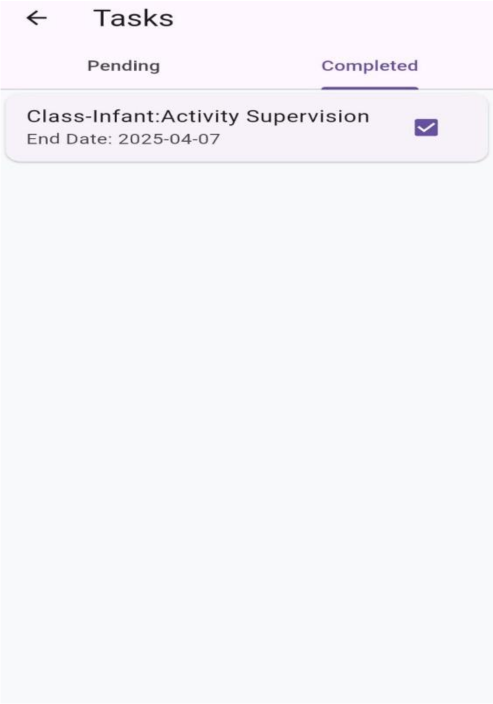
Password
•••••••••••••

### DASHBOARD



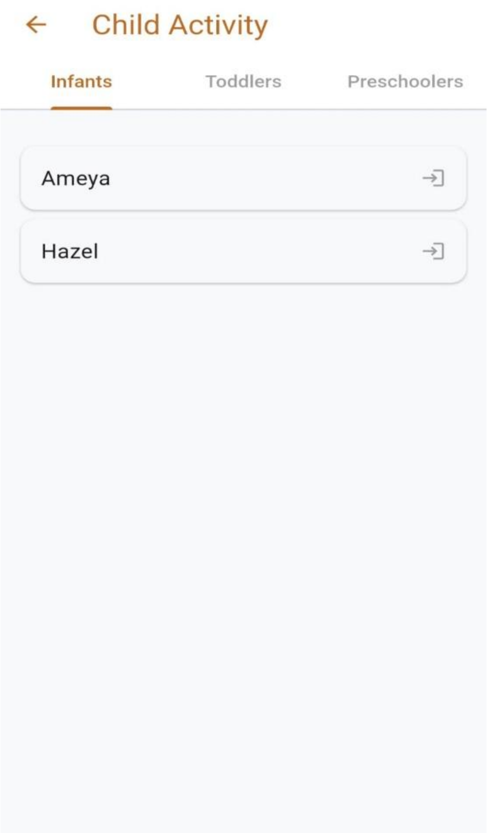
Dashboard

## TASKS



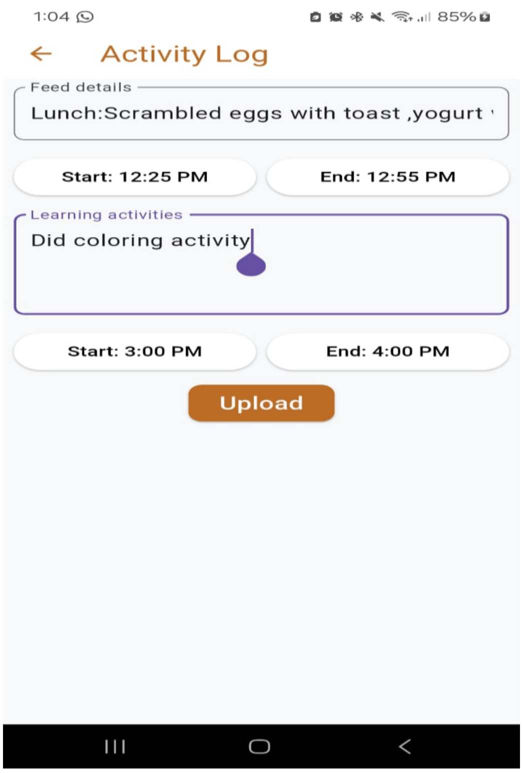
Class-Infant:Activity Supervision ☑
End Date: 2025-04-07

## ACTIVITY



Ameya

Hazel



1:04

85%

Activity Log

Feed details

Lunch:Scrambled eggs with toast ,yogurt

Start: 12:25 PM

End: 12:55 PM

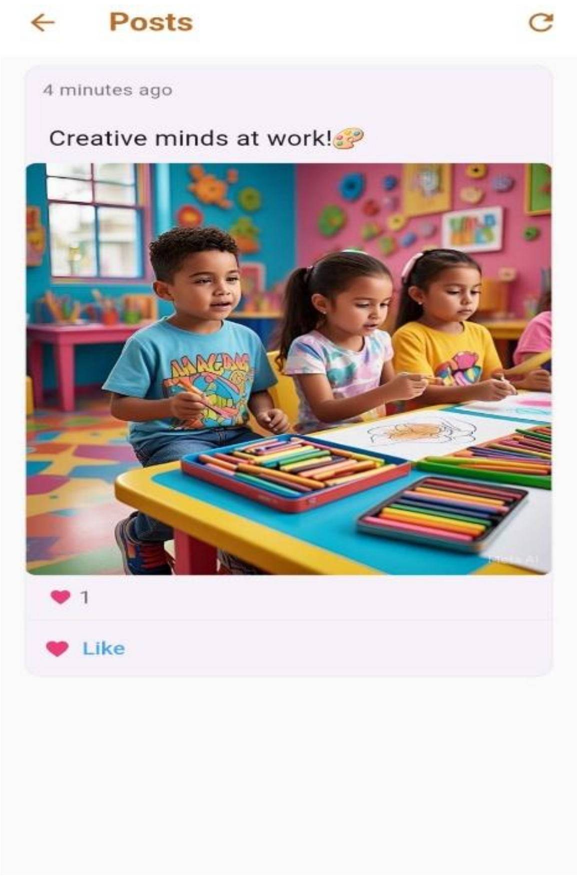
Learning activities

Did coloring activity

Start: 3:00 PM

End: 4:00 PM

## POSTS



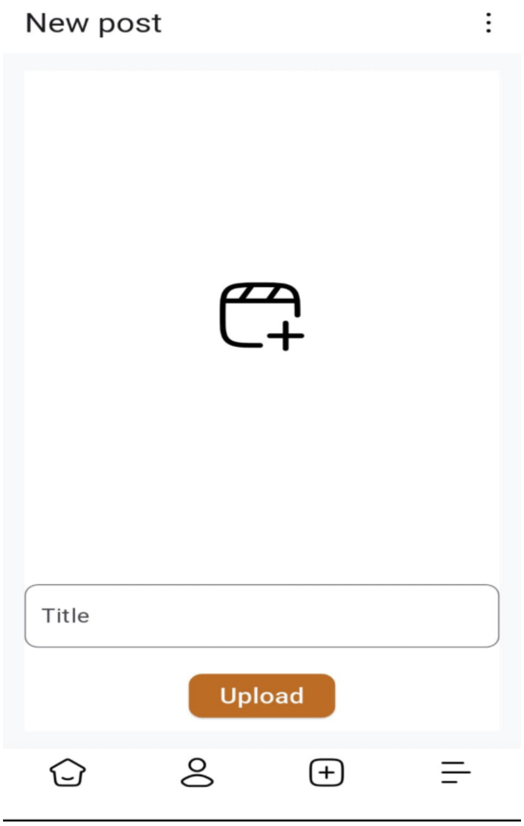
Posts

4 minutes ago

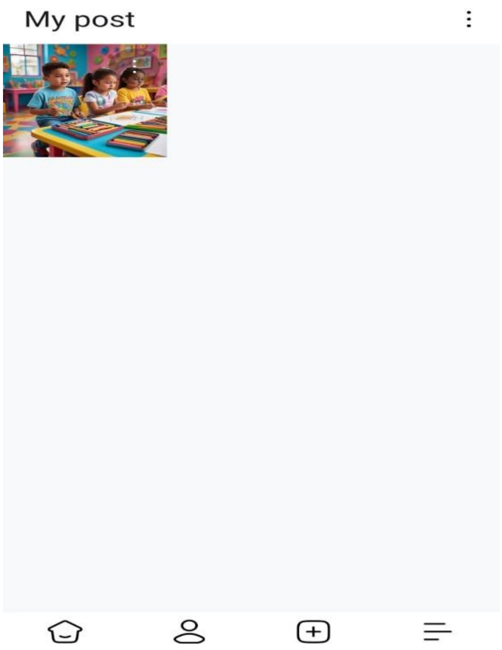
Creative minds at work!🎨

❤ 1

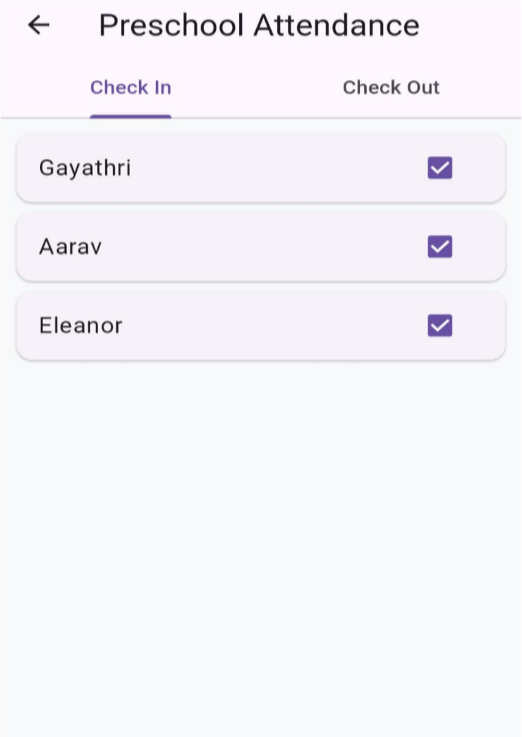
## ADD POST



## MY POSTS



## ATTENDANCE



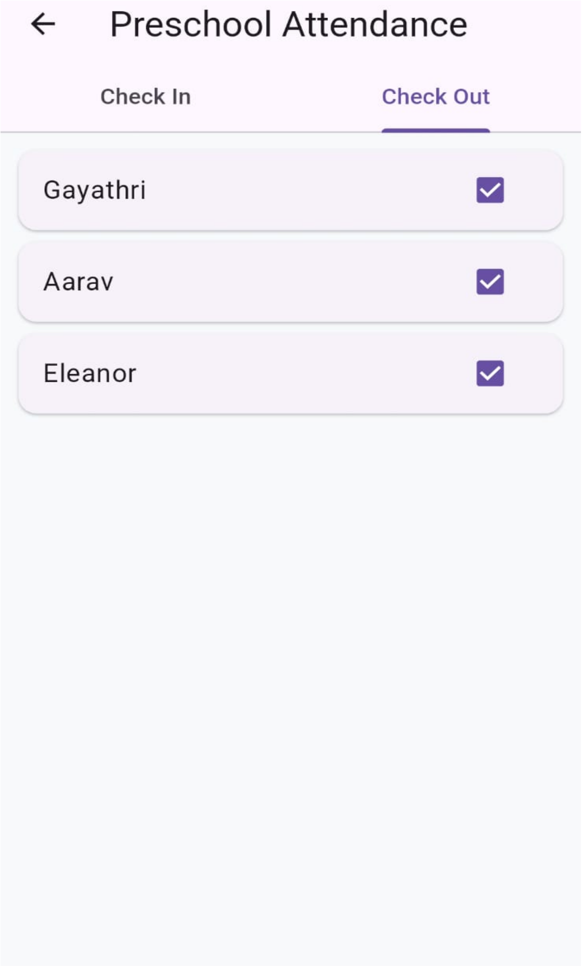
Preschool Attendance

Check In | Check Out

Gayathri ☑

Aarav ☑

Eleanor ☑



Preschool Attendance

Check In | Check Out

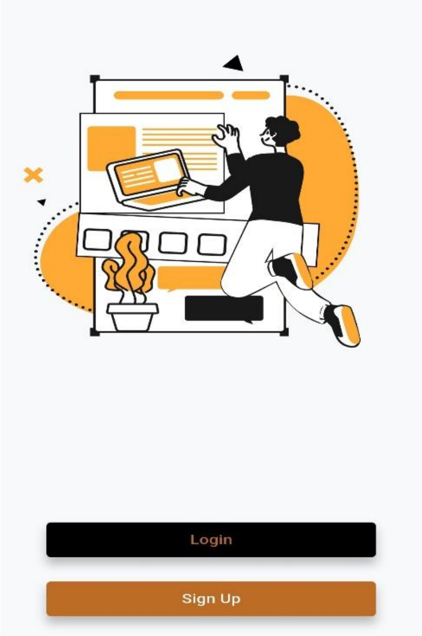
Gayathri ☑

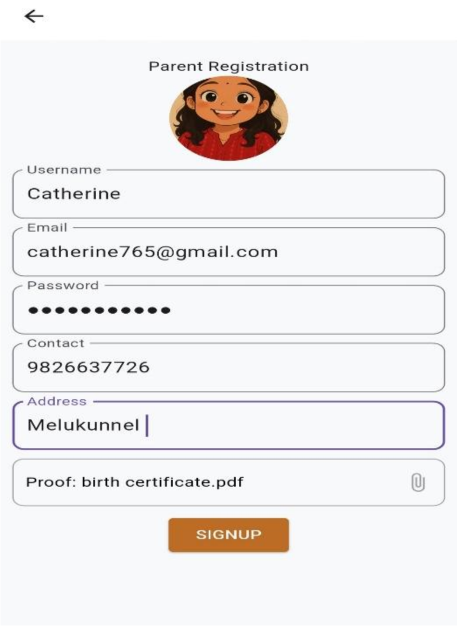
Aarav ☑

Eleanor ☑

# PARENT

## SIGNUP AND LOGIN




Parent Registration

Username: Catherine

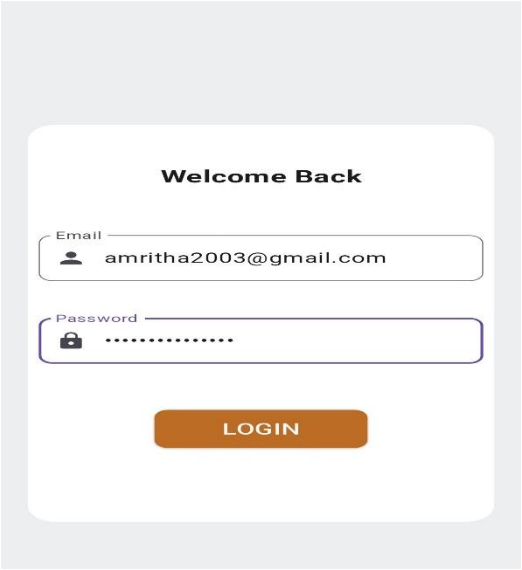
Email: catherine765@gmail.com

Password: ●●●●●●●●●●●

Contact: 9826637726

Address: Melukunnel

Proof: birth certificate.pdf

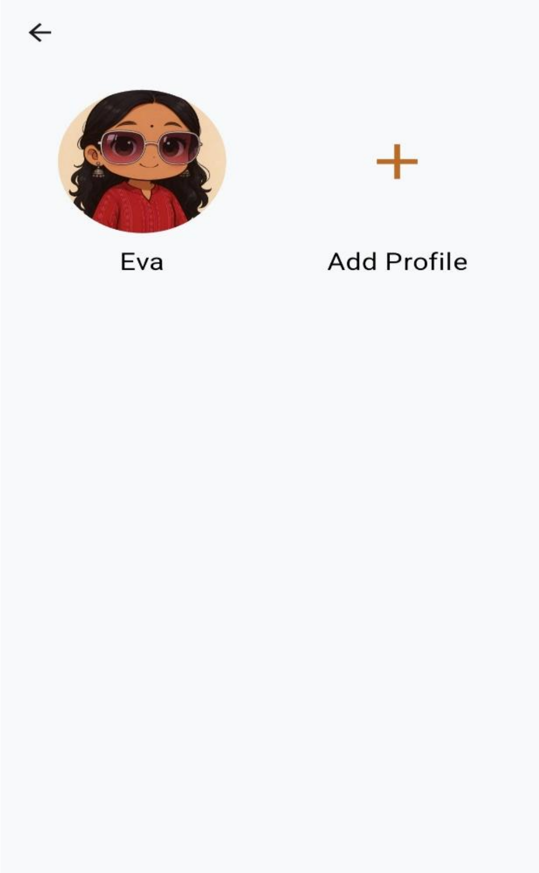


Welcome Back

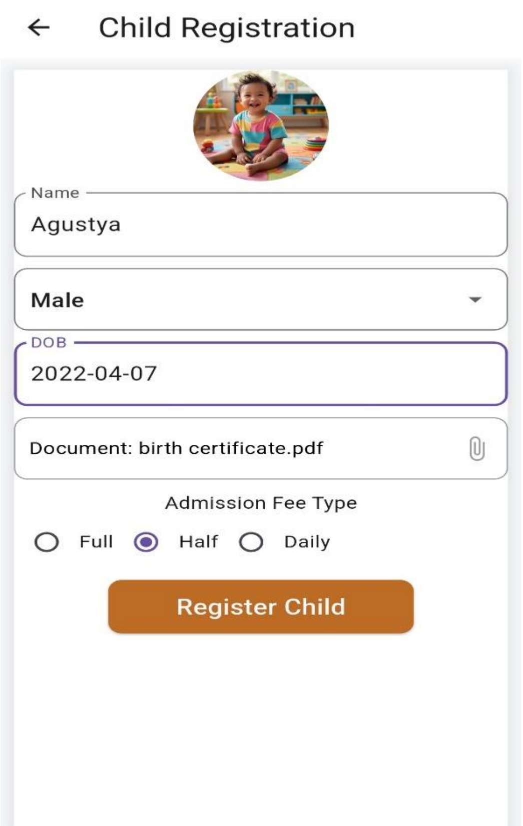
Email: amritha2003@gmail.com

Password: ••••••••••••••••

## PROFILES OF CHILDREN



## REGISTERING CHILD



← Child Registration

Name

Agustya

Male

DOB

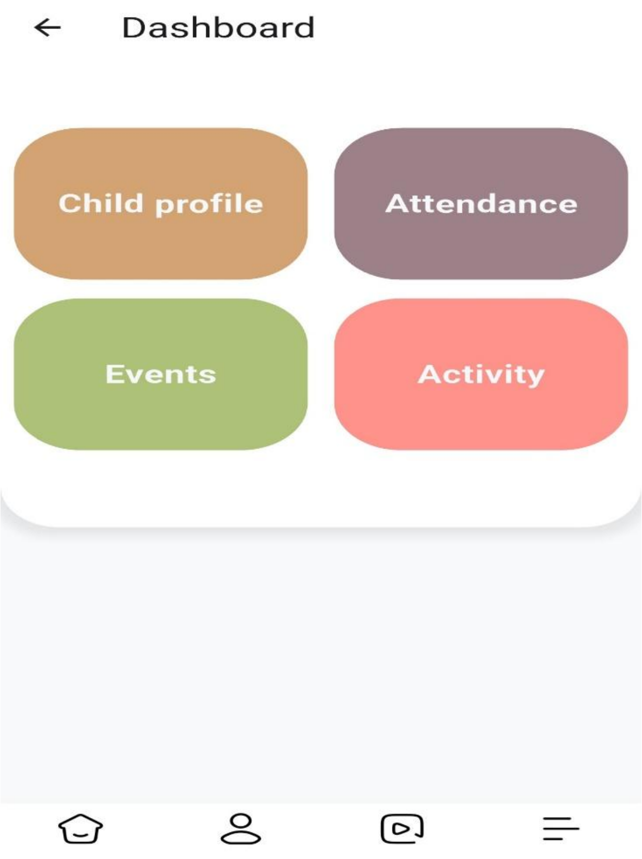
2022-04-07

Document: birth certificate.pdf

Admission Fee Type

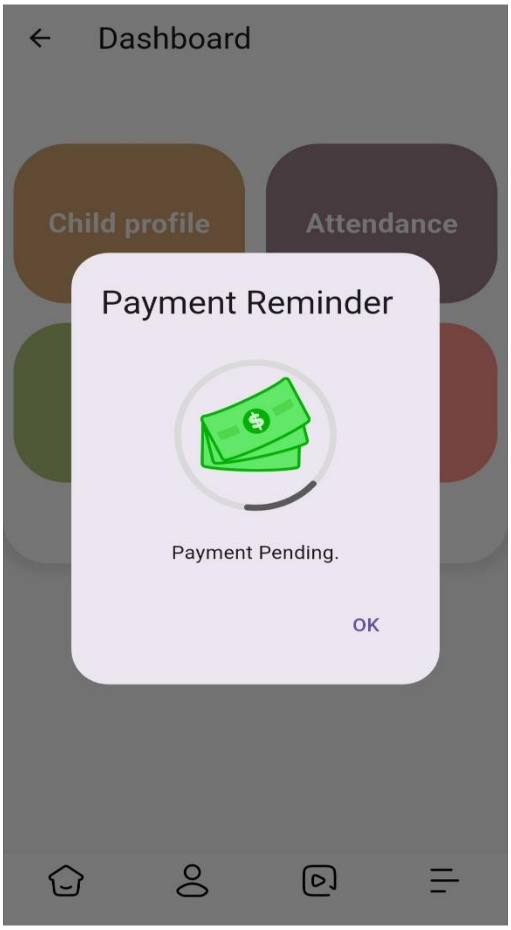
○ Full ● Half ○ Daily

## DASHBOARD



← Dashboard

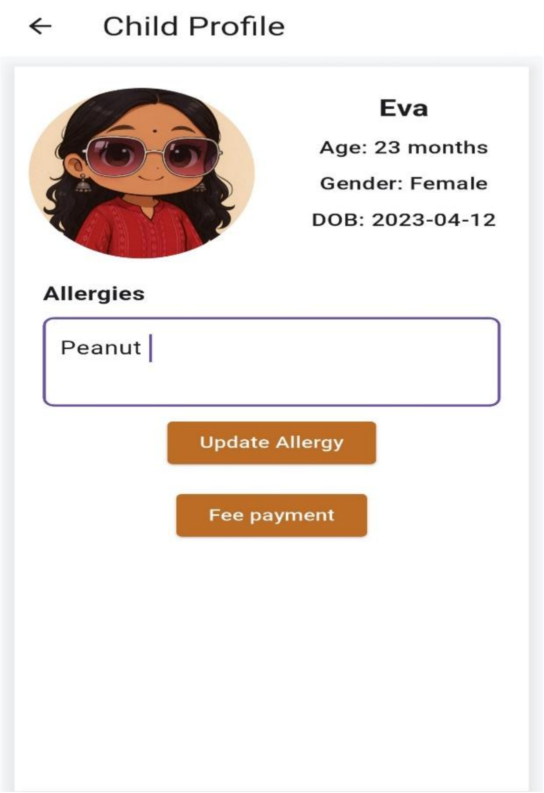
## PAYMENT REMINDER



Payment Reminder

Payment Pending.

## CHILD PROFILE



Child Profile

Eva

Age: 23 months

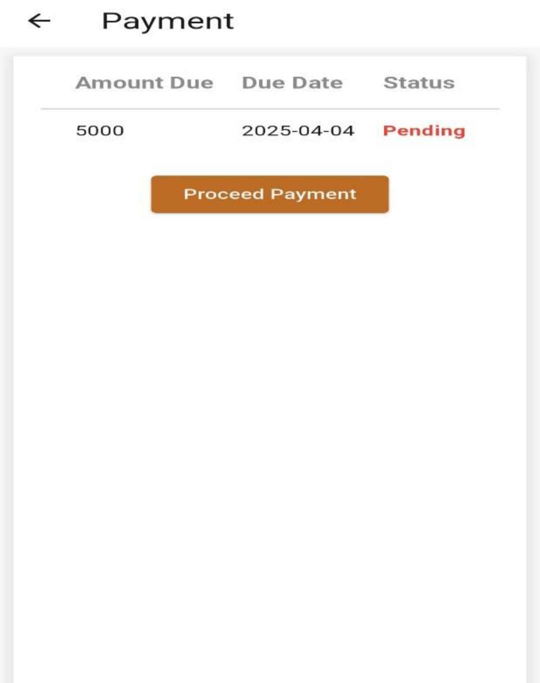
Gender: Female

DOB: 2023-04-12

Allergies

Peanut

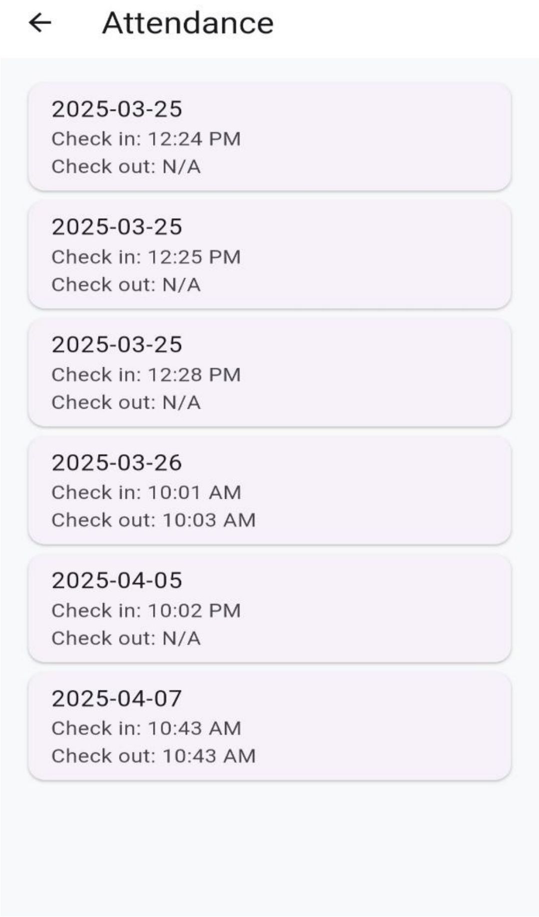
## PAYMENT



← Payment

| Amount Due | Due Date | Status |
|---|---|---|
| 5000 | 2025-04-04 | Pending |

## ATTENDANCE



← Attendance

2025-03-25
Check in: 12:24 PM
Check out: N/A

2025-03-25
Check in: 12:25 PM
Check out: N/A

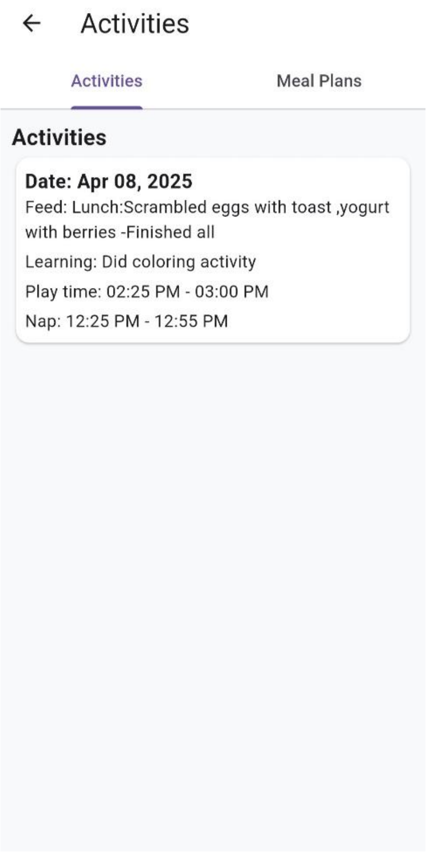
2025-03-25
Check in: 12:28 PM
Check out: N/A

2025-03-26
Check in: 10:01 AM
Check out: 10:03 AM

2025-04-05
Check in: 10:02 PM
Check out: N/A

2025-04-07
Check in: 10:43 AM
Check out: 10:43 AM

## ACTIVITIES



Activities

Date: Apr 08, 2025

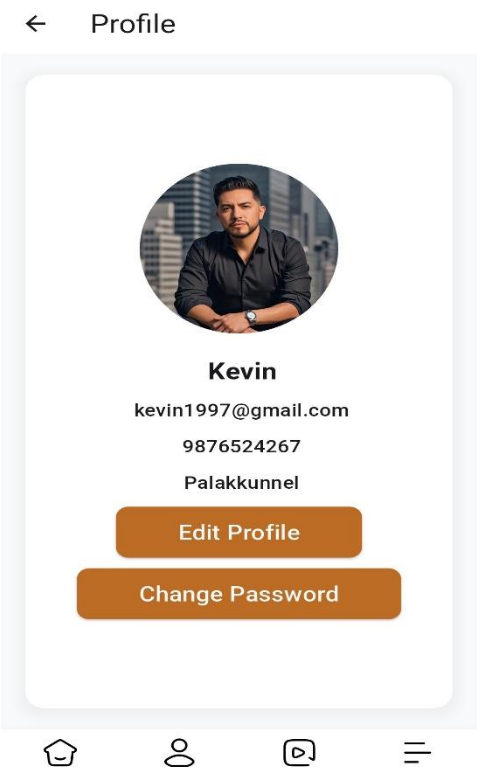
Feed: Lunch:Scrambled eggs with toast ,yogurt with berries -Finished all

Learning: Did coloring activity

Play time: 02:25 PM - 03:00 PM

Nap: 12:25 PM - 12:55 PM

## PARENT'S PROFILE



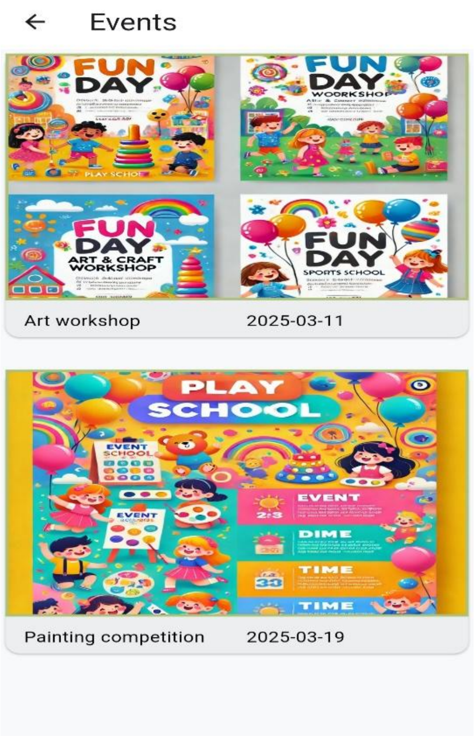
Kevin

kevin1997@gmail.com

9876524267

Palakkunnel

## EVENTS



← Events

Art workshop 2025-03-11

Painting competition 2025-03-19

## CODE

**Main.dart**

```
import 'package:flutter/material.dart';
import 'package:supabase_flutter/supabase_flutter.dart';
import 'package:tinytots_staff/screen/login.dart';


Future<void> main() async {
  await Supabase.initialize(
    url: 'https://dusxeazevqhptwodxcvk.supabase.co',
    anonKey:

'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6ImR1c3hlYXpldnFocHR3b2R4Y3ZrIiwicm9sZSI6ImFub24iLCJpYXQiOjE3MzQzNDY2NjcsImV4cCI6MjA0OTkyMjY2N30.0ely6domWJ9JwKwPeLL2pp8LqQeA--6n-gEG9e8Uh40',
  );
  runApp(const MainApp());
}
final supabase = Supabase.instance.client;
class MainApp extends StatelessWidget {
  const MainApp({super.key});
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Login(),
    );
  }
}
```

**Login.dart**

```
import 'package:flutter/material.dart';
import 'package:tinytots_staff/main.dart';
import 'package:cherry_toast/resources/arrays.dart';
import 'package:cherry_toast/cherry_toast.dart';
import 'package:tinytots_staff/screen/dashboard.dart';

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

  @override
  State<Login> createState() => _LoginState();
}

class _LoginState extends State<Login> {
  final TextEditingController emailController = TextEditingController();
  final TextEditingController passwordController = TextEditingController();

  Future<void> signin() async {
    try {
      final response = await supabase.auth.signInWithPassword(
        email: emailController.text.trim(),
        password: passwordController.text.trim(),
      );
      final user = response.user;
      if (user != null) {
        final parentData = await supabase
            .from('tbl_staff')
            .select()
            .eq('staff_email', emailController.text.trim())
            .single();
        if (parentData != null) {
          Navigator.pushReplacement(
            context,
            MaterialPageRoute(builder: (context) => Dashboard()),
          );
        } else {
          CherryToast.error(
            description: Text("Access denied. Only staffs can log in."),
            animationType: AnimationType.fromRight,
            autoDismiss: true,
          ).show(context);
        }
      }
    } catch (e) {
      print('ERROR: $e');
      CherryToast.error(
              description: Text("No user found for that email.",
                  style: TextStyle(color: Colors.black)),
              animationType: AnimationType.fromRight,
              animationDuration: Duration(milliseconds: 1000),
              autoDismiss: true)
          .show(context);
      print('No user found for that email.');
    }
  }

  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xFFeceef0),
      body: SingleChildScrollView(
        child: Form(
          child: Padding(
            padding: EdgeInsets.only(top: 110, left: 20, right: 20),
            child: Container(
              decoration: BoxDecoration(
                borderRadius: BorderRadius.all(Radius.circular(20)),
                color: Color(0xFFffffff),
              ),
              height: 500,
              child: SingleChildScrollView(
                child: Column(
                  children: [
                    SizedBox(height: 50),
                    Text("Welcome Back",
                        style: TextStyle(
                            fontSize: 20, fontWeight: FontWeight.bold)),
                    SizedBox(height: 50),
                    Padding(
                      padding: const EdgeInsets.all(8.0),
                      child: TextFormField(
                        controller: emailController,
                        decoration: InputDecoration(
                          labelText: 'Email',
                          border: OutlineInputBorder(
                              borderRadius: BorderRadius.circular(10)),
                          prefixIcon: Icon(Icons.person),
                        ),
                      ),
                    ),
                    SizedBox(height: 30),
                    Padding(
                      padding: const EdgeInsets.all(8.0),
                      child: TextFormField(
                        controller: passwordController,
                        obscureText: true,
                        decoration: InputDecoration(
                          labelText: 'Password',
                          border: OutlineInputBorder(
                              borderRadius: BorderRadius.circular(10)),
                          prefixIcon: Icon(Icons.lock),
                        ),
                      ),
                    ),
                    SizedBox(height: 50),
                    ElevatedButton(
                        onPressed: () {
                          signin();
                        },
                        style: ElevatedButton.styleFrom(
                          backgroundColor: Color(0xFFbc6c25),
                          padding: EdgeInsets.symmetric(
                              horizontal: 50, vertical: 10),
                          shape: RoundedRectangleBorder(
                            borderRadius: BorderRadius.circular(10),
                          ),
                        ),
                        child: Text(
                          'LOGIN',
                          style:
                              TextStyle(fontSize: 18, color: Color(0xfff8f9fa)),
                      )),
                   ],
                  ),
                 ),
                ),
               ),

              ),
             ),
            );
          }
        }
```

**attendance.dart**

```
import 'package:flutter/material.dart';
import 'package:tinytots_staff/screen/infant.dart';
import 'package:tinytots_staff/screen/prescchooler.dart';
import 'package:tinytots_staff/screen/toddler.dart';

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

  @override
  State<AttendanceChild> createState() => _AttendanceChildState();
}

class _AttendanceChildState extends State<AttendanceChild> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
          elevation: 0,
          backgroundColor: Colors.white,
          foregroundColor: Color(0xFFbc6c25),
          title: Text('Attendance'),
        ),
        backgroundColor: Color(0xfff8f9fa),
        body: SingleChildScrollView(
          child: Padding(
            padding: const EdgeInsets.symmetric(horizontal: 50),
            child: Column(
              children: [
                SizedBox(height: 20),
                GestureDetector(
                  onTap: () {
                    Navigator.push(
                        context,
                        MaterialPageRoute(
                          builder: (context) => Infant(),
                        ));
                  },
                  child: Container(
                      height: 200,
                      width: 500,
                      decoration: BoxDecoration(
                        borderRadius: BorderRadius.all(Radius.circular(35)),
                        color: Color(0xFFffffff),
                      ),
                      child: Text('Infants')),
                ),
                SizedBox(height: 20),
                GestureDetector(
                  onTap: () {
                    Navigator.push(
                        context,
                        MaterialPageRoute(
                          builder: (context) => ToddlerAttendance(),
                        ));
                  },
                  child: Container(
                    height: 200,
                    width: 500,
                    decoration: BoxDecoration(
                      borderRadius: BorderRadius.all(Radius.circular(35)),
                      color: Color(0xFFffffff),
                    ),
                    child: Text('Toddlers'),
                  ),
                ),
                SizedBox(height: 20),
                GestureDetector(
                  onTap: () {
                    Navigator.push(
                        context,
                        MaterialPageRoute(
                          builder: (context) => PreschoolAttendance(),
                        ));
                  },
                  child: Container(
                    height: 200,
                    width: 500,
                    decoration: BoxDecoration(
                      borderRadius: BorderRadius.all(Radius.circular(35)),
                      color: Color(0xFFffffff),
                    ),
                    child: Text('Preschoolers'),
                  ),
                ),
              ],
            ),
          ),
        ),
      );
    }
  }
```

**activitylog.dart**

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

class Log extends StatefulWidget {
  final int childId;
  const Log({super.key, required this.childId});

  @override
  State<Log> createState() => _LogState();
}

class _LogState extends State<Log> {
  final TextEditingController feedController = TextEditingController();
  TimeOfDay? napStartTime;
```

```
TimeOfDay? napEndTime;
TimeOfDay? playStartTime;
TimeOfDay? playEndTime;
final TextEditingController learningController = TextEditingController();
final TextEditingController playtimeController = TextEditingController();

Future<void> storeData() async {
  try {
    final staffId = supabase.auth.currentUser?.id;
    final napSchedule = napStartTime != null && napEndTime != null
        ? '${napStartTime!.format(context)} - ${napEndTime!.format(context)}'
        : '';
    final playSchedule = playStartTime != null && playEndTime != null
        ? '${playStartTime!.format(context)} - ${playEndTime!.format(context)}'
        : '';

    await supabase.from('tbl_activity').insert([
      {
        'child_id': widget.childId,
        'feed_details': feedController.text,
        'nap_schedule': napSchedule,
        'learning_activities': learningController.text,
        'play_time_activities': playSchedule,
        'staff_id': staffId,
      }
    ]);

    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text(
          "Activity added successfully",
          style: TextStyle(color: Colors.white),
        ),
        backgroundColor: Colors.black54,
      ),
    );
  } catch (e) {
    print(' upload failed: $e');
  }
}

Future<void> _selectNapTime(BuildContext context, bool isStart) async {
  final TimeOfDay? picked = await showTimePicker(
    context: context,
    initialTime: TimeOfDay.now(),
  );
  if (picked != null) {
    setState(() {
      if (isStart) {
        napStartTime = picked;
      } else {
        napEndTime = picked;
      }
    });
  }
}

Future<void> _selectPlayTime(BuildContext context, bool isStart) async {
  final TimeOfDay? picked = await showTimePicker(
    context: context,
    initialTime: TimeOfDay.now(),
  );
  if (picked != null) {
    setState(() {
      if (isStart) {
        playStartTime = picked;
      } else {
        playEndTime = picked;
      }
    });
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      elevation: 0,
      backgroundColor: Colors.white,
      foregroundColor: const Color(0xFFbc6c25),
      title: const Text('Activity Log'),
    ),
    backgroundColor: const Color(0xfff8f9fa),
    body: SingleChildScrollView(
      child: Column(
        children: [
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: TextField(
              controller: feedController,
              decoration: InputDecoration(
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(8),
                ),
                labelText: 'Feed details',
              ),
            ),
          ),
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                Expanded(
                  child: ElevatedButton(
                    style: ElevatedButton.styleFrom(
                      backgroundColor: Colors.white,
                      foregroundColor: Colors.black,
                    ),
                    onPressed: () => _selectNapTime(context, true),
                    child: Text(
                      napStartTime == null
                          ? 'Select Nap Start'
                          : 'Start: ${napStartTime!.format(context)}',
                    ),
                  ),
                ),
                const SizedBox(width: 8),
                Expanded(
                  child: ElevatedButton(
                    style: ElevatedButton.styleFrom(
                      backgroundColor: Colors.white,
                      foregroundColor: Colors.black,
                    ),
                    onPressed: () => _selectNapTime(context, false),
                    child: Text(
                      napEndTime == null
                          ? 'Select Nap End'
                          : 'End: ${napEndTime!.format(context)}',
                    ),
                  ),
                ),
              ],
            ),
          ),
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: TextField(
              controller: learningController,
              maxLines: 3,
              decoration: InputDecoration(
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(8),
                ),
                labelText: 'Learning activities',
                alignLabelWithHint: true,
              ),
            ),
          ),
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                Expanded(
                  child: ElevatedButton(
                    style: ElevatedButton.styleFrom(
                      backgroundColor: Colors.white,
                      foregroundColor: Colors.black,
                    ),
                    onPressed: () => _selectPlayTime(context, true),
                    child: Text(
                      playStartTime == null
                          ? 'Select Play Start'
                          : 'Start: ${playStartTime!.format(context)}',
                    ),
                  ),
                ),
                const SizedBox(width: 8),
                Expanded(
                  child: ElevatedButton(
                    style: ElevatedButton.styleFrom(
                      backgroundColor: Colors.white,
                      foregroundColor: Colors.black,
                    ),
                    onPressed: () => _selectPlayTime(context, false),
                    child: Text(
                      playEndTime == null
                          ? 'Select Play End'
                          : 'End: ${playEndTime!.format(context)}',
                    ),
                  ),
                ),
              ],
            ),
          ),
          ElevatedButton(
            style: ElevatedButton.styleFrom(
              backgroundColor: Color(0xFFbc6c25),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(10),
              ),
            ),
            onPressed: storeData,
            child: const Text(
              'Upload',
              style: TextStyle(
                fontSize: 18,
                color: Color(0xfff8f9fa),
              ),
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

**post .dart**

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:hugeicons/hugeicons.dart';
import 'package:image_picker/image_picker.dart';
import 'package:tinytots_staff/main.dart';

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

  @override
  State<Post> createState() => _PostState();
}

class _PostState extends State<Post> {
  final TextEditingController titleController = TextEditingController();
  File? _image;
  final ImagePicker _picker = ImagePicker();

  Future<void> _pickImage() async {
    final pickedFile = await _picker.pickImage(source: ImageSource.gallery);
    if (pickedFile != null) {
      setState(() {
        _image = File(pickedFile.path);
      });
    }
  }

  Future<String?> uploadImage(String staffId) async {
    try {
      final fileName =
        '$staffId-${DateTime.now().millisecondsSinceEpoch}.jpg'; // Unique file name
      await supabase.storage.from('post').upload(fileName, _image!);

      // Get public URL of the uploaded image
      final imageUrl = supabase.storage.from('post').getPublicUrl(fileName);
      return imageUrl;
    } catch (e) {
      print('Image upload failed: $e');
      return null;
    }
  }

  Future<void> storeData() async {
    try {
      final staffId = supabase.auth.currentUser?.id;
      if (staffId == null) {
        print("User is not authenticated");
        return;
      }

      final title = titleController.text;
      if (title.isEmpty) {
        print("Title is empty");
        return;
      }

      if (_image == null) {
        print("No image selected");
        return;
```

```
}

// Upload image and get URL
final imageUrl = await uploadImage(staffId);
if (imageUrl == null) {
  print("Image upload failed");
  return;
}

// Insert data into the database
await supabase.from('tbl_post').insert({
  'staff_id': staffId,
  'post_file': imageUrl,
  'post_title': title,
});

print("Data inserted successfully");

ScaffoldMessenger.of(context).showSnackBar(
  const SnackBar(
    content: Text(
      "Post added successfully",
      style: TextStyle(color: Colors.white),
    ),
    backgroundColor: Colors.black54,
  ),
);

titleController.clear();
setState(() {
        _image = null;
      });
    } catch (e) {
      print("Error inserting post data: $e");
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text(
            "Failed to add post",
            style: TextStyle(color: Colors.white),
          ),
          backgroundColor: Colors.red,
        ),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(16.0),
      child: SingleChildScrollView(
        child: Container(
          decoration: BoxDecoration(
            color: Color(0xffffffff),
          ),
          child: Column(
            children: [
              GestureDetector(
                  onTap: _pickImage,
                  child: Container(
      height: 450,
      width: 500,
      decoration: BoxDecoration(
        image: _image != null
          ? DecorationImage(
            image: FileImage(_image!),
            fit: BoxFit.cover,
           )
          : null,
        color: Color(0xffffffff)),
      child: _image == null
        ? HugeIcon(
          icon: HugeIcons.strokeRoundedPlayListAdd,
          color: Colors.black,
          size: 70.0,
         )
        : null,
    )),
  const SizedBox(height: 20),
  TextFormField(
   controller: titleController,
   decoration: InputDecoration(
    labelText: 'Title',
    border: OutlineInputBorder(
     borderRadius: BorderRadius.circular(10),
    ),
   ),
  ),
  const SizedBox(height: 20),
  ElevatedButton(
            style: ElevatedButton.styleFrom(
              backgroundColor: Color(0xFFbc6c25),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(10),
              ),
            ),
            onPressed: storeData,
            child: const Text(
              'Upload',
              style: TextStyle(
                fontSize: 18,
                color: Color(0xfff8f9fa),
              ),
            ),
          ),
          SizedBox(
            height: 10,
          )
        ],
      ),
    ),
  ),
  );
 }
}
```

**<u>toddler.dart</u>**

```
import 'package:flutter/material.dart';
import 'package:tinytots_staff/main.dart';
import 'package:intl/intl.dart';
```

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

  @override
  State<ToddlerAttendance> createState() => _ToddlerAttendanceState();
}

class _ToddlerAttendanceState extends State<ToddlerAttendance>
    with SingleTickerProviderStateMixin {
  late TabController _tabController;
  List<Map<String, dynamic>> toddlers = [];
  Map<int, bool> checkInAttendance = {};
  Map<int, bool> checkOutAttendance = {};
  Map<int, String> attendanceIds = {};

  @override
  void initState() {
    super.initState();
    _tabController = TabController(length: 2, vsync: this);
    fetchToddlers();
  }

  int calculateAgeInMonths(String dob) {
    try {
      DateTime birthDate = DateFormat("yyyy-MM-dd").parse(dob);
      DateTime today = DateTime.now();
      int years = today.year - birthDate.year;
      int months = today.month - birthDate.month;
      if (today.day < birthDate.day) {
        months -= 1;
      }
      return (years * 12) + months;
    } catch (e) {
      print("Error parsing date: $e");
      return 0;
    }
  }

  Future<void> fetchToddlers() async {
    try {
      String todayDate = DateFormat('yyyy-MM-dd').format(DateTime.now());
      final childrenResponse =
          await supabase.from('tbl_child').select().eq('status', 1);
      ;
      List<Map<String, dynamic>> allChildren =
          List<Map<String, dynamic>>.from(childrenResponse);

      final attendanceResponse =
          await supabase.from('tbl_attendance').select().eq('date', todayDate);

      Map<int, String> existingAttendance = {};
      Map<int, bool> existingCheckIns = {};
      Map<int, bool> existingCheckOuts = {};

      for (var record in attendanceResponse) {
        final id = record['id'];
        if (id != null) {
          existingAttendance[record['child_id']] = id.toString();
          existingCheckIns[record['child_id']] = record['check_in'] != null;
          existingCheckOuts[record['child_id']] = record['check_out'] != null;
        }
      }

      setState(() {
        toddlers = allChildren
            .where((child) =>
                calculateAgeInMonths(child['dob']) > 12 &&
                calculateAgeInMonths(child['dob']) <= 36)
            .toList();

        checkInAttendance = {
          for (var toddler in toddlers)
            toddler['id']: existingCheckIns[toddler['id']] ?? false
        };

        checkOutAttendance = {
          for (var toddler in toddlers)
            toddler['id']: existingCheckOuts[toddler['id']] ?? false
        };

        attendanceIds = {
          for (var toddler in toddlers)
            toddler['id']: existingAttendance[toddler['id']] ?? ''
        };
      });
    } catch (e) {
      print("ERROR $e");
    }
  }
```

```
Future<void> updateAttendance(int childId, bool isCheckIn, bool value) async {
  try {
    String todayDate = DateFormat('yyyy-MM-dd').format(DateTime.now());
    final String currentId = attendanceIds[childId] ?? '';

    if (value) {
      if (currentId.isEmpty) {
        final response = await supabase.from('tbl_attendance').insert({
          'child_id': childId,
          'date': todayDate,
          'role': 'CHILD',
          isCheckIn ? 'check_in' : 'check_out':
              DateTime.now().toIso8601String(),
        }).select();

        final newId = response[0]['id'];
        if (newId != null) {
          setState(() {
            attendanceIds[childId] = newId.toString();
          });
        }
      } else {
        await supabase.from('tbl_attendance').update({
          isCheckIn ? 'check_in' : 'check_out':
              DateTime.now().toIso8601String(),
        }).eq('id', currentId);
      }
    } else if (currentId.isNotEmpty) {
      await supabase.from('tbl_attendance').update({
        isCheckIn ? 'check_in' : 'check_out': null,
   }).eq('id', currentId);
  }

  setState(() {
   if (isCheckIn) {
    checkInAttendance[childId] = value;
   } else {
    checkOutAttendance[childId] = value;
   }
  });
 } catch (e) {
  print("ERROR $e");
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text("Error updating attendance: $e")));
 }
}

Widget buildAttendanceTab(bool isCheckIn) {
 final attendanceMap = isCheckIn ? checkInAttendance : checkOutAttendance;

 return toddlers.isEmpty
   ? const Center(child: CircularProgressIndicator())
   : ListView.builder(
     itemCount: toddlers.length,
     itemBuilder: (context, index) {
      final toddler = toddlers[index];
      return Card(
       child: ListTile(
        title: Text(toddler['name']),
        trailing: Checkbox(
                    value: attendanceMap[toddler['id']] ?? false,
                    onChanged: (bool? value) {
                      updateAttendance(toddler['id'], isCheckIn, value!);
                    },
                  ),
                ),
              );
            },
          );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xfff8f9fa),
      appBar: AppBar(
        title: const Text("Toddler Attendance"),
        bottom: TabBar(
          controller: _tabController,
          tabs: const [
            Tab(text: 'Check In'),
            Tab(text: 'Check Out'),
          ],
        ),
      ),
      body: Padding(
        padding: const EdgeInsets.all(8.0),
        child: TabBarView(
          controller: _tabController,
          children: [
            buildAttendanceTab(true),
            buildAttendanceTab(false),
          ],
        ),
      ),
    );
  }

  @override
  void dispose() {
    _tabController.dispose();
    super.dispose();
  }
}
```

**mypost.dart**

```
import 'package:flutter/material.dart';
import 'package:hugeicons/hugeicons.dart';
import 'package:intl/intl.dart';
import 'package:tinytots_staff/main.dart';

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

  @override
  State<Mypost> createState() => _MypostState();
}

class _MypostState extends State<Mypost> {
  List<Map<String, dynamic>> _postList = [];
  bool _isLoading = true;
```

```
 Future<void> delete(int delId) async {
 try {
   await supabase.from('tbl_like').delete().eq('post_id', delId);

   await supabase.from('tbl_post').delete().eq('id', delId);

   await display();

   ScaffoldMessenger.of(context).showSnackBar(
     const SnackBar(
       content: Text(
         'Post deleted',
         style: TextStyle(color: Colors.white),
       ),
       backgroundColor: Colors.black54,
     ),
   );
 } catch (e) {
   print('ERROR DELETING POST $e');
 }
}

 Future<void> display() async {
   try {
     final response = await supabase
         .from('tbl_post')
         .select()
         .order('created_at', ascending: false);
```

```
    setState(() {
      _postList = response;
      _isLoading = false;
    });
  } catch (e) {
    print('ERROR DISPLAYING POST $e');
    setState(() => _isLoading = false);
  }
}

String formatTimeAgo(String timestamp) {
  DateTime postTime = DateTime.parse(timestamp);
  Duration difference = DateTime.now().difference(postTime);

  if (difference.inMinutes < 1) {
    return 'Just now';
  } else if (difference.inMinutes < 60) {
    return '${difference.inMinutes}m';
  } else if (difference.inHours < 24) {
    return '${difference.inHours}h';
  } else if (difference.inDays == 1) {
    return '1d';
  } else if (difference.inDays < 7) {
    return '${difference.inDays}d';
  } else {
    return DateFormat('MMM d').format(postTime);
  }
}

void _showPostDialog(BuildContext context, Map<String, dynamic> post) {
```

```
showDialog(
  context: context,
  builder: (BuildContext context) {
    return Dialog(
      backgroundColor: Colors.transparent,
      insetPadding: const EdgeInsets.all(10),
      child: Stack(
        children: [
          InteractiveViewer(
            boundaryMargin: const EdgeInsets.all(20),
            minScale: 0.1,
            maxScale: 4.0,
            child: Container(
              decoration: BoxDecoration(
                image: DecorationImage(
                  image: NetworkImage(post['post_file']),
                  fit: BoxFit.contain,
                ),
              ),
            ),
          ),
          Positioned(
            bottom: 10,
            left: 0,
            right: 0,
            child: Container(
              color: Colors.white,
              padding: const EdgeInsets.all(16),
              child: Column(
                mainAxisSize: MainAxisSize.min,
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    post['post_title'],
                    style: const TextStyle(
                      color: Colors.black,
                      fontSize: 18,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                  const SizedBox(height: 8),
                  Text(
                    formatTimeAgo(post['created_at']),
                    style: const TextStyle(
                      color: Colors.black,
                      fontSize: 14,
                    ),
                  ),
                ],
              ),
            ),
          ),
          Positioned(
            top: 10,
            right: 10,
            child: IconButton(
              icon: const Icon(Icons.close, color: Colors.white),
              onPressed: () => Navigator.of(context).pop(),
            ),
          ),
                    ],
                  ),
                );
              },
            );
          }

          @override
          void initState() {
            super.initState();
            display();
          }

          @override
          Widget build(BuildContext context) {
            return Expanded(
              child: _isLoading
                  ? const Center(child: CircularProgressIndicator())
                  : _postList.isEmpty
                      ? const Center(child: Text('No posts available'))
                      : GridView.builder(
                          gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
                            crossAxisCount: 3,
                            crossAxisSpacing: 2.0,
                            mainAxisSpacing: 2.0,
                            childAspectRatio: 1.0,
                          ),
                          itemCount: _postList.length,
                          itemBuilder: (context, index) {
                            final post = _postList[index];
```

```
return Stack(
  children: [
    GestureDetector(
      onTap: () => _showPostDialog(context, post),
      child: Container(
        decoration: BoxDecoration(
          image: DecorationImage(
            image: NetworkImage(post['post_file']),
            fit: BoxFit.cover,
          ),
        ),
      ),
    ),
    Positioned(
      top: 5,
      right: 5,
      child: Container(
        width: 40,
        height: 40,
        decoration: BoxDecoration(
          shape: BoxShape.circle,
          color: Colors.black.withOpacity(0.3),
        ),
        child: PopupMenuButton<String>(
          icon: HugeIcon(
            icon: HugeIcons.strokeRoundedMoreVertical,
            color: Colors.white,
            size: 24.0,
          ),
          onSelected: (String result) {
                if (result == 'delete') {
                 delete(post['id']);
                }
               },
               itemBuilder: (BuildContext context) => <PopupMenuEntry<String>>[
                const PopupMenuItem<String>(
                 value: 'delete',
                 child: Text('Delete'),
                ),
               ],
              ),
             ),
            ),
           ],
          );
         },
        ),
  );
 }
}
```