

1. INTRODUCTION

NSync is an innovative college event and activity management platform developed using Flutter. The system aims to streamline communication, coordination, and student engagement by providing a centralized platform for managing college events, clubs, and projects. It is designed to bridge the gap between students and faculty members, fostering a well-connected campus environment where event participation and project collaboration become effortless.

NSync offers an intuitive and user-friendly interface that enables students to explore upcoming events, join clubs, receive real-time notifications, and access college news. The platform ensures that students stay updated with the latest campus activities and opportunities. Additionally, it allows faculty members to manage event logistics, approve club memberships, and publish important announcements efficiently.

The system enhances accessibility and usability while maintaining a well-structured information flow within the college environment. By incorporating modern technology and a seamless user experience, NSync promotes a culture of engagement and productivity, making it an essential tool for academic institutions. The platform also supports scalability, ensuring that it can accommodate expanding user bases and additional functionalities in the future.

2. SYSTEM ANALYSIS

System analysis involves gathering and interpreting facts, diagnosing problems, and designing an improved system. The analysis phase includes understanding user needs, identifying system requirements, and developing structured tools to enhance efficiency. This process also involves evaluating the feasibility of implementation, analyzing risks, and determining optimal solutions to improve system performance. System analysis ensures that the final product effectively meets user expectations and institutional needs by conducting comprehensive research, collecting user feedback, and identifying bottlenecks.

2.1 EXISTING SYSTEM

Colleges rely on traditional methods such as notice boards, social media groups, and word-of-mouth for event coordination and communication. These methods often result in:

- Lack of centralized communication.
- Difficulty in tracking ongoing events and activities.
- Inefficient club membership management.
- Limited accessibility and real-time updates.

2.2 PROPOSED SYSTEM

NSync overcomes the shortcomings of the existing system by providing a **centralized, automated** event and activity management platform. Key features include:

- **Personalized Event Notifications** – Students receive real-time updates on upcoming events.
- **Club Membership Management** – Students can explore and join college clubs with ease.
- **College News Section** – Faculty can publish announcements and updates.
- **Project Collaboration Tools** – Students can manage academic projects collaboratively.
- **Secure User Authentication** – Role-based access control for students, faculty, and administrators.

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 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.

BUSINESS REQUIREMENTS

- The system should be **cost-effective** for both developers and the institution.
- The project must be completed **within the allocated timeframe**.
- Developers are responsible for **developing, deploying, and maintaining** the system.
- Regular **software updates and bug fixes** must be provided.
- Any required **user training** should be conducted for faculty and administrators.

USER REQUIREMENTS

The **User Requirements Specification (URS)** outlines what users expect from NSync.

For Students:

- Ability to **register for events** and receive updates.
- **Explore and join clubs** through the platform.
- Receive **real-time notifications** for college announcements.
- **Collaborate on projects** with peers and faculty.

For Faculty:

- **Create and manage events**, including approvals.
- **Oversee club memberships and activities**.
- **Post college news and announcements**.
- **Monitor student participation and engagement**.

For Administrators:

- **Manage users** (students, faculty, and other admins).
- **Oversee event planning and execution.**
- **Ensure system security and data integrity.**
- **Generate reports** on college activities.

2.3.1 Hardware Requirements

- **Processor:** Intel Core i3 or higher
- **RAM:** 8 GB or more
- **Storage:** 512 GB HDD/SSD
- **Display:** 15.6" LCD Monitor
- **Keyboard:** 108 keys

2.3.2 Software Requirements

- **Operating System:** Windows, macOS, Linux
- **Frontend:** Flutter (Dart)
- **Backend:** Firebase / Node.js
- **Database:** Supabase
- **Development Tools:** Android Studio, Visual Studio Code
- **Designing Tools:** Figma
- **Browser:** Chrome

2.3.3 Front End

The front end of NSync 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**.

Technology Stack

The NSync system utilizes **Supabase** as the back-end database solution. Supabase is an open-source alternative to Firebase, built on **PostgreSQL**, providing **authentication, real-time data updates, and API-based interactions** without requiring a traditional server-side programming language.

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**, NSync ensures that event management, user authentication, and database interactions remain **fast, secure, and responsive** for a seamless user experience.

2.4 FEASIBILITY ANALYSIS

A feasibility analysis evaluates the viability of implementing NSync by examining various aspects, including technical, operational, economic, and behavioral feasibility. This analysis ensures that the system is practical and beneficial for its intended users.

The four aspects of the feasibility study are:

- Technical feasibility
- Operational feasibility
- Economical feasibility
- Behavioral feasibility

Technical Feasibility

Technical feasibility assesses whether the existing infrastructure and technology can support the development and deployment of NSync. 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 NSync 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.

Operational Feasibility

Operational feasibility examines how well NSync will integrate into daily college activities and how efficiently users can adapt to it. The system is designed to be intuitive, minimizing the learning curve for students and faculty. Training sessions and user documentation will be provided to ensure smooth adoption.

Key operational benefits:

- **User-friendly interface** tailored for students, faculty, and administrators.
- **Automated notifications and event management**, reducing manual workload.
- **Seamless integration** with existing college communication channels.
- **Scalable structure** that can be expanded to accommodate more institutions.

Economic Feasibility

Economic feasibility evaluates the cost-effectiveness of implementing NSync. The project relies on open-source tools like Flutter and Firebase, minimizing software costs. Development and maintenance expenses are kept low due to the system's efficient architecture.

Cost-benefit analysis:

- **Low development costs** due to the use of free and open-source technologies.
- **Reduced administrative expenses** by automating event and club management.
- **Minimal hardware investment**, as most users already own compatible devices.
- **Long-term savings** by reducing reliance on paper-based communication

Behavioral Feasibility

Behavioral feasibility assesses how well users will accept and engage with the system. Since students and faculty members are already accustomed to mobile applications and online platforms, the transition to NSync will be smooth. Encouraging user participation through training sessions and feedback mechanisms will further improve adoption rates.

Key behavioral factors:

- **Positive user experience** with an intuitive and visually appealing design.
- **Easy adoption** due to familiarity with similar apps and online tools.
- **Continuous improvement** based on user feedback and analytics.
- **Enhanced student engagement** through interactive features like club memberships and project collaboration.

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."

Data 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.

Developing 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.

DFD Symbols

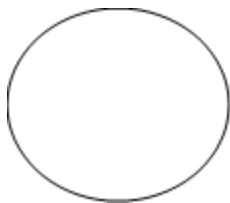
Square- Defines the source or destination of the system.



Data flow - Identifies data flow Circle



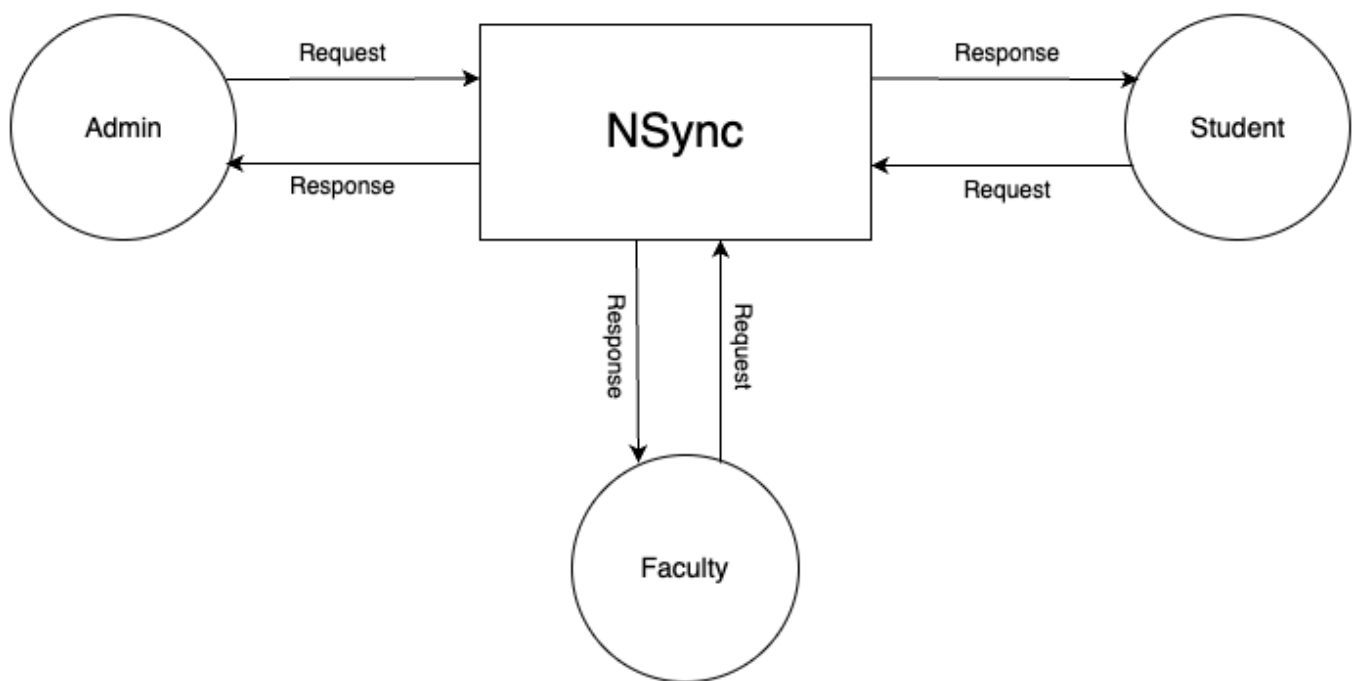
Bubble - Represents a process that transforms incoming data into outgoing data.



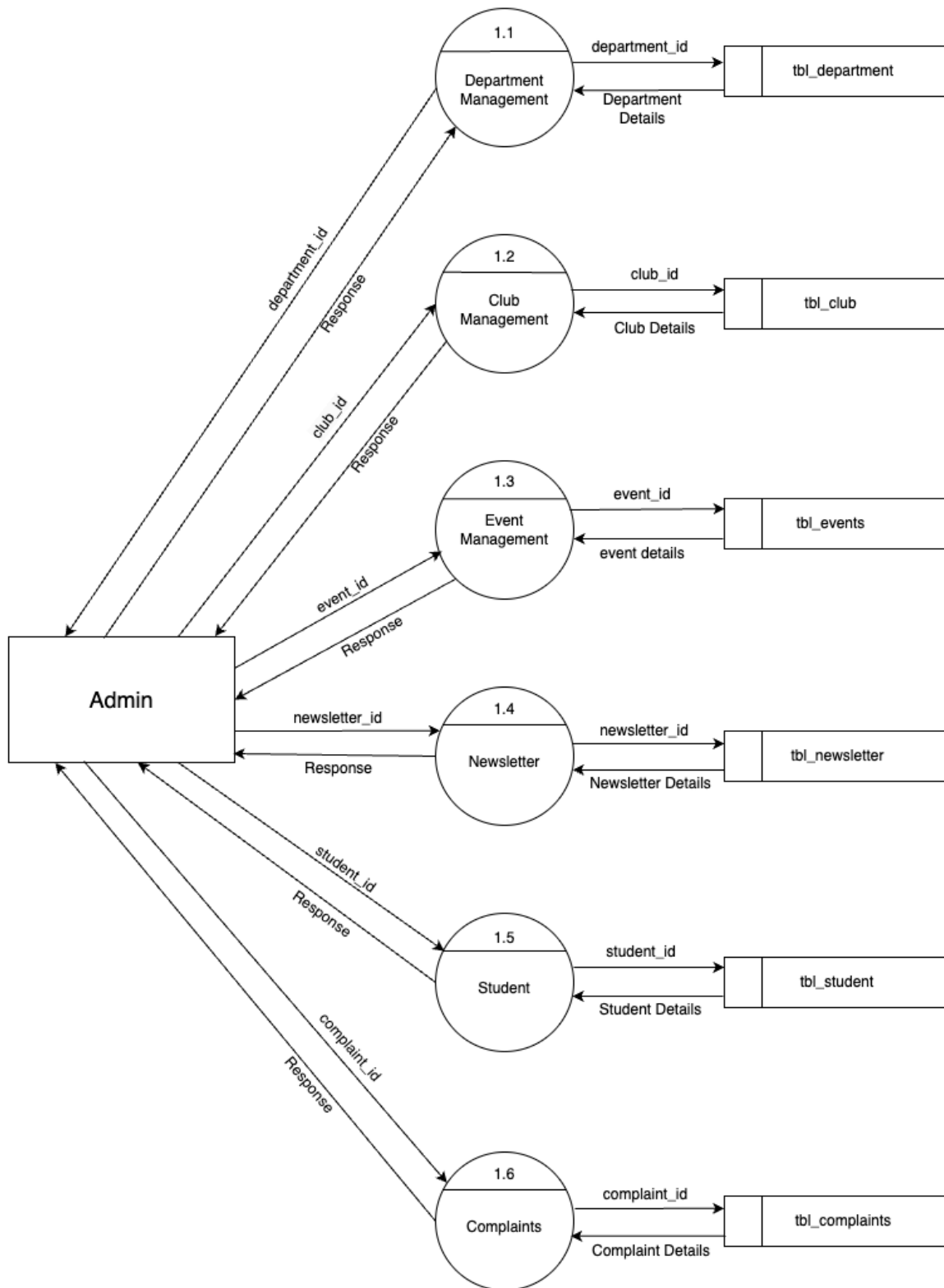
Open rectangle- Datastore



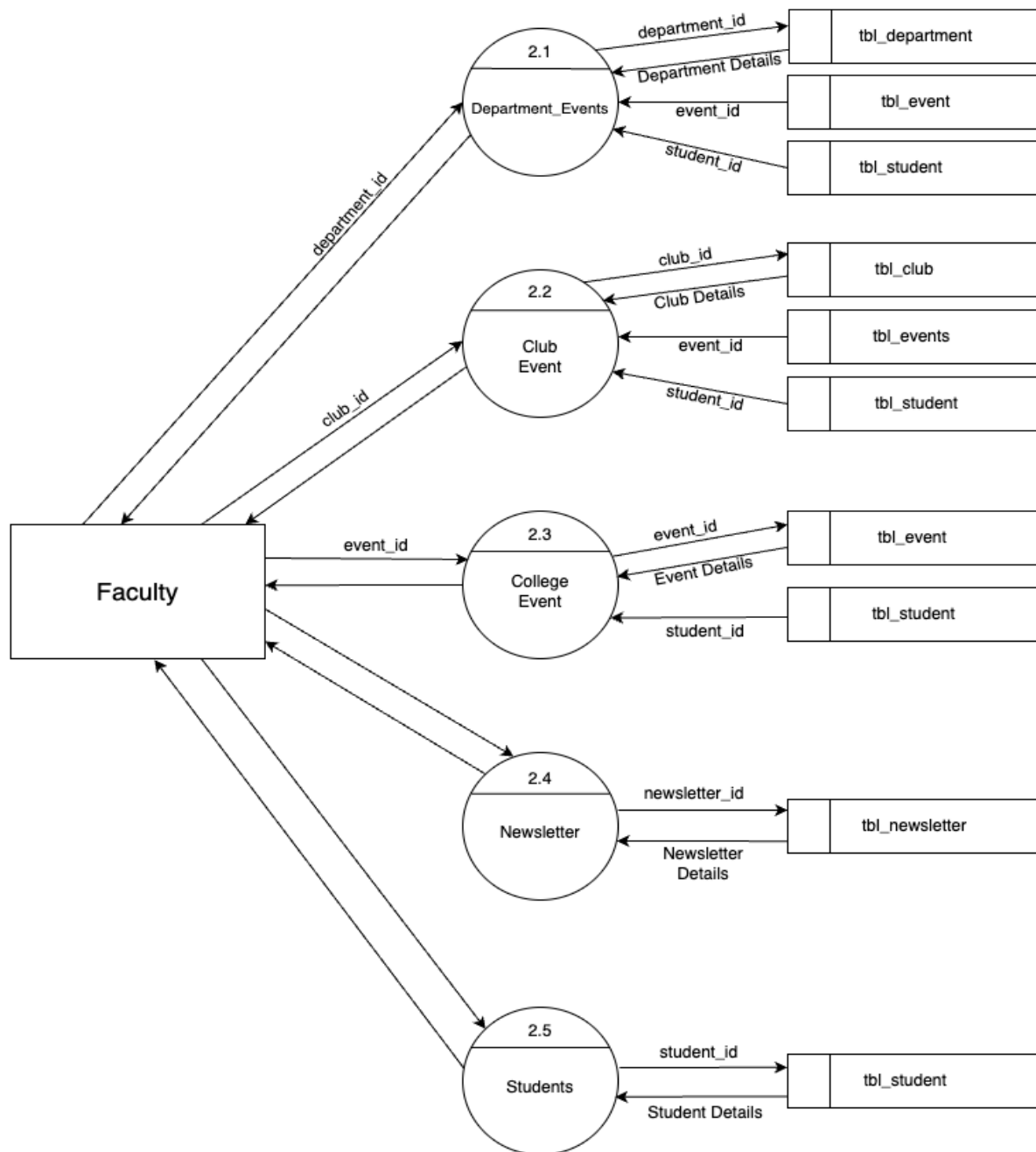
Level 0



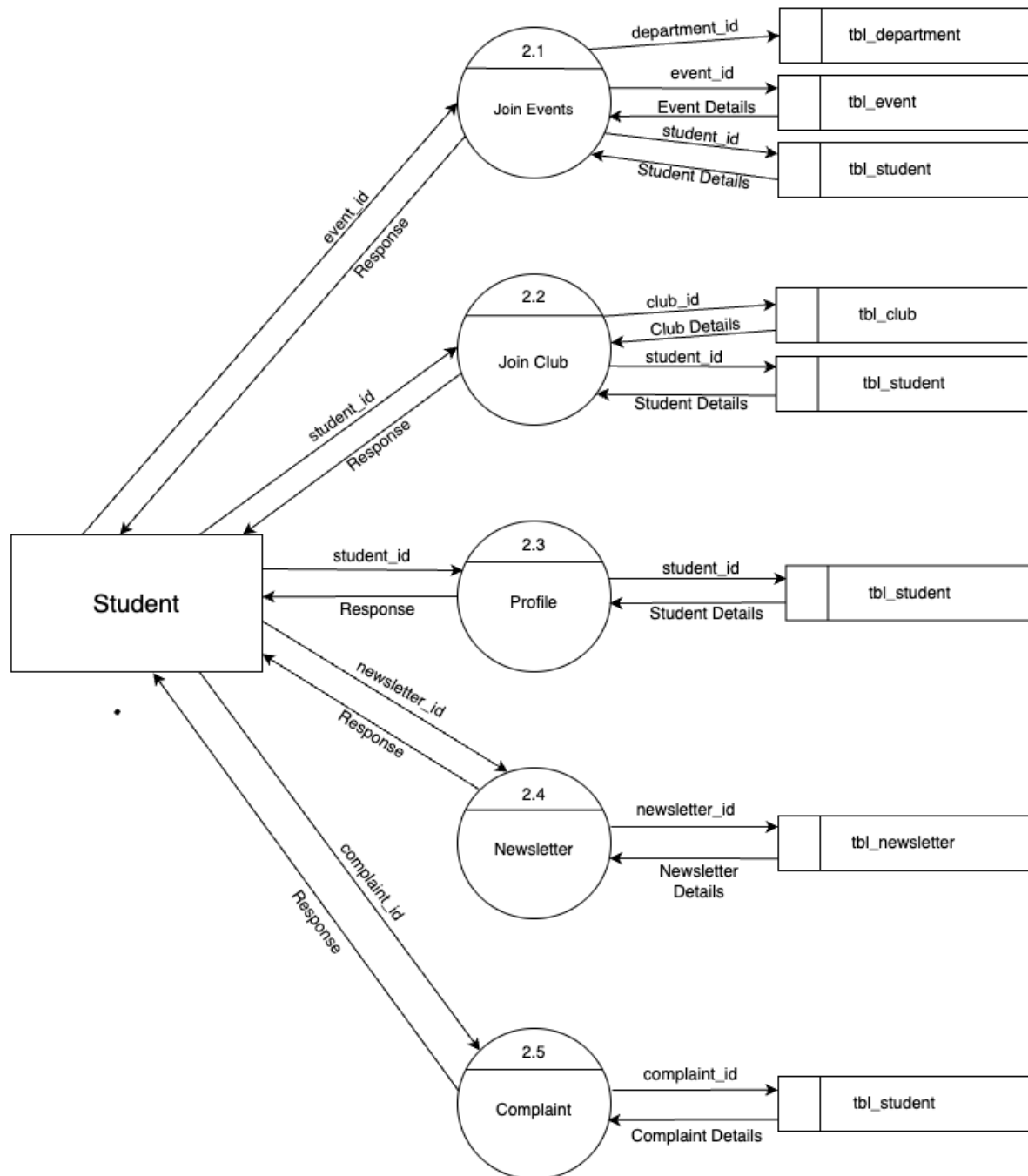
Level 1



Level 2



Level 3



3. SYSTEM DESIGN

3.1 INPUT DESIGN

The quality of the system input determines the quality of the system output. **Input specification** describes how data enters the system for processing. Well-designed input features ensure system reliability by preventing incorrect or inconsistent data entry. The input design also determines whether the user can interact efficiently with the system.

In **NSync**, most inputs are retrieved dynamically from the database to minimize errors and enhance usability. The system ensures structured data entry by utilizing dropdowns, selection menus, and validation checks.

User Roles & Input Responsibilities:

Admin

- Controls the overall system and has access to the admin panel.
- Can manage **faculty, students, events, clubs, and announcements**.
- Manages **user roles and authentication**.

Faculty

- Creates and manages events.
- Reviews and approves **club membership requests**.
- Posts **news and announcements**.

Students

- Registers for **events and clubs**.
- Receives real-time **notifications and updates**.
- Collaborates on **projects with peers and faculty**.

3.2 OUTPUT DESIGN

Output design is crucial in ensuring users receive relevant and easy-to-understand information. The system's outputs must be accurate, well-structured, and accessible in various formats (e.g., dashboards, reports, notifications).

Types of Output in NSync:

- **Event Updates:** Students receive notifications for upcoming events.
- **Club Membership Status:** Students and faculty can view membership details.
- **User Dashboards:** Provides personalized summaries for **students, faculty, and administrators**.
- **Announcements & News Feeds:** Displays **college updates, achievements, and notices**.
- **Activity Reports:** Generates **statistical insights on event participation and club activities**.

Users generally evaluate a system based on its output. Therefore, NSync's output mechanisms are **user-friendly, dynamic, and visually appealing**.

3.3 DATABASE DESIGN

Database design plays a crucial role in ensuring efficient data storage, retrieval, and security. NSync follows a structured database architecture, optimizing performance while maintaining **data integrity and security**.

Key Steps in Database Design:

1. **Identifying Data Requirements:** Understanding relationships between users, events, clubs, and system functionalities.
2. **Conceptual Design:** Creating an **Entity-Relationship Diagram (ERD)** to map out data relationships.
3. **Logical Design:** Defining tables, columns, primary keys, and foreign key constraints while ensuring **database normalization**.
4. **Physical Design:** Optimizing **storage structures, indexing, and query performance**.
5. **Data Integrity & Security:** Implementing validation mechanisms, access controls, and **encryption techniques**.
6. **Scalability & Performance:** Using **sharding, indexing, and caching** to ensure smooth performance with increasing system load.

Database Tables in NSync:

1. **Users Table** – Stores student, faculty, and admin details.
2. **Events Table** – Contains event information (name, date, organizer, participants).
3. **Clubs Table** – Manages club details and student memberships.
4. **Announcements Table** – Stores college-wide updates and news.
5. **Notifications Table** – Handles event reminders and system alerts
6. **Project Collaboration Table** – Facilitates student and faculty collaboration.

By structuring NSync's **database efficiently**, the system ensures **fast data access, seamless user experience, and high security** for all stakeholders.

The various database tables that are used in this project are the following:

1. tbl_department

Primary Key: department_id

Foreign Key: NIL

Description: Stores data about the Departments.

SL NO	NAME	DATA TYPE	SIZE	DESCRIPTION
1	department_id	INT	30	Unique ID of Department
2	department_name	VARCHAR	30	Name of Department

2. tbl_faculty

Primary Key: faculty_id

Foreign Key: department_id

Description: Stores data about the faculty.

SL NO	NAME	DATA TYPE	SIZE	DESCRIPTION
1	faculty_id	INT	30	Unique ID of Faculty
2	faculty_name	VARCHAR	30	Name of faculty
3	faculty_email	VARCHAR	30	E-mail of Faculty
4	faculty_password	VARCHAR	30	Password of Faculty
5	faculty_contact	INT	10	Contact of Faculty
6	faculty_photo	VARCHAR	200	Photo of Faculty
7	faculty_designation	VARCHAR	30	Designation of Faculty
8	department_id	INT	30	Unique ID of Department

3. tbl_class

Primary Key: class_id

Foreign Key: faculty_id

Description: Stores data about academic years.

SL NO	NAME	DATA TYPE	SIZE	DESCRIPTION
1	class_id	INT	30	Unique ID for Class
2	faculty_id	INT	30	Unique ID of Faculty
3	academic_year	INT	30	Academic Year

4. tbl_student

Primary Key: student_id

Foreign Key: department_id, academic_year

Description: Stores data about students.

SL NO	NAME	DATA TYPE	CONSTRAINTS	DESCRIPTION
1	student_id	INT	30	Unique ID of Student
2	student_name	VARCHAR	30	Name of Student
3	student_email	VARCHAR	30	E-mail of Student
4	student_password	VARCHAR	30	Password of Student
5	student_photo	VARCHAR	200	Photo of Student
6	student_status	VARCHAR	30	Status of Student
7	student_admno	INT	30	Admission Number of Student
8	student_contact	INT	10	Contact of Student
9	department_id	INT	30	Unique ID of Department
10	academic_year	INT	30	Academic Year

5. tbl_club

Primary Key: club_id

Foreign Key: faculty_id, student_id

Description: Stores data about clubs.

SL NO	NAME	DATA TYPE	SIZE	DESCRIPTION
1	club_id	INT	30	Unique ID of Club
2	club_name	VARCHAR	30	Name of Club
3	faculty_id	INT	30	Unique ID of Faculty
4	student_id	INT	30	Unique ID of Student

6. tbl_members

Primary Key: member_id

Foreign Key: club_id, student_id

Description: Stores data about club members.

SL NO	NAME	DATA TYPE	SIZE	DESCRIPTION
1	member_id	INT	30	Unique ID of Member
2	club_id	INT	30	Unique ID of Club
3	student_id	INT	30	Unique ID of Student
4	member_status	VARCHAR	30	Member Status

7. tbl_participants

Primary Key: participant_id

Foreign Key: student_id, event_id

Description: Stores data about participants.

SL NO	NAME	DATA TYPE	SIZE	DESCRIPTION
1	participant_id	INT	30	Unique ID of Participant
2	participant_status	VARCHAR	30	Participant Status
3	student_id	INT	30	Unique ID of Student
4	event_id	INT	30	Unique ID of Event

8. tbl_event

Primary Key: event_id

Foreign Key: club_id, department_id

Description: Stores data about events.

SL NO	NAME	DATA TYPE	CONSTRAINTS	DESCRIPTION
1	event_id	INT	30	Unique ID of Event
2	event_name	VARCHAR	30	Name of Event
3	event_details	VARCHAR	30	Details of Event
4	event_fordate	DATE	30	For Date of Event
5	event_lastdate	DATE	30	Last Date for Event
6	event_participants	INT	30	Number of Event Participants
7	event_status	VARCHAR	30	Status of Event
8	event_venue	VARCHAR	30	Venue of Event
9	club_id	INT	30	Unique ID of Club

N-SYNC - CAMPUS EVENT MANAGEMENT APP

10	department_id	INT	30	Unique ID of Department
11	event_poster	VARCHAR	200	Event poster

9. tbl_newsletter

Primary Key: newsletter_id

Foreign Key: NIL

Description: Stores data about newsletters.

SL NO	NAME	DATA TYPE	SIZE	DESCRIPTION
1	newsletter_id	INT	30	Unique ID of News Letter
2	newsletter_title	VARCHAR	30	Title of News Letter
3	newsletter_content	VARCHAR	30	Content of News Letter
4	newsletter_image	VARCHAR	200	Image for News Letter

10. tbl_complaint

Primary Key: complaint_id

Foreign Key: user_id

Description: Stores data about complaints raised.

SL NO	NAME	DATA TYPE	SIZE	DESCRIPTION
1	complaint_id	INT	30	Unique ID of Complaint
2	complaint_title	VARCHAR	30	Title of Complaint
3	complaint_content	VARCHAR	30	Content of Complaint
4	complaint_screenshot	VARCHAR	200	Image of Complaint
5	complaint_date	DATE	30	Date of Complaint

6	complaint_reply	VARCHAR	30	Reply to Complaint
7	complaint_status	VARCHAR	30	Status of Complaint
8	user_id	INT	30	Unique ID of User

11. tbl_ban

Primary Key: ban_id

Foreign Key: student_id

Description: Stores data about banned students.

SL NO	NAME	DATA TYPE	SIZE	DESCRIPTION
1	ban_id	INT	30	Unique ID for Ban
2	student_id	INT	30	Unique ID of Student
3	ban_status	INT	30	Status of Ban
4	ban_days	INT	30	Number of Days Banned

4. SYSTEM TESTING AND IMPLEMENTATION

4.1 SYSTEM TESTING

Testing plays a vital role in the development of the Presence Pro, serving as a quality assurance process to validate the app'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 app from the end users' perspective. Manual and automated testing techniques will be employed to catch defects early, ensure compliance with requirements, and optimize the app'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. 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.

SYSTEM INTEGRITY refers to the proper functioning of hardware and programs, appropriate physical security and safety against external threats such as fires dropping 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 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

NSync has significant potential for expansion, with various features planned for future updates to improve usability, engagement, and system intelligence.

1. Advanced Performance Analytics:

- Integrate **detailed performance tracking** with visual reports, graphs, and insights.
- Provide **personalized recommendations** for event participation based on user engagement.
- Generate **real-time analytics** on event popularity, student activity, and faculty interactions.

2. Gamification Features:

- Introduce **badges and rewards** for active participation in events and clubs.
- Implement a **leaderboard system** to highlight student engagement and contributions.
- Add **daily challenges and streaks** to encourage consistent participation.

3. AI-Powered Event & Club Recommendations:

- Use AI to **suggest events and clubs** based on user interests and past participation.
- Implement **smart notifications** to remind students about relevant opportunities.
- Develop **predictive analytics** to optimize event scheduling and participation rates.

4. Multilingual Support:

- Add support for **multiple languages** to enhance accessibility for diverse student groups.
- Enable **localized content and announcements** for region-specific needs.
- Introduce **language-based communication tools** for better faculty-student interaction.

5. AI Chatbot for Assistance:

- Implement an **AI-powered chatbot** to assist users with **event details, club queries, and system navigation**.
- Provide **real-time support** for technical issues and feature inquiries.
- Enable **voice-based interaction** for easy accessibility and convenience.

6. ERP Integration & Academic Collaboration:

- Integrate with **college ERP systems** for seamless student data synchronization.
- Add features for **academic project collaboration**, allowing students and faculty to manage coursework alongside events.
- Enhance **document sharing** and **task tracking** for streamlined group work.

7. Offline Access & Push Notifications:

- Enable limited **offline access** to event schedules and club details.
- Implement **push notifications** for event reminders, announcements, and club updates.
- Provide **custom notification settings** for a personalized user experience.

With these enhancements, NSync will continue evolving as a **powerful, intelligent, and user-friendly** college event and activity management platform.

8. CONCLUSION

The proposed **NSync system** offers a **comprehensive and interactive** platform for managing college events, clubs, and student activities, addressing the inefficiencies of traditional event coordination methods. By leveraging **Flutter for cross-platform development** and **Supabase for real-time database management**, the system ensures a **seamless and engaging user experience**.

With its **intuitive interface, structured event management, and role-based access**, NSync enables students and faculty members to **participate efficiently in campus activities**. The system's scalability and planned future enhancements such as **AI-powered event recommendations, gamification, and ERP integration** make it a **robust and evolving solution** for campus-wide engagement and management.

NSync is **designed for long-term adaptability**, ensuring continuous improvements and **enhanced user experience** in the dynamic academic environment.

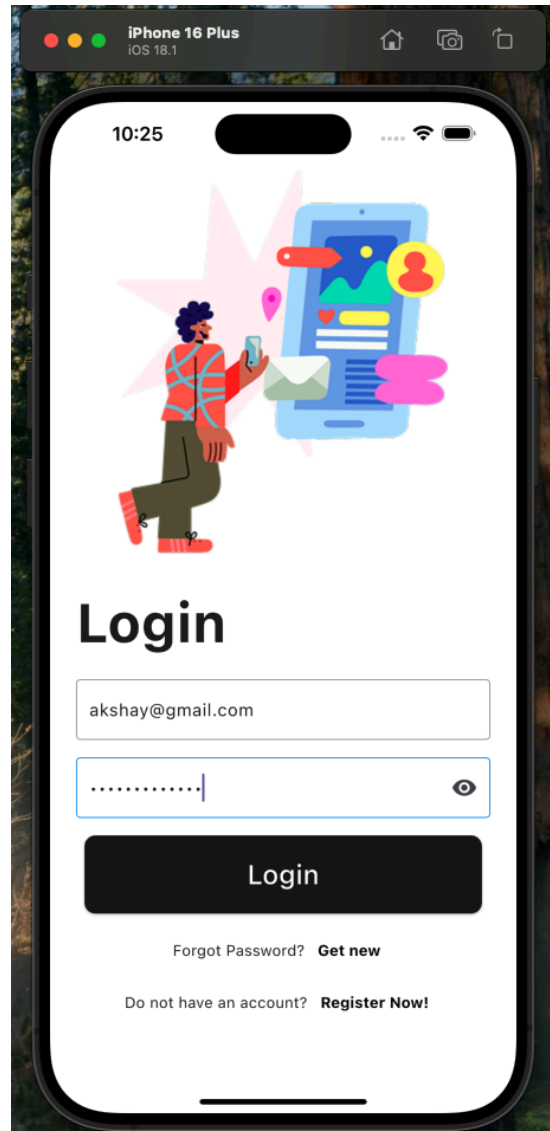
9. BIBLIOGRAPHY

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>

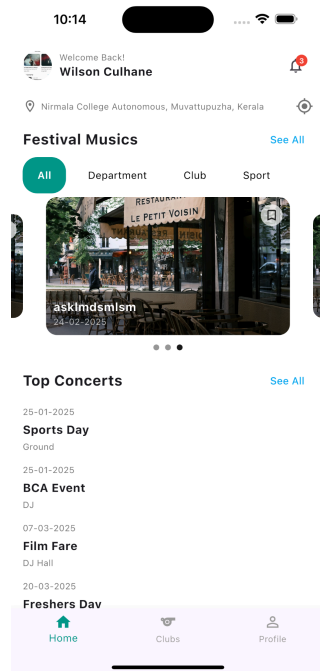
10. APPENDIX

10.1 Screenshots

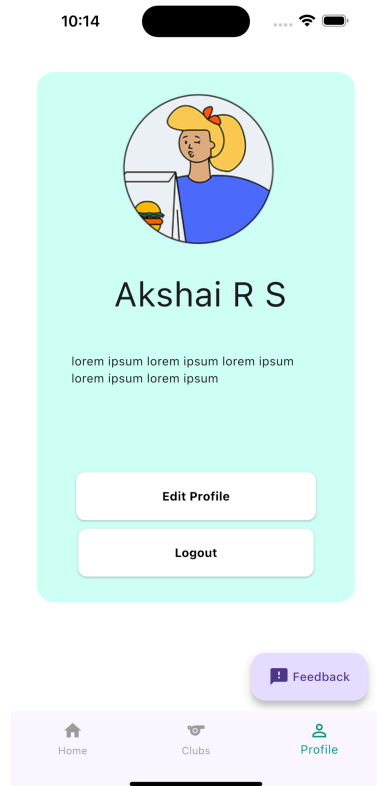
10.1.1 Student Login



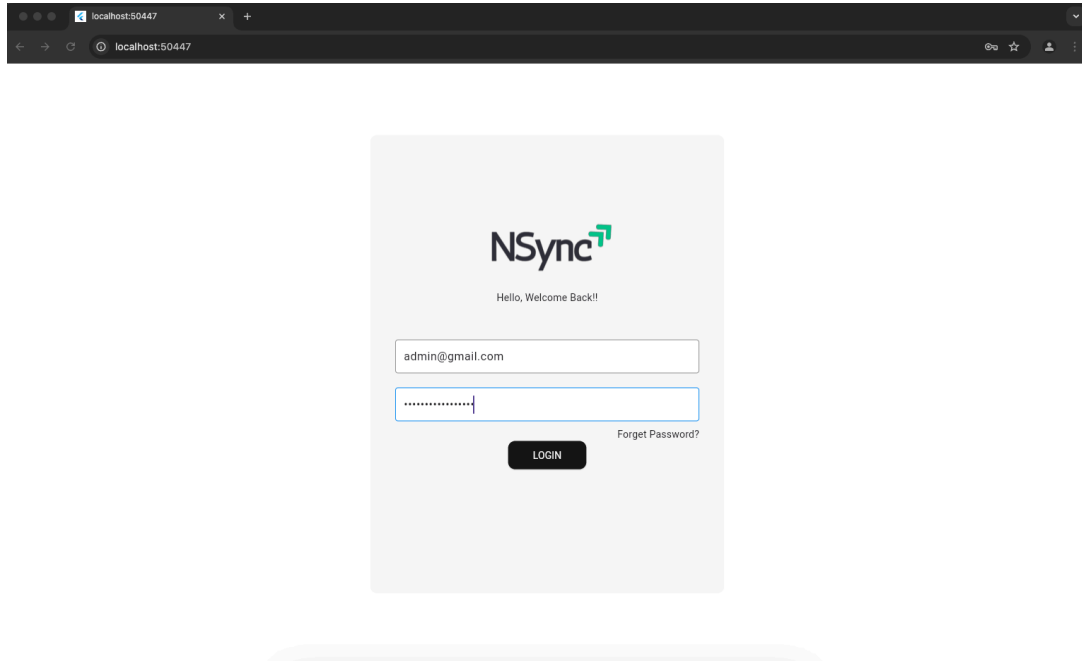
10.1.2 Events & NewsLetter Page



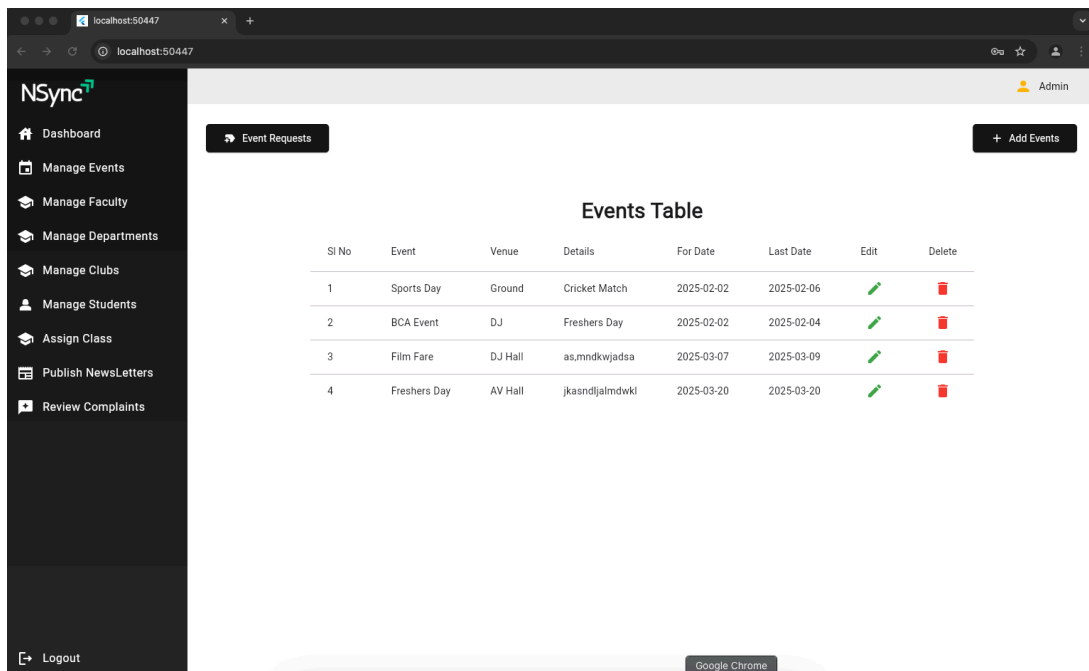
10.1.3 Student Profile



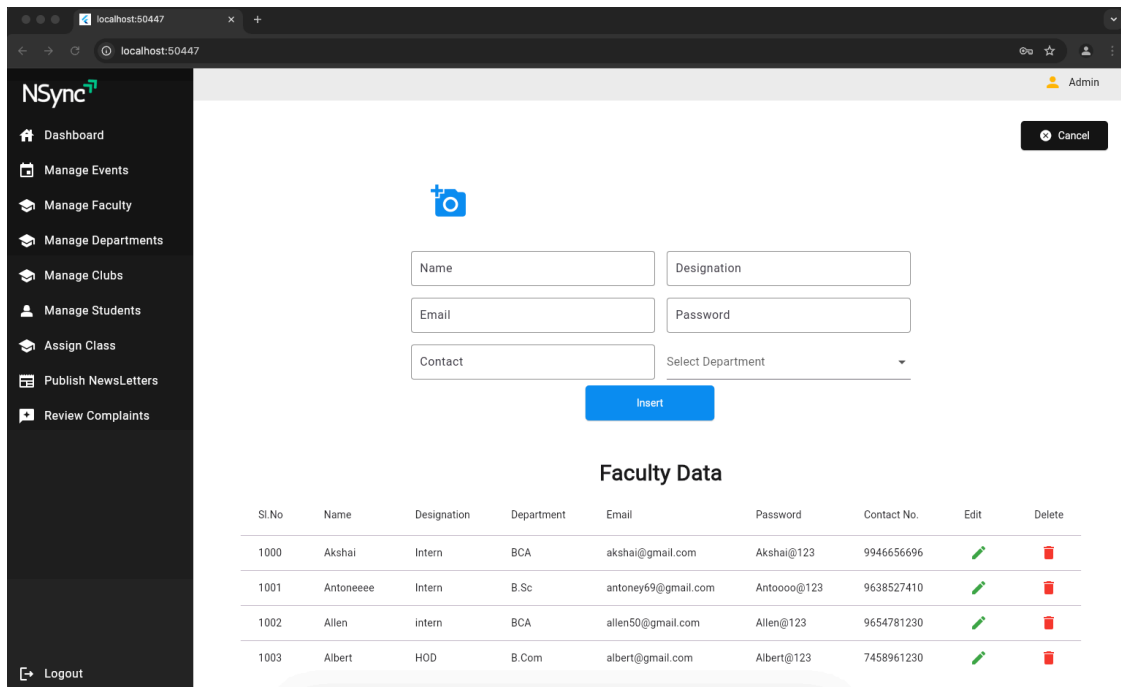
10.1.4 Admin Login



10.1.5 Add Events



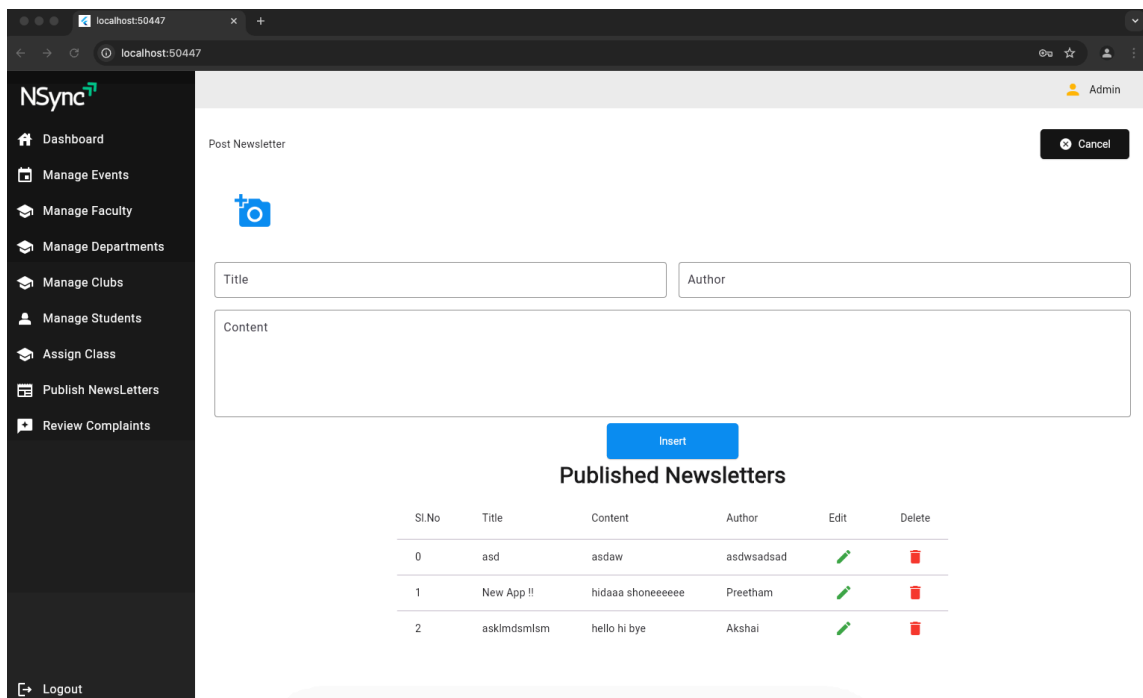
10.1.6 Manage Faculty



Faculty Data

Sl.No	Name	Designation	Department	Email	Password	Contact No.	Edit	Delete
1000	Akshai	Intern	BCA	akshai@gmail.com	Akshai@123	9946656696		
1001	Antoneeee	Intern	B.Sc	antoney69@gmail.com	Antoooo@123	9638527410		
1002	Allen	intern	BCA	allen50@gmail.com	Allen@123	9654781230		
1003	Albert	HOD	B.Com	albert@gmail.com	Albert@123	7458961230		

10.1.7 Manage Newsletter



Published Newsletters

Sl.No	Title	Content	Author	Edit	Delete
0	asd	asdaw	asdwadsad		
1	New App !!	hidaaa shoneeeeeee	Preetham		
2	asklmsmlsm	hello hi bye	Akshai		

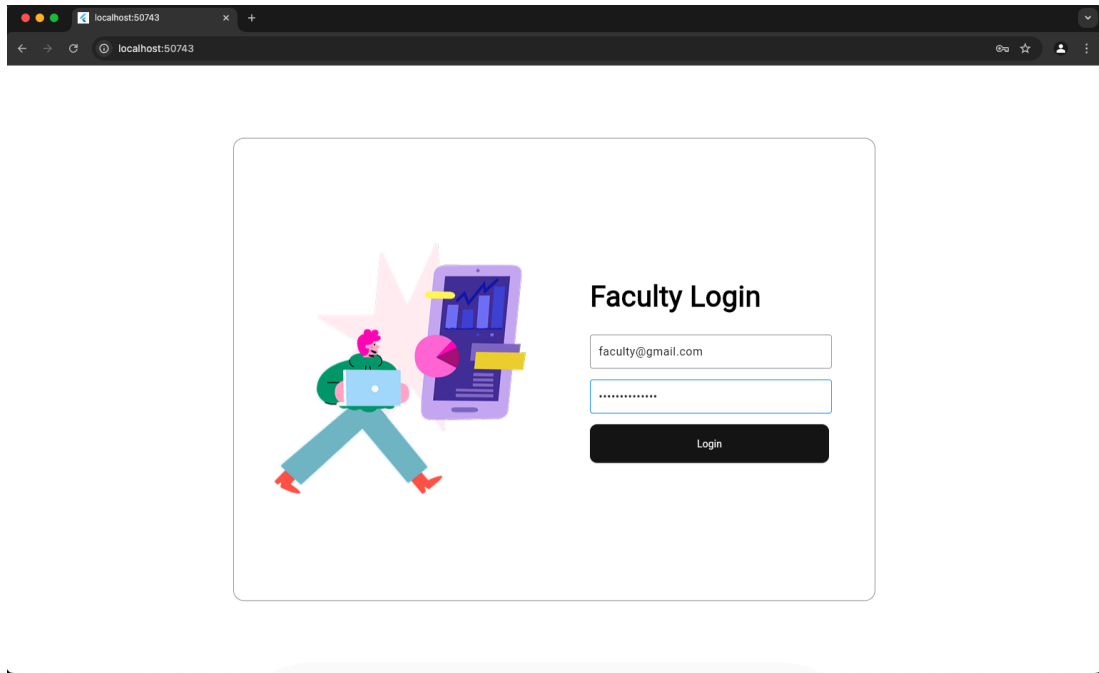
10.1.8 Manage Students

The screenshot shows the 'Manage Students' page of the N-Sync application. The sidebar on the left contains the following menu items: Dashboard, Manage Events, Manage Faculty, Manage Departments, Manage Clubs, Manage Students (active), Assign Class, Publish NewsLetters, and Review Complaints. The main content area features a form for adding new students with fields for Name, Admission_No, Email, Password, Phone, and Academic_Year. A 'Select Department' dropdown menu is also present. A blue 'Insert' button is located below the form. Below the form, a table titled 'Students Data' displays the following information:

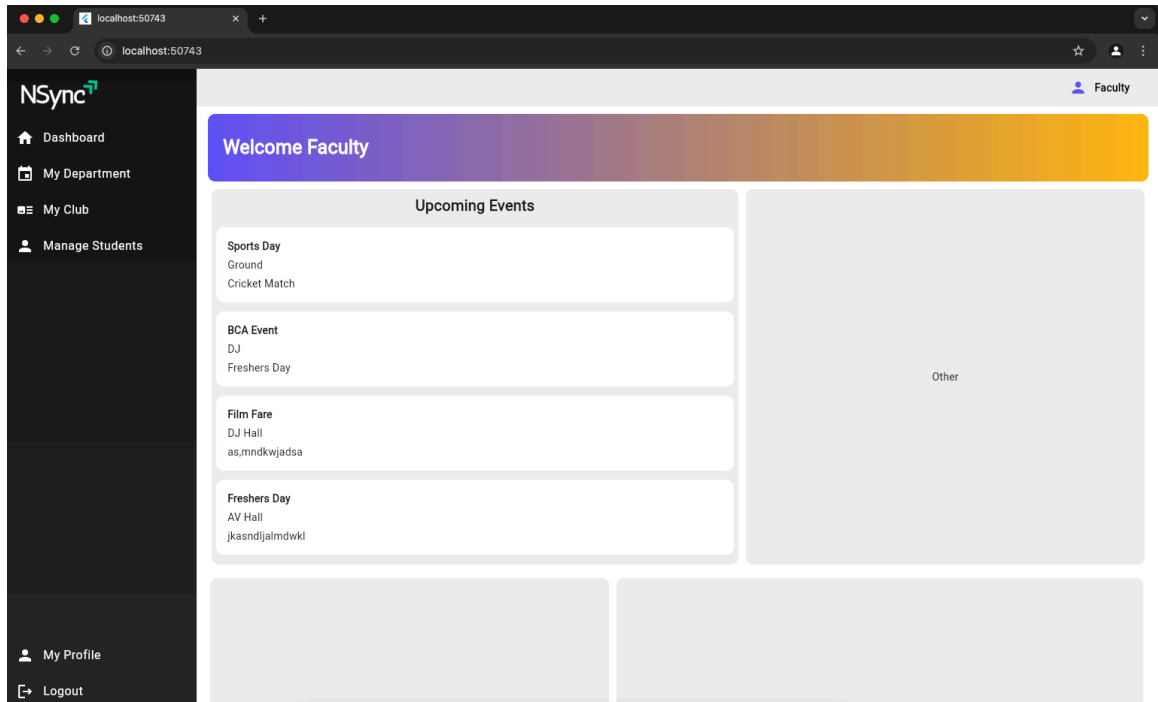
SL.No	Name	Adm No.	Email	Password	Contact No.	Department	Academic Year	Delete
1000	Bimi	2454	bimi@gmail.com	Bimi@123	9784563210	BCA	2022	
1001	Akshai	2408	akshai27@gmail.com	Akshai@123	9946656696	B.Sc	2022	
1002	Benny	2409	benny@gmail.com	Benny@123	9874561230	B.Com	2022	
1003	Jose	2469	jose@gmail.com	Jose@123	7894561230	Physics	2023	

The bottom of the sidebar contains a 'Logout' button.

10.1.9 Faculty Login



10.1.10 Faculty Dashboard



10.1.11 Request Events

The screenshot shows the NSync app interface. On the left is a dark sidebar with navigation links: Dashboard, My Department, My Club, Manage Students, My Profile, and Logout. The main content area is titled 'Department Events' and lists four events in a light gray box:

- Sports Day**
Ground
Cricket Match
- BCA Event**
DJ
Freshers Day
- Film Fare**
DJ Hall
as,mndkwjadsa
- Freshers Day**
AV Hall
jksndjajmndwkl

At the top right of the main area, there is a 'Faculty' user indicator and a 'Host Event' button.

10.1.12 Manage Students

The screenshot shows the NSync app interface for managing students. The sidebar is the same as in the previous screenshot. The main content area has a 'Cancel' button at the top right and a form to add a new student. The form fields are:

- Name: Tom
- Email: tom@gmail.com
- Password: 9876543210
- Contact No.: 1234
- Department: Tom@123
- Year: 2022

Below the form is a dropdown menu set to 'Physics' and an 'Insert' button. Below the form is a table titled 'Students Data' with the following data:

SL.No	Name	Adm.No.	Email	Password	Contact No.	Department	Delete
1000	Bimi	2454	bimi@gmail.com	Bimi@123	9784563210	BCA	
1001	Akshai	2408	akshai27@gmail.com	Akshai@123	9946656696	B.Sc	
1002	Benny	2409	benny@gmail.com	Benny@123	9874561230	B.Com	
1003	Jose	2469	jose@gmail.com	Jose@123	7894561230	Physics	

10.2 Code

10.2.1 main.dart

```
import 'package:flutter/material.dart';
import 'package:nsync_admin/screen/admin_home.dart';
import 'package:supabase_flutter/supabase_flutter.dart';

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

'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6I6Imd4b213a3B3b3htaGR0ZHN4anBoIiwicm9sZSI6ImFub24iLCJpYXQiOiJlZ3MzQzNDU5ODAsImV4cCI6MjA0OTkyMTk4MH0.AksgXgzqkpAGnGxsypvcaotmPeFSdyt1AalMljjdLdw',
  );
  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: AdminHome(),
    );
  }
}
```


10.2.2 Login.dart

```
import 'package:flutter/material.dart';
import 'package:nsync_admin/components/form_validation.dart';
import 'package:nsync_admin/main.dart';
import 'package:nsync_admin/screen/admin_home.dart';
import 'package:cherry_toast/resources/arrays.dart';
import 'package:cherry_toast/cherry_toast.dart';

class Login1 extends StatefulWidget {
  const Login1({super.key});

  @override
  State<Login1> createState() => _Login1State();
}

class _Login1State extends State<Login1> {
  final TextEditingController _adminEmailController =
    TextEditingController();
  final TextEditingController _adminPassController =
    TextEditingController();
  final formKey = GlobalKey<FormState>();

  //sign in

  Future<void> signIn() async {
    try {
      await supabase.auth.signInWithPassword(
        password: _adminPassController.text,
        email: _adminEmailController.text);
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(
          builder: (context) => AdminHome(),

```

```
    ));  
  } catch (e) {  
    print("Error occur in login:$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.');
```

```
  }  
}  
  
@override  
Widget build(BuildContext context) {  
  return Scaffold(  
    body: Container(  
      decoration: BoxDecoration(color: Colors.white),  
      child: SafeArea(  
        child: Center(  
          child: Container(  
            decoration: BoxDecoration(  
              color: Color.fromARGB(253, 246, 246, 246),  
              borderRadius: BorderRadius.circular(10)),  
            width: 500,  
            height: 650,  
            child: Form(  
              key: formKey,  
              child: Column(  
                children: [  
                  SizedBox(  
                    height: 120,  
                  ),  
                  Image.asset('../assets/logo200.png'),  
                  SizedBox(  
                    height: 120,  
                  ),  
                ],  
              ),  
            ),  
          ),  
        ),  
      ),  
    ),  
  );  
}
```

```
        height: 20,
      ),
      Text("Hello, Welcome Back!!"),
      SizedBox(
        height: 50,
      ),
      Padding(
        padding: EdgeInsets.symmetric(horizontal: 35.0),
        child: TextFormField(
          validator: (value) =>
FormValidation.validateEmail(value),
          controller: _adminEmailController,
          decoration: InputDecoration(
            fillColor: Colors.white,
            filled: true,
            hintText: "Enter You ID",
            enabledBorder: OutlineInputBorder(
              borderSide: BorderSide(color:
Colors.grey)),
            focusedBorder: OutlineInputBorder(
              borderSide: BorderSide(color:
Colors.blue))),
        ),
      ),
      SizedBox(
        height: 20,
      ),
      Padding(
        padding: EdgeInsets.symmetric(horizontal: 35.0),
        child: TextFormField(
          validator: (value) =>
          FormValidation.validatePassword(value),
          controller: _adminPassController,
          decoration: InputDecoration(
            fillColor: Colors.white,
            filled: true,
```

```
        hintText: "Password",
        enabledBorder: OutlineInputBorder(
            borderSide: BorderSide(color:
Colors.grey)),
        focusedBorder: OutlineInputBorder(
            borderSide: BorderSide(color:
Colors.blue)),
    ),
    obscureText: true,
  ),
),
SizedBox(
  height: 8,
),
Row(
  mainAxisAlignment: MainAxisAlignment.end,
  children: [
    Padding(
      padding: EdgeInsets.only(right: 35.0),
      child: Text("Forget Password?"),
    )
  ],
),
ElevatedButton(
  style: ElevatedButton.styleFrom(
    backgroundColor: Color(0xFF161616),
    shape: RoundedRectangleBorder(
      borderRadius:
BorderRadius.circular(10)),
    padding: EdgeInsets.symmetric(
      horizontal: 35, vertical: 18)),
  onPressed: () {
    Navigator.pushReplacement(
      context,
      MaterialPageRoute(
        builder: (context) => AdminHome(),

```

```
        ));

        /* if (formKey.currentState!.validate()) {
            signIn();
        } */ //sign in function
    },
    child: Text(
        "LOGIN",
        style: TextStyle(color: Colors.white),
    ))
],
),
),
),
),
)),
),
);
}
}
```

10.2.3 manage_events.dart

```
import 'package:flutter/material.dart';
import 'package:nsync_admin/components/insert_form.dart';
import 'package:nsync_admin/main.dart';
import 'package:intl/intl.dart';
import 'package:nsync_admin/screen/requested_events.dart';

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

    @override
    State<EventsScreen> createState() => _EventsScreenState();
}
```

```
}

class _EventsScreenState extends State<EventsScreen>
  with SingleTickerProviderStateMixin {
  bool _isFormVisible = false; // To manage form visibility
  final Duration _animationDuration = const Duration(milliseconds:
300);

  //controllers
  final TextEditingController _eventController =
TextEditingController();
  final TextEditingController _evDetailController =
TextEditingController();
  final TextEditingController _evVenueController =
TextEditingController();
  final TextEditingController _evForDateController =
TextEditingController();
  final TextEditingController _evLastDateController =
TextEditingController();

  //list to store tbl data
  List<Map<String, dynamic>> EventList = [];

  //insert
  Future<void> eventInsert() async {
    try {
      String Name = _eventController.text;
      String Details = _evDetailController.text;
      String Venue = _evVenueController.text;
      String ForDate = _evForDateController.text;
      String LastDate = _evLastDateController.text;
      await supabase.from('tbl_events').insert({
        'event_name': Name,
        'event_details': Details,
        'event_venue': Venue,
        'event_fordate': ForDate,
```

```
        'event_lastdate': LastDate,
        'event_status': 1 // to auto approve events added by admin
    });
    ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
      content: Text(
        "Event Details Inserted Sucessfully",
        style: TextStyle(color: Colors.white),
      ),
      backgroundColor: Colors.green,
    ));
    _eventController.clear();
    _evDetailController.clear();
    _evVenueController.clear();
    _evForDateController.clear();
    _evLastDateController.clear();
  } catch (e) {
    print("ERROR INSERTING DATA: $e");
  }
}

//Select

Future<void> fetchEvents() async {
  try {
    final response =
                                                                    await
supabase.from('tbl_events').select().eq('event_status', 1);
    setState(() {
      EventList = response;
    });
    fetchEvents();
  } catch (e) {
    print("ERROR FETCHING DATA: $e");
  }
}
```

```
//Edit

int eid = 0;

Future<void> editEvent() async {
  try {
    await supabase.from('tbl_events').update({
      'event_name': _eventController.text,
      'event_venue': _evVenueController.text,
      'event_details': _evDetailController.text,
      'event_fordate': _evForDateController.text,
      'event_lastdate': _evLastDateController.text
    }).eq('event_id', eid);
    fetchEvents();
    _eventController.clear();
    _evDetailController.clear();
    _evVenueController.clear();
    _evForDateController.clear();
    _evLastDateController.clear();
  } catch (e) {
    print("ERROR UPDATING DATA: $e");
  }
}

//Delete

Future<void> DelEvent(String did) async {
  try {
    await supabase.from("tbl_events").delete().eq("event_id", did);
    fetchEvents();
  } catch (e) {
    print("ERROR: $e");
  }
}

//select date
```



```
Future<void> _selectDate(
    BuildContext context, TextEditingController controller) async {
    DateTime? pickedDate = await showDatePicker(
        context: context,
        initialDate: DateTime.now(),
        firstDate: DateTime.now(), // Prevents past dates
        lastDate: DateTime.now().add(Duration(days: 365)), // Limits to
1 year
    );

    if (pickedDate != null) {
        setState(() {
            // Update the text field with the selected date
            controller.text =
DateFormat('yyyy-MM-dd').format(pickedDate);
        });
    }
}

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

@override
Widget build(BuildContext context) {
    return Padding(
        padding: const EdgeInsets.all(18.0),
        child: Column(
            children: [
                Row(
                    mainAxisAlignment: MainAxisAlignment.spaceBetween,
                    children: [
                        Padding(
                            padding: const EdgeInsets.all(8.0),
```

```
        child: ElevatedButton.icon(
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) =>
RequestedEvents() ));
          },
          label: Text(
            "Event Requests",
            style: TextStyle(color: Colors.white),
          ),
          style: ElevatedButton.styleFrom(
            backgroundColor: Color(0xFF161616),
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(5)),
            padding:
              EdgeInsets.symmetric(horizontal: 25,
vertical: 18)),
          icon: Icon(
            Icons.new_label,
            color: Colors.white,
          ),
        )),
      Padding(
        padding: const EdgeInsets.all(10.0),
        child: ElevatedButton.icon(
          style: ElevatedButton.styleFrom(
            backgroundColor: Color(0xFF161616),
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(5)),
            padding:
              EdgeInsets.symmetric(horizontal: 25,
vertical: 18)),
          onPressed: () {
            setState(() {
```

```
        _isFormVisible =
            !_isFormVisible; // Toggle form visibility
    });
},
label: Text(
    _isFormVisible ? "Cancel" : "Add Events",
    style: TextStyle(color: Colors.white),
),
icon: Icon(
    _isFormVisible ? Icons.cancel : Icons.add,
    color: Colors.white,
),
),
),
],
),
AnimatedSize(
    duration: _animationDuration,
    curve: Curves.easeInOut,
    child: _isFormVisible
        ? Form(
            child: SizedBox(
                width: 700,
                child: Column(
                    children: [
                        Row(
                            children: [
                                Expanded(
                                    child: Padding(
                                        padding: const EdgeInsets.all(8.0),
                                        child: TextFieldStyle(
                                            inputController: _eventController,
                                            label: "Event Name",
                                        ),
                                    ),
                                Expanded(
```

```
        child: Padding(
          padding: EdgeInsets.all(8),
          child: TextFieldStyle(
            label: "Venue",
            inputController: _evVenueController,
          ),
        )),
      ],
    ),
    Row(
      children: [
        Expanded(
          child: Padding(
            padding: EdgeInsets.all(8),
            child: TextFieldStyle(
              label: "Event Details",
              inputController:
_evDetailController),
          )),
      ],
    ),
    Row(
      children: [
        Expanded(
          child: Padding(
            padding: EdgeInsets.all(8),
            child: TextFormField(
              controller: _evForDateController,
              readOnly: true,
              decoration: const InputDecoration(
                labelText: "For_Date",
                enabledBorder: OutlineInputBorder(
                  borderSide:
BorderSide(color:
Colors.grey)),
```

```
suffixIcon:
Icon(Icons.calendar_today),
),
onTap: () =>
_selectDate(context,
_evForDateController),
),
)),
Expanded(
  child: Padding(
padding: EdgeInsets.all(8),
child: TextFormField(
  controller: _evLastDateController,
  readOnly: true,
  decoration: const InputDecoration(
    labelText: "Last_date",
    enabledBorder: OutlineInputBorder(
      borderSide:
        BorderSide(color:
Colors.grey)),
suffixIcon:
Icon(Icons.calendar_today),
),
onTap: () =>
_selectDate(context,
_evLastDateController),
),
))
],
),
ElevatedButton(
  style: ElevatedButton.styleFrom(
    backgroundColor: Colors.blue,
    padding: EdgeInsets.symmetric(
      vertical: 22, horizontal: 70),
    shape: RoundedRectangleBorder(
```

```
borderRadius:
BorderRadius.circular(5))),
    onPressed: () {
      if (eid == 0) {
        eventInsert();
      } else {
        editEvent();
      }
    },
    child: Text(
      "Insert",
      style: TextStyle(color: Colors.white),
    )),
  ],
),
))
: Container(),
),
SizedBox(
  height: 50,
),
Container(
  child: Center(
    child: Text("Events Table",
      style: TextStyle(fontWeight: FontWeight.bold,
fontSize: 30)),
  ),
),
Container(
  child: Padding(
    padding: EdgeInsets.all(8),
    child: DataTable(
      columns: [
        DataColumn(label: Text("Sl No")),
        DataColumn(label: Text("Event")),
        DataColumn(label: Text("Venue")),
```

```
        DataColumn(label: Text("Details")),
        DataColumn(label: Text("For Date")),
        DataColumn(label: Text("Last Date")),
        DataColumn(label: Text("Edit")),
        DataColumn(label: Text("Delete"))
    ],
    rows: EventList.asMap().entries.map((entry) {
        return DataRow(cells: [
            DataCell(Text((entry.key + 1).toString())),
            DataCell(Text(entry.value['event_name'])),
            DataCell(Text(entry.value['event_venue'])),
            DataCell(Text(entry.value['event_details'])),
            DataCell(Text(entry.value['event_fordate'])),
            DataCell(Text(entry.value['event_lastdate'])),
            DataCell(IconButton(
                icon: const Icon(Icons.edit, color:
Colors.green),
                onPressed: () {
                    setState(() {
                        _evForDateController.text =
                            entry.value['event_fordate'];
                        _evLastDateController.text =
                            entry.value['event_lastdate'];
                        _eventController.text =
entry.value['event_name'];
                        _evDetailController.text =
                            entry.value['event_details'];
                        _evVenueController.text =
                            entry.value['event_venue'];

                        eid = entry.value['event_id'];
                        _isFormVisible = true;
                        print(eid);
                    });
                },
            )),
        ]),
    )
```

```
        DataCell(IconButton(
            icon: const Icon(Icons.delete, color:
Colors.red),
            onPressed: () {
DelEvent(entry.value['event_id'].toString());
                //delete function
            },
        ))
    ].toList()),
    ),
)
],
),
);
}
```

manage_club.dart

```
import 'package:flutter/material.dart';
import 'package:nsync_admin/components/insert_form.dart';
import 'package:nsync_admin/main.dart';

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

  @override
  State<ClubsScreen> createState() => _ClubsScreenState();
}

class _ClubsScreenState extends State<ClubsScreen>
  with SingleTickerProviderStateMixin {
```



```
bool _isFormVisible = false; // To manage form visibility
final Duration _animationDuration = const Duration(milliseconds:
300);

String? selectedFac;
String? selectedStud;

final TextEditingController _clubController =
TextEditingController();

// list to store tbl data for displaying
List<Map<String, dynamic>> Clublist = []; // display table
List<Map<String, dynamic>> FacList = [];
List<Map<String, dynamic>> StudList = [];

//insert
Future<void> insertClub() async {
  try {
    String club = _clubController.text;

    // Ensure both selections are made
    if (selectedFac == null || selectedStud == null) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Please select both Faculty and
Student")),
      );
      return;
    }

    await supabase.from("tbl_club").insert({
      "club_name": club,
      "faculty_id": selectedFac, // Use the selected faculty UUID
      "student_id": selectedStud, // Use the selected student UUID
    });

    fetchClub();
  }
}
```

```
        ScaffoldMessenger.of(context).showSnackBar(const SnackBar(  
            content: Text("Data Added", style: TextStyle(color:  
Colors.white)),  
            backgroundColor: Colors.green,  
        ));  
    } catch (e) {  
        print("ERROR ADDING DATA: $e");  
    }  
}  
  
//select  
  
Future<void> fetchClub() async {  
    try {  
        final response = await supabase  
            .from('tbl_club')  
            .select('*', tbl_faculty (faculty_name), tbl_student  
(student_name));  
        setState(() {  
            Clublist = response;  
        });  
    } catch (e) {  
        print("ERROR FETCHING DATA: $e");  
    }  
}  
  
Future<void> fetchFaculty() async {  
    try {  
        final response = await supabase.from('tbl_faculty').select();  
        if (response.isNotEmpty) {  
            setState(() {  
                FacList = response;  
            });  
        }  
    } catch (e) {  
        print("ERROR FETCHING FACULTY: $e");  
    }  
}
```

```
    }  
  }  
  
  Future<void> fetchStudent() async {  
    try {  
      final response = await supabase.from('tbl_student').select();  
      if (response.isNotEmpty) {  
        setState(() {  
          StudList = response;  
        });  
      }  
    } catch (e) {  
      print('ERROR FETCHING STUDENTS: $e');  
    }  
  }  
  
  //delete  
  
  Future<void> delCLub(String did) async {  
    try {  
      await supabase.from('tbl_club').delete().eq("club_id", did);  
      fetchClub();  
    } catch (e) {  
      print("ERROR: $e");  
    }  
  }  
  
  //edit  
  int eid = 0;  
  
  Future<void> editclub() async {  
    try {  
      await supabase  
        .from('tbl_club')  
        .update({'club_name': _clubController.text}).eq('club_id',  
eid);  
    }  
  }  
}
```

```
        fetchClub();
        _clubController.clear();
    } catch (e) {
        print("ERROR: $e");
    }
}

@override
void initState() {
    super.initState();
    fetchClub();
    fetchFaculty();
    fetchStudent();
}

@override
Widget build(BuildContext context) {
    return Padding(
        padding: const EdgeInsets.all(18.0),
        child: Column(
            children: [
                Row(
                    mainAxisAlignment: MainAxisAlignment.spaceBetween,
                    children: [
                        Text("Manage Clubs"),
                        Padding(
                            padding: const EdgeInsets.all(10.0),
                            child: ElevatedButton.icon(
                                style: ElevatedButton.styleFrom(
                                    backgroundColor: Color(0xFF161616),
                                    shape: RoundedRectangleBorder(
                                        borderRadius: BorderRadius.circular(5)),
                                    padding:
                                        EdgeInsets.symmetric(horizontal: 25,
vertical: 18)),
                                onPressed: () {
```

```
      setState(() {
        _isFormVisible =
          !_isFormVisible; // Toggle form visibility
      });
    },
    label: Text(
      _isFormVisible ? "Cancel" : "Add Club",
      style: TextStyle(color: Colors.white),
    ),
    icon: Icon(
      _isFormVisible ? Icons.cancel : Icons.add,
      color: Colors.white,
    ),
  ),
],
),
),
AnimatedSize(
  duration: _animationDuration,
  curve: Curves.easeInOut,
  child: _isFormVisible
    ? Form(
        child: Container(
          width: 700,
          padding: EdgeInsets.all(20),
          child: Column(
            children: [
              Row(
                children: [
                  Expanded(
                    child: Padding(
                      padding: const EdgeInsets.all(8.0),
                      child: TextFieldStyle(
                        inputController: _clubController,
                        label: "Club",
                      ),
                    ),

```

```
        )),
      ],
    ),
    Row(
      children: [
        Expanded(
          child: Padding(
            padding: EdgeInsets.all(8),
            child: DropdownButtonFormField(
              value: selectedFac,
              hint: const Text("Select Faculty"),
              items: FacList.map((faculty) {
                return DropdownMenuItem(
                  value:
faculty['faculty_id'].toString(),
                  child:
Text(faculty['faculty_name']),
                );
              }).toList(),
              onChanged: (newValue) {
                setState(() {
                  selectedFac = newValue;
                });
              },
            ),
          ),
        Expanded(
          child: Padding(
            padding: EdgeInsets.all(8),
            child: DropdownButtonFormField(
              value: selectedStud,
              hint: const Text("Select Student"),
              items: StudList.map((student) {
                return DropdownMenuItem(
                  value:
student['student_id'].toString(),
```

```
child:
Text(student['student_name']),
    );
    }).toList(),
    onChanged: (newValue) {
      setState(() {
        selectedStud = newValue;
      });
    },
  ),
))
],
),
ElevatedButton(
  style: ElevatedButton.styleFrom(
    backgroundColor: Color(0xFF017AFF),
    padding: EdgeInsets.symmetric(
      horizontal: 70, vertical: 22),
    shape: RoundedRectangleBorder(
      borderRadius:
BorderRadius.circular(5))),
  onPressed: () {
    if (eid == 0) {
      insertClub();
    } else {
      editclub();
    }
  },
  child: Text(
    "Insert",
    style: TextStyle(color: Colors.white),
  ))
],
),
))
: Container(),
```

```
    ),  
    Container(  
      child: Center(  
        child: Text("Clubs Table",  
          style: TextStyle(fontWeight: FontWeight.bold,  
fontSize: 30)),  
      ),  
    ),  
    SizedBox(height: 20),  
    Container(  
      decoration: BoxDecoration(  
        color: const Color.fromARGB(255, 249, 249, 249),  
        border: Border.all(color: Colors.grey, width: 1),  
        borderRadius: BorderRadius.circular(6),  
      ),  
      height: 500,  
      width: 850,  
      child: Padding(  
        padding: EdgeInsets.all(8),  
        child: DataTable(  
          columns: [  
            DataColumn(label: Text("Sl.No")),  
            DataColumn(label: Text("Club")),  
            DataColumn(label: Text("Faculty")),  
            DataColumn(label: Text("Student")),  
            DataColumn(label: Text("Edit")),  
            DataColumn(label: Text("Delete"))  
          ],  
          rows: Clublist.asMap().entries.map((entry) {  
            print(entry.value);  
            return DataRow(cells: [  
              DataCell(Text((entry.key + 1).toString())),  
              DataCell(Text(entry.value['club_name'])),  
              DataCell(  
Text(entry.value['tbl_faculty']['faculty_name'])),
```



```
        DataCell(  
  
        Text(entry.value['tbl_student']['student_name'])),  
        DataCell(IconButton(  
            icon: const Icon(Icons.edit, color:  
Colors.green),  
            onPressed: () {  
                setState(() {  
                    _clubController.text =  
entry.value['club_name'];  
                    eid = entry.value['club_id'];  
                    _isFormVisible = !_isFormVisible;  
                });  
            },  
        )),  
        DataCell(IconButton(  
            icon: const Icon(Icons.delete, color:  
Colors.red),  
            onPressed: () {  
                delCLub(entry.value['club_id'].toString());  
                //delete function  
            },  
        ))  
    ]);  
}).toList()),  
    ),  
    ],  
    ),  
);  
}
```