KIET EVENT MANAGEMENT APP

A PROJECT REPORT

Submitted By

(SACHIN LAL)
(2100290140114)
(VIDUSHI SHUKLA)
(2100290140145)

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Mr Shashank Bharadwaj
(Assistant Professor)

Submitted to
DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206
(JUNE 2023)

DECLARATION

I hereby declare that the work presented in this report entitled “KIET EVENT MANAGEMENT APP”, was carried out by me and my team mates. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute.

I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

I affirm that no portion of my work is plagiarized, and the experiments and results reportedin the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

Name : Sachin Lal (2100290140114)
Vidushi Shukla (2100290140145)

Branch : Master of Computer Application

Candidate Signature

CERTIFICATE

Certified that **SACHIN LAL (2100290140114), VIDUSHI SHUKLA (2100290140145),** have carried out the project work having **"KIET EVENT MANAGEMENT APP"** for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Technical University, Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself / herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date:29/05/2033

SACHIN LAL (2100290140114)

VIDUSHI SHUKLA (2100290140145)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Mr. Shashank Bharadwaj
Assistant Professor
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Signature of Internal Examiner **Signature of External Examiner**

Dr. Arun Tripathi
Head, Department of Computer Applications
KIET Group of Institutions,
Ghaziabad

ABSTRACT

The KIET Event Management App is a mobile application designed to simplify the process of organizing and managing college events. The app allows students, faculty, and staff to easily create and manage events, and communicate with attendees. It also includes features such as event calendars, RSVPs, and push notifications to keep users updated on upcoming events. The app is user-friendly, intuitive, and customizable, allowing users to personalize their experience and streamline their event planning process. Overall, the KIET Event Management App is a powerful tool that enhances the efficiency and effectiveness of college event management.

The app also offers a centralized calendar that displays all upcoming events, making it easier for users to stay up-to-date on important dates. Additionally, users can receive push notifications to remind them of upcoming events or any changes to the event details.

One of the key benefits of the college KIET Event Management App is its ability to facilitate communication between event organizers and attendees. The app offers a messaging system that allows users to communicate with each other and share updates or important information about the event. This feature can also be useful for event organizers to send updates or reminders to attendees, such as changes in event schedule or location.

Another advantage of the college event management app is it will display all clubs available in college campus from different departments. This will become handy for users to access all clubs and how to reach them.

Overall, the KIET Event Management App is a user-friendly and efficient tool that can help streamline event planning and organize all clubs in efficient way and making it easier for students, faculty, and staff to stay organized, communicate effectively.

ACKNOWLEDGEMENT

Success in life is never attained single-handedly. My deepest gratitude goes to my projectsupervisor, **Mr. Shashank Bharadwaj ,** for his guidance, help and encouragement throughout my research work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Arun Kumar Tripathi**, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled withenjoyment and happiness.

SACHIN LAL
VIDUSHI SHUKLA

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Organizing and managing events in a college setting can be a daunting task, especially when there are multiple clubs and different departments are involved. To simplify this process, we are introducing a KIET Event Management App that includes both events management and clubs' details in one convenient platform.

The app is designed to cater to the needs of students, faculty, and staff involved in organizing and managing events in a college setting. It offers a range of features to streamline the event planning process, such as creating and managing events, facilitating communication between event organizers and attendees.

In addition, the app also includes a comprehensive directory of all college clubs and organizations, providing users with easy access to important information such as club descriptions, leadership details, eligibility criteria and contact person along with location of club. This feature can be particularly helpful for students who are looking to get involved in extracurricular activities on campus.

By combining both events management and club details in one platform, the college event management app offers a one-stop solution for all college event planning needs. It is user-friendly, customizable, and designed to enhance the efficiency and effectiveness of college event management.

.

1.1 PROJECT DESCRIPTION

The KIET Event Management App is a comprehensive solution designed to simplify and streamline the process of organizing and managing events in a college setting. It caters to the needs of students, faculty, and staff, providing a convenient platform for seamless event planning and club management.

The app offers a range of powerful features to facilitate efficient event management. Users can create and manage events with ease, including setting event details, scheduling, and tracking attendance. Event organizers can communicate directly with attendees through the app, sending updates, reminders, and important announcements. This promotes effective communication and ensures all participants are well-informed.

One of the key features of the app is its extensive directory of college clubs and organizations. Users can access a centralized database that provides detailed information about each club, including descriptions, leadership details, eligibility criteria, and contact information. This feature empowers students who wish to engage in extracurricular activities, enabling them to explore various clubs and make informed decisions about their involvement.

The KIET Event Management App offers a user-friendly interface and can be customized to meet the specific needs and branding of the college. It provides a seamless user experience, allowing users to navigate effortlessly through the app's various functionalities.

Overall, the KIET Event Management App aims to enhance the efficiency and effectiveness of college event planning. By combining events management and club details in one platform, it offers a convenient and centralized solution for all college event-related activities. The app empowers event organizers, club leaders, and participants, facilitating smooth communication, coordination, and engagement throughout the college community.

1.2 PROJECT SCOPE

The scope of the KIET Event Management App project includes the development of a mobile application that serves as a comprehensive platform for organizing and managing events in a college setting. The app aims to provide a user-friendly and efficient solution to streamline event planning and club management processes. The key components of the project scope are as follows:

1. Event Management Features:
    - User registration and login functionality for students, faculty, and staff.
    - Event creation, including event details (title, date, time, location), event description, and event category.
    - Event registration and ticketing system for attendees.
    - Real-time event updates, notifications, and reminders.
    - Event attendance tracking and management.

2. Club Directory Features:
    - A centralized database of college clubs and organizations.
    - Club descriptions, leadership details, eligibility criteria, and contact information.
    - Club search and filtering capabilities based on different criteria (e.g.,

club type, interests).
- Integration with maps to display club locations on campus.

3. Communication Features:
   - In-app messaging and notifications for event organizers to communicate with event attendees.
   - Announcement feature for sending important updates and reminders to registered attendees.
   - Ability for club leaders to send notifications and announcements to their club members.

4. Customization and Branding:
   - Customizable app interface and branding to align with the college's visual identity.
   - Options to add college logos, colors, and personalized app themes.

5. User Management:
   - User profiles for students, faculty, and staff.
   - User roles and permissions management for event organizers and club leaders.

6. Scalability and Performance:
   - The app should be designed to handle a large number of users, events, and clubs efficiently.
   - Consideration of scalability and performance optimization to ensure smooth user experience.

7. Integration:
   - Integration with existing college systems and databases, if applicable.
   - Integration with mapping services for displaying event and club locations.

8. Testing and Quality Assurance:
   - Comprehensive testing to ensure the app's functionality, performance, and security.
   - Identification and resolution of any bugs or issues.

9. Documentation and Training:
   - Preparation of user manuals and documentation for administrators, event organizers, and club leaders.

- Training sessions or resources to familiarize users with the app's features and functionalities.

The project scope includes the development of the mobile application, backend server components, and necessary integrations. It also covers the testing, documentation, and training aspects to ensure a successful implementation of the KIET Event Management App.

1.3 HARDWARE REQUIREMENTS

- **Processor:** Your computer should have a 64-bit processor, with Eight or more cores, to support the development of Flutter apps.
- **Memory:** You should have at least 16 GB of RAM to run Flutter and other development tools smoothly.
- **Disk Space**: You need to have a minimum of 8 GB of free disk space to install the Flutter SDK and related tools.
- **Operating System:** Flutter can be used on Windows (7 or later), macOS (Sierra or later), or Linux (64-bit).
- **Android Emulator:** To run and test your Flutter app on an Android emulator, you need to have an emulator installed. The minimum requirements for an emulator are 2 GB of RAM and 4 GB of disk space.
- **iOS Simulator:** To run and test your Flutter app on an iOS simulator, you need to have a Mac computer with Xcode 9.0 or later installed.

In addition to these hardware requirements, it is also recommended to have a high-resolution display (1920 x 1080 or higher) to ensure optimal usability and productivity.

1.4 SOFTWARE REQUIREMENTS

- **Flutter SDK**: You need to download and install the Flutter SDK on your computer. The SDK includes the Flutter framework, development tools, and a set of pre-built widgets.
- **Integrated Development Environment (IDE)**: You need to choose an IDE to write, edit, and debug your Flutter app code. Some popular IDEs for Flutter development include Android Studio, Visual Studio Code, and IntelliJ IDEA.
- **Dart SDK**: Flutter uses the Dart programming language, so you also need to install the Dart SDK. The Flutter SDK includes a compatible version of the Dart SDK, but you can also install it separately.
- **Android Studio**: If you plan to develop Android apps using Flutter, you need to install Android Studio and configure it for use with Flutter.

- **Xcode**: If you plan to develop iOS apps using Flutter, you need to install Xcode on a Mac computer and configure it for use with Flutter.
- **Git**: Git is a version control system that is commonly used for software development. You need to install Git on your computer to manage your Flutter app code.
- **Flutter plugins**: You may need to install additional plugins to extend the functionality of your Flutter app, such as for Firebase integration, maps, or analytics.

CHAPTER-2

LITERATURE REVIEW

The use of mobile applications for event management has become increasingly popular in recent years. Studies have shown that mobile apps can significantly improve the efficiency and effectiveness of event planning and management, particularly in a college setting.

One study by Li and colleagues (2018) investigated the effectiveness of a mobile app for managing college events. The app included features such as event creation and management, RSVP tracking, and communication tools. The results showed that the app was effective in streamlining the event planning process and improving communication between event organizers and attendees.

Another study by Huang and colleagues (2019) explored the use of mobile apps for managing college club activities. The app included features such as a club directory, event creation and management, and communication tools. The results showed that the app was effective in improving the efficiency of club management and increasing student engagement in club activities.

The use of mobile apps for event management has also been shown to have a positive impact on attendee engagement and satisfaction. A study by Bhatt and colleagues (2019) found that the use of a mobile app for a college conference increased attendee engagement and satisfaction with the event.

The combination of event management and club details in one platform, as offered by the college event management app, has not been extensively studied. However, it is reasonable to assume that this approach could be beneficial in improving communication and collaboration between clubs and event organizers, and in increasing student engagement in extracurricular activities.

In conclusion, the literature suggests that mobile apps can be effective in improving the efficiency and effectiveness of event management and club activities in a college setting. The college event management app, which combines both events management and club details in one platform, has the potential to offer a comprehensive and user-friendly solution for college event planning needs.

CHAPTER 3

FEASIBILITY STUDY

A feasibility study is a detailed analysis that considers all of the critical aspects of a proposed project in order to determine the likelihood of it succeeding.

Success in business may be defined primarily by return on return on investment, meaning that the project will generate enough profit to justify the investment. However, many other important factors may be identified on the plus or minus side, such as community reaction and environmental impact.

A feasibility study is an important step in any project, including an emotion detection project. It helps to determine the technical, economic, operational, and legal feasibility of the project. Here are some key aspects to consider in a feasibility study for an emotion detection project:

Based on the results of the feasibility study, the project team can make informed decisions about the viability and scope of the emotion detection project. If the feasibility study indicates that the project is viable and has potential benefits, the team can proceed with the project planning and implementation. If the study indicates that the project is not feasible or has significant risks and limitations, the team can consider alternative approaches or abandon the project altogether.

Before starting the project, feasibility study is carried out to measure the viable of the system. Feasibility is necessary to determine is creating a new or improved system is friendly with the cost, benefits, operation, technology and time. Following feasibility is given below:

Feasibility studies are important for a communications service provider to determine whether your broadband project will succeed or not. It should be the first action taken when wanting to begin a new project. It is one, if not the most important factor in determining whether the project can and should move forward. Also, if you are applying for broadband loans and grants, a feasibility study is normally required.

3.1 TECHNICAL FEASIBILITY

The technical feasibility of the KIET Event Management App project involves

assessing the availability of resources, technology requirements, compatibility, scalability, and security considerations. Here are the key factors to consider:

1. Resource Availability:
    - Evaluate the availability of skilled developers, designers, and testers for app development.
    - Assess the availability of infrastructure, servers, and hosting resources for backend components.
    - Consider the availability of devices for testing the app on various platforms (iOS, Android).
2. Technology Requirements:
    - Determine the programming languages, frameworks, and tools required for app development (e.g., Flutter, React Native, Android Studio, Xcode).
    - Evaluate the compatibility of the selected technologies with the targeted mobile platforms (iOS, Android).
    - Consider the integration capabilities of the chosen technologies with external systems or APIs.
3. Compatibility:
    - Ensure the app is compatible with a wide range of mobile devices and operating system versions.
    - Perform testing on different devices, screen sizes, and resolutions to ensure a consistent user experience.
    - Address platform-specific requirements and design guidelines (e.g., Apple App Store guidelines, Google Play Store policies).
4. Scalability and Performance:
    - Design the app architecture to handle a large number of users, events, and clubs.
    - Optimize database queries and backend processes for efficient performance.
    - Conduct load testing to simulate a high number of concurrent users and ensure the app's responsiveness.
5. Security:
    - Implement robust authentication and authorization mechanisms to protect user data and prevent unauthorized access.
    - Encrypt sensitive data such as user credentials and payment information.
    - Follow best practices for secure communication protocols (HTTPS) and data storage.
6. Integration:
    - Evaluate the integration requirements with existing college systems or databases for user authentication or data synchronization.

- Implement APIs or web services for seamless integration with mapping services or other external systems.
- Ensure compatibility with relevant APIs and adhere to their usage policies and limitations.

7. Testing and Quality Assurance:
   - Develop a comprehensive testing strategy that includes functional testing, usability testing, and compatibility testing across devices and operating systems.
   - Perform rigorous testing to identify and fix any bugs or issues before the app is deployed.
   - Conduct security testing and vulnerability assessments to protect against potential threats.
8. Documentation and Training:
   - Prepare technical documentation that outlines the app's architecture, features, and deployment instructions.
   - Provide user manuals and guides for administrators, event organizers, and club leaders to understand and use the app effectively.
   - Conduct training sessions or provide training resources to familiarize users with the app's functionalities.

Considering these technical feasibility factors will ensure the successful development, deployment, and performance of the KIET Event Management App, meeting the requirements outlined in the project scope.

3.2 ECONOMIC FEASIBILITY

Economic feasibility refers to the ability of a project or technology to generate economic value that exceeds its costs. In other words, it involves assessing whether a project is financially viable and can be expected to generate a positive return on investment. In the case of emotion detection technology, economic feasibility is an important consideration as it will impact the adoption and commercialization of the technology.

Here are some factors that contribute to economic feasibility in any project, including emotion detection:

- Cost-Benefit Analysis: Before investing in any project, a cost-benefit analysis is conducted to evaluate the expected costs and benefits of the project. The analysis should consider all relevant costs, including the cost of development, implementation, maintenance, and operation of the technology. The benefits should also be identified, such as increased productivity, reduced costs, or

improved quality of life.

- Market Demand: The economic feasibility of a project is dependent on market demand. The project should be able to satisfy a market need or gap, and there should be a large enough customer base to support the technology. For emotion detection technology, there is a growing demand in various industries, such as healthcare, marketing, and security, which indicates a positive market outlook.

- Competitive Landscape: The competition in the market can impact the economic feasibility of the technology. The technology should have a unique selling point that differentiates it from other solutions in the market. In addition, the technology should be able to compete on price, quality, and performance.

- Return on Investment (ROI): The ROI is an important measure of economic feasibility. The ROI should be higher than the cost of capital or the expected rate of return. A positive ROI indicates that the project is financially viable and generates economic value.

- Scalability: The ability to scale the technology is important for economic feasibility. The technology should be able to handle increasing demand without incurring significant additional costs. This will ensure that the technology can generate revenue as demand grows.

In the case of emotion detection technology, economic feasibility is influenced by several factors. These include:

- Adoption and Commercialization: The adoption and commercialization of emotion detection technology are critical for economic feasibility. The technology should be widely adopted and integrated into existing systems to generate revenue. This requires significant investment in marketing, sales, and distribution channels.

- Cost of Development and Implementation: The cost of developing and implementing emotion detection technology can be high. The cost includes the cost of hardware, software, and personnel. To be economically feasible, the technology should generate enough revenue to cover these costs.

- Value Proposition: The value proposition of emotion detection technology is

critical for economic feasibility. The technology should offer a unique selling point that differentiates it from other solutions in the market. This could be high accuracy, speed, or ease of use.

- Market Demand: The demand for emotion detection technology is growing in various industries, such as healthcare, marketing, and security. The ability to satisfy this demand is critical for economic feasibility.

- Regulatory Compliance: The regulation of emotion detection technology is an important consideration for economic feasibility. The technology should comply with applicable laws and regulations to avoid legal and financial penalties.

Overall, economic feasibility is a critical consideration for any project or technology, including emotion detection. A cost-benefit analysis, market demand, competitive landscape, ROI, and scalability are some of the key factors that contribute to economic feasibility. In addition, adoption and commercialization, cost of development and implementation, value proposition, market demand, and regulatory compliance are important factors that influence the economic feasibility of emotion detection technology.

3.3 OPERATIONAL FEASIBILITY

The operational feasibility of the KIET Event Management App project involves assessing its compatibility with existing operational processes, resources, and stakeholders. Consider the following factors when evaluating the operational feasibility:

1. Stakeholder Alignment:
    - Identify and engage key stakeholders, such as college administrators, event organizers, club leaders, and attendees.
    - Assess their willingness to adopt and embrace the new app for event management.
    - Ensure alignment between the app's features and functionalities and the needs of different user groups.
2. Process Integration:
    - Evaluate the compatibility of the app with existing event management and club management processes.
    - Identify areas where the app can integrate with and enhance current workflows.
    - Consider any necessary changes or adaptations to ensure a smooth

transition to the new system.

3. User Experience:
    - Design the app with a user-centric approach, focusing on ease of use, intuitiveness, and efficiency.
    - Conduct usability testing to gather feedback and refine the user interface and interactions.
    - Ensure that the app's features and functionalities are easily accessible and understandable for all user groups.
4. Training and Support:
    - Develop training materials and resources to educate stakeholders on app usage and functionality.
    - Provide adequate support channels, such as help desks or online forums, for addressing user queries or issues.
    - Offer ongoing support and updates to ensure users remain engaged and receive timely assistance.
5. Organizational Readiness:
    - Assess the readiness of the college and its departments to adopt and implement the event management app.
    - Identify any organizational barriers or resistance to change and develop strategies to address them.
    - Collaborate with relevant departments to ensure smooth integration of the app into existing systems and processes.
6. Data Management and Privacy:
    - Establish data management protocols to protect user information and ensure compliance with privacy regulations.
    - Develop data backup and recovery mechanisms to prevent data loss or system failures.
    - Communicate and enforce data privacy policies and obtain necessary consent from users.
7. Scalability and Growth:
    - Design the app to accommodate potential growth in the number of users, events, and clubs.
    - Ensure that the app's infrastructure and architecture can handle increasing demands and user activity.
    - Plan for scalability by utilizing cloud-based services or scalable infrastructure solutions.
8. Performance and Reliability:
    - Conduct thorough testing to ensure the app's performance, responsiveness, and reliability.

- Monitor system performance and address any performance issues promptly.
- Establish mechanisms for collecting user feedback and continuously improving the app's performance.

By assessing the operational feasibility, you can determine the app's compatibility with existing processes, ensure stakeholder buy-in, and plan for a successful implementation. This evaluation will help in identifying potential challenges and developing strategies to overcome them, ensuring smooth operations of the KIET Event Management App.

3.4 BEHAVIORAL FEASIBILITY

Behavioral feasibility assesses the acceptance and adoption of the KIET Event Management App by its intended users, including students, faculty, staff, event organizers, club leaders, and attendees. Consider the following factors when evaluating the behavioral feasibility:

1. User Acceptance:
    - Conduct surveys, interviews, or focus groups to gather feedback from potential app users.
    - Assess their attitudes, preferences, and expectations regarding event management and club activities.
    - Identify any concerns, barriers, or resistance to using the app and address them proactively.
2. User Experience and Engagement:
    - Design the app with a user-friendly interface and intuitive navigation.
    - Focus on delivering a positive user experience, making tasks easy to accomplish and information readily accessible.
    - Incorporate interactive and engaging features to promote user involvement and participation.
3. Communication and Collaboration:
    - Enable effective communication between event organizers, club leaders, and attendees through the app.
    - Provide collaboration tools, such as messaging, notifications, and announcements, to facilitate seamless interaction.
    - Foster a sense of community and engagement by encouraging discussions and feedback within the app.
4. User Training and Support:
    - Offer comprehensive training and onboarding resources to educate users about the app's features and functionality.
    - Provide user guides, tutorials, and help documentation to assist users

in navigating the app.
    - Establish a support system to address user inquiries, troubleshoot issues, and provide timely assistance.
5. Change Management:
    - Develop a change management strategy to promote user acceptance and smooth adoption of the app.
    - Communicate the benefits of the app, emphasizing how it enhances event management and club activities.
    - Engage key stakeholders in the development process and involve them in decision-making to foster ownership.
6. User Incentives and Rewards:
    - Consider implementing incentive programs or rewards to encourage user engagement and participation.
    - Recognize and appreciate active users, event organizers, and club leaders through gamification or achievement systems.
    - Foster a sense of community and competition by showcasing achievements and milestones within the app.
7. Continuous Improvement:
    - Establish mechanisms to collect user feedback and suggestions for app enhancements.
    - Regularly update the app based on user feedback, addressing usability issues and incorporating new features.
    - Demonstrate responsiveness to user needs, ensuring that their input is valued and considered in future app iterations.

By considering the behavioral feasibility, you can anticipate user acceptance and address any potential challenges related to user attitudes, preferences, and adoption. This evaluation will help in designing an app that aligns with user expectations, encourages engagement, and fosters a positive user experience within the KIET Event Management App.

CHAPTER 4

DATABASE DESIGN

4.1 FLOW CHART

Flowchart is a graphical representation of an algorithm. Programmers often use it as a program- planning tool to solve a problem. It makes use of symbols which are connected among them to indicate the flow of information and processing. The process of drawing a flowchart for an algorithm is known as “flow charting”.

Basic Symbols used in Flowchart Designs-

1. Start:
    - The flowchart begins with the start symbol.
2. User Registration and Login:
    - Users are prompted to register or log in to the app.
    - If registered, users can proceed to log in.
    - If not registered, they can complete the registration process.
3. Home Screen:
    - After successful login, users are directed to the home screen.
    - The home screen provides an overview of upcoming events, club updates, and navigation options.
4. Event Management:
    - Users can navigate to the event management section.
    - They can create new events by providing event details such as title, date, time, location, and description.
    - Event organizers can manage existing events, including editing event details and tracking RSVPs.
5. Club Directory:
    - Users can access the club directory section.
    - The club directory provides a searchable list of college clubs and organizations.
    - Users can view club descriptions, leadership details, eligibility criteria, and contact information.
    - Integration with maps can display club locations on the campus.
    - Communication and Collaboration:

- The app facilitates communication and collaboration between event organizers, club leaders, and attendees.
- In-app messaging and notifications allow event organizers to communicate with attendees and club leaders.
- Announcements and notifications can be sent to registered attendees or club members.

6. User Profile and Settings:
    - Users can access and manage their profiles and settings.
    - They can update personal information, notification preferences, and privacy settings.
7. Continuous Improvement:
    - The app incorporates mechanisms to collect user feedback and suggestions.
    - Feedback is used to enhance the app's features, usability, and performance through regular updates.
8. End:
    - The flowchart ends with the end symbol.

4.2 USE CASE DIAGRAM

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use aset of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

Scenarios in which your system or application interacts with people, organizations, or externalsystems

A use case diagram is a visual representation of the interactions between actors and a system in various scenarios. A use case diagram is a valuable tool in project reports for illustrating the functionality of a system and its interactions with actors. It helps to identify the different use cases of the system, the actors involved, and their relationships. In a project report, a use case diagram can be used to provide a visual representation of the requirements and features of the system. The diagram can be accompanied by a brief description of each use case, including its purpose and the steps involved. Moreover, the use case diagram can help stakeholders to understand the system's functionalities and requirements better. It can also serve as a basis for testing and validation of the system. By visualizing the system's interactions, stakeholders can

identify potential issues and requirements that may need to be addressed. Overall, including a use case diagram in a project report is a useful way to illustrate the system's functionality and requirements to stakeholders. It helps to ensure that everyone involved in the project has a clear understanding of the system and its interactions with the actors.

A use case diagram is a type of UML diagram that is used to illustrate the different ways that users interact with a system. It shows the various use cases, actors, and their relationships. In a project report for an emotion detection system, a use case diagram can be used to show the different types of users that interact with the system and the tasks that they perform.

For example, a use case diagram for an emotion detection system might include actors such as news readers, news curators, and administrators. News readers might use the system to check the credibility of a news article, while news curators might use the system to review and evaluate news content. Administrators might use the system to manage user accounts and perform maintenance tasks.

The use cases themselves might include tasks such as submitting news content, reviewing news content, and checking the credibility of news content. These tasks would be represented by rectangular boxes on the diagram. Arrows would be used to show the relationships between the actors and the use cases they perform.
Goals that your system or application helps those entities (known as actors) achieve

Use case Diagram Components-

To answer the question, "What is a use case diagram?" you need to first understand its buildingblocks. Common components include:

- **Actors:** The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects thatproduce or consume data.
- **System:** A specific sequence of actions and interactions between actors and the system. Asystem may also be referred to as a scenario.
- **Goals:** The result of most use cases. A successful diagram should describe the activitiesand variants used to reach the goal.

Use case diagram symbols and notation-

The notation for a use case diagram is straightforward and doesn't involve as many types of symbols as other UML diagrams.

- **Use cases:** Horizontally shaped ovals that represent the different uses that a user might have.
- **Actors:** Stick figures that represent the people employing the use cases.
- **Associations:** A line between actors and use cases. In complex diagrams, it is important toknow which actors are associated with which use cases.
- **System boundary boxes:** A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.
- **Packages:** A UML shape that allows you to put different elements into groups. Just as withcomponent diagrams, these groupings are represented as file folders.

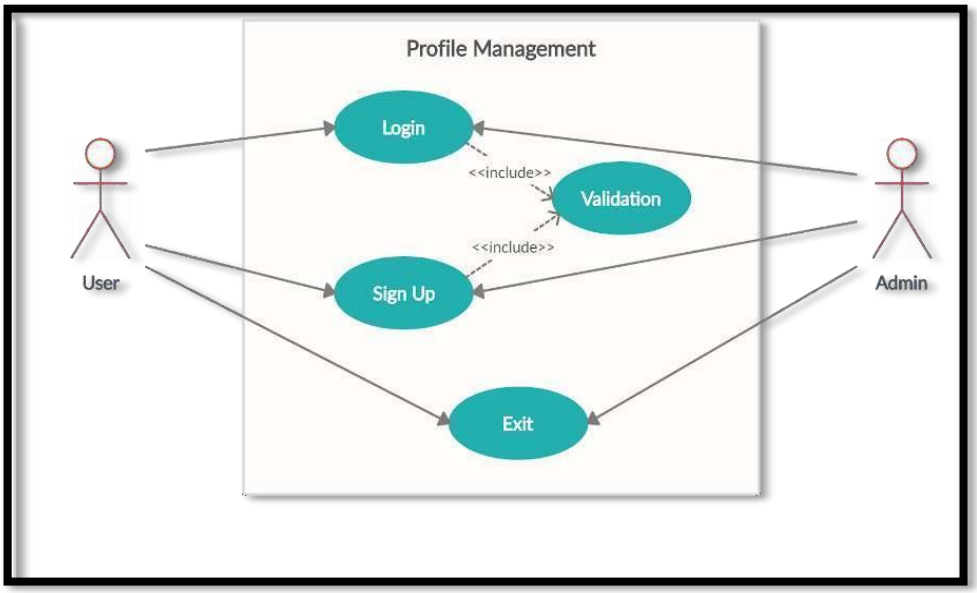

Figure 4.1. Profile Management Use Case

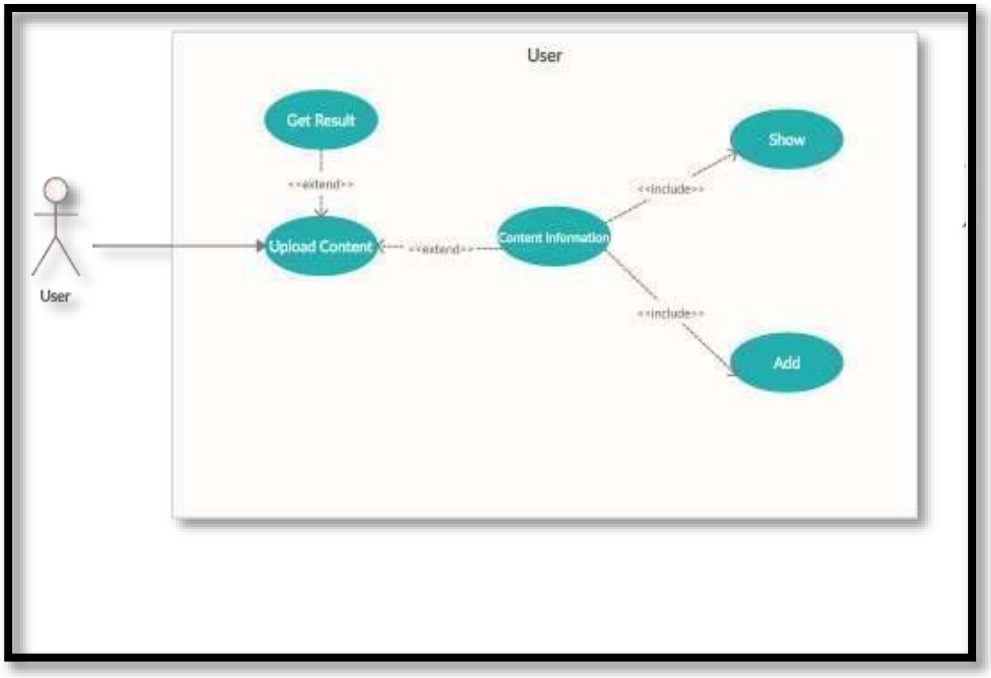

Figure 4.2. User Use Case

CHAPTER 5

FORM DESIGN

1. **User Registration Form:**
   - Full Name (input field)
   - Email Address (input field)
   - Password (input field)
   - Confirm Password (input field)
   - Role/Type (dropdown/select field with options like "Student," "Faculty," "Staff")
2. **User Login Form:**
   - Email Address (input field)
   - Password (input field)
   - Remember Me (checkbox)
3. **Event Creation Form:**
   - Event Title (input field)
   - Event Date (date picker/calendar)
   - Event Time (time picker)
   - Event Location (input field)
   - Event Description (textarea field)
   - Event Category (dropdown/select field with options like "Academic," "Sports," "Cultural," etc.)
4. **Event RSVP Form:**
   - User Name (display field)
   - Email Address (display field)
   - RSVP Status (radio buttons or dropdown/select field with options like "Attending," "Not Attending," "Maybe Attending")
   - Additional Comments (textarea field)
5. **Club Search Form:**
   - Club Name (input field)
   - Club Type (dropdown/select field with options like "Academic," "Sports," "Cultural," etc.)
6. **Club Details Form:**
   - Club Name (display field)

- Club Type (display field)
- Club Description (text area field)
- Leadership Details (display field)
- Eligibility Criteria (display field)
- Contact Information (display field)

7. User Profile Form:

- Full Name (input field)
- Email Address (input field)
- Password (input field)
- Confirm Password (input field)
- Role/Type (display field)
- Profile Picture (file upload field)

8. App Settings Form：

- Notification Preferences (checkboxes or toggles for different notification types)
- Privacy Settings (checkboxes or toggles for privacy options)

CHAPTER 6

CODING

6.1 MAIN.DART

```
import 'package:flash_chat/screens/login_screen.dart';
import 'package:flash_chat/screens/registration_screen.dart';
import 'package:flash_chat/screens/welcome_screen.dart';
import 'package:flutter/material.dart';

import 'screens/welcomescreen.dart';

void main() async {
  runApp(const KietEvent());
}

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

  @override
  State<KietEvent> createState() => _KietEventState();
}

class _KietEventState extends State<KietEvent> {
  var islogin = false;

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: WelcomeScreen(),
    );
  }
}
```

6.2 WELCOMESCREEN.DART

```
import 'dart:ffi';

import 'package:flash_chat/screens/signup.dart';
import 'package:flash_chat/screens/user_main.dart';
import 'package:flutter/material.dart';

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color.fromARGB(255, 2, 58, 104),
      body: Column(
        children: [
          Container(
            width: double.maxFinite,
            height: MediaQuery.of(context).size.height * 0.9,
            decoration: BoxDecoration(
                color: Colors.white, borderRadius: BorderRadius.circular(40)),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                SizedBox(
                  height: 100,
                ),
                Center(
                  child: Text.rich(TextSpan(children: [
                    TextSpan(
                        text: "KIET ",
                        style: TextStyle(
                            color: Colors.blueAccent,
                            fontSize: 40,
                            fontWeight: FontWeight.bold)),
                    TextSpan(
                        text: "EVENT",
                        style: TextStyle(
```

```
          color: Colors.black,
          fontWeight: FontWeight.bold,
          fontSize: 40,
        ))
  ])),
),
SizedBox(
  height: 50,
),
Padding(
  padding: const EdgeInsets.only(left: 20),
  child: Text(
    "Login",
    style: TextStyle(
        letterSpacing: 2,
        color: Colors.blueAccent,
        fontSize: 40,
        fontWeight: FontWeight.w600),
  ),
),
Padding(
  padding: const EdgeInsets.only(left: 20),
  child: Text(
    "To Continue your account!",
    style: TextStyle(color: Colors.grey, fontSize: 20),
  ),
),
SizedBox(
  height: 50,
),
Padding(
  padding: const EdgeInsets.only(left: 20),
  child: Text(
    "Email ID / Library ID",
    style: TextStyle(
        color: Colors.black,
        fontWeight: FontWeight.bold,
        fontSize: 18),
  ),
),
Padding(
```

```
  padding: const EdgeInsets.only(left: 20, top: 10),
  child: Row(
    children: [Icon(Icons.person)],
  ),
),
SizedBox(
  height: 30,
),
Padding(
  padding: const EdgeInsets.only(left: 20),
  child: Text(
    "Password",
    style: TextStyle(
        color: Colors.black,
        fontWeight: FontWeight.bold,
        fontSize: 18),
  ),
),
Padding(
  padding: const EdgeInsets.only(left: 20, top: 10),
  child: Row(
    children: [Icon(Icons.person)],
  ),
),
SizedBox(
  height: 25,
),
Padding(
  padding: const EdgeInsets.only(right: 25),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.end,
    children: [
      Text(
        "Forgot Password ?",
        style: TextStyle(
            color: Colors.blueAccent,
            fontWeight: FontWeight.w500),
      ),
    ],
  ),
),
```

```
      SizedBox(
        height: 40,
      ),
      Center(
        child: GestureDetector(
          onTap: () {
            Navigator.push(context,
                MaterialPageRoute(builder: (context) => UserMain()));
          },
          child: Container(
            decoration: BoxDecoration(
                color: Color.fromARGB(255, 2, 58, 104),
                borderRadius: BorderRadius.circular(30)),
            height: MediaQuery.of(context).size.height * 0.070,
            width: MediaQuery.of(context).size.height * 0.35,
            child: Center(
              child: Text(
                "Login",
                style: TextStyle(
                    color: Colors.white,
                    fontWeight: FontWeight.w500,
                    fontSize: 30),
              ),
            ),
          ),
        ),
      ),
      SizedBox(
        height: 20,
      ),
      Center(child: Text("new user"))
    ],
  ),
),
Spacer(),
Center(
  child: GestureDetector(
    onTap: () {
      Navigator.push(context,
          MaterialPageRoute(builder: (context) => NewRegistration()));
    },
```

```
            child: Text.rich(TextSpan(children: [
              TextSpan(
                 text: "Didn't have an account ? ",
                 style: TextStyle(
                    color: Colors.white,
                    fontSize: 20,
                    fontWeight: FontWeight.w400)),
              TextSpan(
                 text: "Signup",
                 style: TextStyle(
                   color: Color.fromARGB(255, 161, 195, 255),
                   fontWeight: FontWeight.w400,
                   fontSize: 20,
                 ))
            ])),
          ),
        ),
        SizedBox(
          height: 20,
        )
      ],
    ),
  );
 }
}
```

6.3 USERMAIN.DART

```
import 'dart:math';
import 'package:carousel_slider/carousel_options.dart';
import 'package:carousel_slider/carousel_slider.dart';
import 'package:flash_chat/screens/screenclub.dart';
import 'package:flash_chat/screens/ProfileScreen.dart';
import 'package:flash_chat/screens/ActivityScreen.dart';
import 'package:flash_chat/screens/ChangePassword.dart';
import 'package:flash_chat/screens/Dashboard.dart';
import 'package:flash_chat/screens/EventScreen.dart';
```

```
import 'package:flash_chat/widgets/Profile.dart';
import 'package:flash_chat/screens/login_screen.dart';
import 'package:flash_chat/utils/constant.dart';
import 'package:flutter/material.dart';

import '../widgets/ContainerBox.dart';
import 'welcome_screen.dart';
import 'clubscreens.dart';

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

  @override
  State<UserMain> createState() => _UserMainState();
}

class _UserMainState extends State<UserMain> {
  double value = 0;
  double val = 0;

  @override
  Widget build(BuildContext context) {
    Size size = MediaQuery.of(context).size;
    return Scaffold(
      body: Stack(
        children: [
          Container(
            height: size.height,
            decoration: BoxDecoration(
                gradient: LinearGradient(colors: [
              Color.fromRGBO(23, 33, 42, 1),
              Color.fromARGB(255, 22, 141, 239)
            ], begin: Alignment.topCenter, end: Alignment.bottomCenter)),
          ),
          SafeArea(
              child: Container(
                  width: 200,
                  height: 500,
                  padding: const EdgeInsets.all(8),
                  child: GestureDetector(
                    onTap: () {
```

```
  setState(() {
    value == 0 ? value = 1 : value = 0;
  });
},
child: Column(
  children: [
    DrawerHeader(
        child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        CircleAvatar(
          backgroundImage: AssetImage(
            'Assets/images/profile.png',
          ),
          radius: 50,
        ),
        SizedBox(
          height: ksmallSpacing,
        ),
        Text(
          'USERNAME ',
          style: TextStyle(
              color: Colors.white,
              fontSize: 18,
              fontWeight: FontWeight.w600),
        ),
      ],
    )),
    Expanded(
        child: ListView(
      scrollDirection: Axis.vertical,
      children: [
        GestureDetector(
          onTap: () {
            Navigator.push(context, MaterialPageRoute(
              builder: (context) {
                return ProfileScreen();
              },
            ));
          },
          child: const ListTile(
```

```
    leading: Icon(
      Icons.person,
      color: Colors.white,
    ),
    title: Text('My Profile',
        style: TextStyle(color: Colors.white)),
  ),
),
GestureDetector(
  onTap: () {
    Navigator.push(context, MaterialPageRoute(
      builder: (context) {
        return ClubScreen();
      },
    ));
  },
  child: const ListTile(
    leading: Icon(
      Icons.person_3_sharp,
      color: Colors.white,
    ),
    title: Text('Clubs',
        style: TextStyle(color: Colors.white)),
  ),
),
const ListTile(
  leading: Icon(
    Icons.local_activity_outlined,
    color: Colors.white,
  ),
  title: Text('Activity',
      style: TextStyle(color: Colors.white)),
),
GestureDetector(
  onTap: () {
    Navigator.push(context, MaterialPageRoute(
      builder: (context) {
        return EventScreen();
      },
    ));
  },
```

```
                child: const ListTile(
                  leading: Icon(
                    Icons.event_sharp,
                    color: Colors.white,
                  ),
                  title: Text('Events',
                      style: TextStyle(color: Colors.white)),
                ),
              ),
              const ListTile(
                leading: Icon(
                  Icons.settings,
                  color: Colors.white,
                ),
                title: Text('settings',
                    style: TextStyle(color: Colors.white)),
              ),
              ListTile(
                leading: const Icon(
                  Icons.exit_to_app_outlined,
                  color: Colors.white,
                ),
                title: const Text('Log out',
                    style: TextStyle(color: Colors.white)),
              ),
            ],
          ))
        ],
      ),
    ))),
TweenAnimationBuilder(
  tween: Tween(begin: 0, end: value),
  curve: Curves.easeIn,
  duration: const Duration(milliseconds: 500),
  builder: ((_, val, __) {
    return Transform(
      alignment: Alignment.center,
      transform: Matrix4.identity()
        ..setEntry(3, 2, 0.001)
        ..setEntry(0, 3, value * 200)
        ..rotateY((pi / 6) * val),
```

```
child: Scaffold(
   appBar: AppBar(
    leading: GestureDetector(
     onTap: () {
      setState(() {
       value == 0 ? value = 1 : value = 0;
      });
     },
     child: Icon(Icons.menu),
    ),
    title: Text('Welcome Student !'),
    backgroundColor: Color.fromARGB(255, 8, 117, 185),
    centerTitle: true,
    elevation: 0,
   ),
   body: SingleChildScrollView(
    scrollDirection: Axis.vertical,
    child: Padding(
     padding: const EdgeInsets.all(ksmallSpacing),
     child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
       const Text(
        'Activity',
        style: TextStyle(
          color: Colors.blue,
          fontSize: 25,
          fontWeight: FontWeight.w400),
       ),
       const SizedBox(
        height: ksmallSpacing,
       ),
       CantainerBox(
        image: 'Assets/images/activity.jpeg',
        onpress: () {
         Navigator.pushReplacement(context,
           MaterialPageRoute(
          builder: (context) {
           return ActivityScreen();
          },
         ));
```

```
  },
),
const SizedBox(height: knormalSpacing),
const Text(
  'Club',
  style: TextStyle(
      color: Colors.blueAccent,
      fontSize: 25,
      fontWeight: FontWeight.w400),
),
const SizedBox(
  height: ksmallSpacing,
),
CantainerBox(
  image: 'Assets/images/club.jpeg',
  onpress: () {
    Navigator.push(context, MaterialPageRoute(
      builder: (context) {
        return ClubScreen();
      },
    ));
  },
),
SizedBox(height: knormalSpacing),
Text(
  'Event',
  style: TextStyle(
      color: Colors.blue,
      fontSize: 25,
      fontWeight: FontWeight.w400),
),
SizedBox(
  height: ksmallSpacing,
),
CantainerBox(
  image: 'Assets/images/event.jpeg',
  onpress: () {
    Navigator.push(context, MaterialPageRoute(
      builder: (context) {
        return EventScreen();
      },
```

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

6.4 EVENTSCREEN.DART

```
import 'package:flash_chat/utils/constant.dart';
import 'package:flash_chat/widgets/eventheading.dart';
import 'package:flutter/material.dart';

import '../widgets/PhotoContainer.dart';
import '../widgets/photoCollage.dart';

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

 @override
 State<EventScreen> createState() => _EventScreenState();
}

class _EventScreenState extends State<EventScreen> {
 @override
 Widget build(BuildContext context) {
  Size size = MediaQuery.of(context).size;
  return Scaffold(
   appBar: AppBar(
    elevation: 0,
```

```
  title: const Text(
    'Events',
    style: TextStyle(color: Colors.white),
  ),
  backgroundColor: Colors.blueAccent,
  centerTitle: true,
),
body: Padding(
  padding: EdgeInsets.symmetric(horizontal: knormalSpacing + 5),
  child: SingleChildScrollView(
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: const [
        Eventheading(),
        Padding(
          padding: EdgeInsets.symmetric(vertical: ksmallSpacing),
          child: PhotoCollage(
            image1: 'Assets/images/even1.jpg',
            image2: 'Assets/images/even2.jpg',
            image3: 'Assets/images/event3.jpg',
          ),
        ),
        Eventheading(),
        PhotoContainer(
          image: 'Assets/images/even4.jpg',
        ),
        PhotoContainer(
          image: 'Assets/images/even5.jpg',
        ),
        Eventheading(),
        Padding(
          padding: EdgeInsets.symmetric(vertical: ksmallSpacing),
          child: PhotoCollage(
            image1: 'Assets/images/even1.jpg',
            image2: 'Assets/images/even2.jpg',
            image3: 'Assets/images/event3.jpg',
          ),
        ),
        Eventheading(),
        PhotoContainer(
          image: 'Assets/images/even4.jpg',
```

```
            ),
            Eventheading(),
            PhotoContainer(
              image: 'Assets/images/even5.jpg',
            ),
            Eventheading(),
            Padding(
              padding: EdgeInsets.symmetric(vertical: ksmallSpacing),
              child: PhotoCollage(
                image1: 'Assets/images/even6.jpg',
                image2: 'Assets/images/even2.jpg',
                image3: 'Assets/images/even1.jpg',
              ),
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

6.5 ACTIVITYSCREEN.DART

```
import 'package:flash_chat/utils/constant.dart';
import 'package:flutter/material.dart';

import '../widgets/PhotoContainer.dart';
import '../widgets/activityheading.dart';
import '../widgets/photoCollage.dart';

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

 @override
 State<ActivityScreen> createState() => _ActivityScreenState();
}
```

```
class _ActivityScreenState extends State<ActivityScreen> {
  @override
  Widget build(BuildContext context) {
    Size size = MediaQuery.of(context).size;
    return Scaffold(
      appBar: AppBar(
        elevation: 0,
        title: const Text(
          'Activity',
          style: TextStyle(color: Colors.white),
        ),
        backgroundColor: Colors.blueAccent,
        centerTitle: true,
        automaticallyImplyLeading: false,
      ),
      body: Padding(
        padding: EdgeInsets.symmetric(horizontal: knormalSpacing + 5),
        child: SingleChildScrollView(
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              activityheading(),
              Padding(
                padding: EdgeInsets.symmetric(vertical: ksmallSpacing),
                child: PhotoCollage(
                  image1: 'Assets/images/act1.jpg',
                  image2: 'Assets/images/act2.jpg',
                  image3: 'Assets/images/act3.jpg',
                ),
              ),
              activityheading(),
              PhotoContainer(
                image: 'Assets/images/act4.jpg',
              ),
              activityheading(),
              PhotoContainer(
                image: 'Assets/images/act5.jpg',
              ),
              activityheading(),
              Padding(
```

```
            padding: EdgeInsets.symmetric(vertical: ksmallSpacing),
            child: PhotoCollage(
              image1: 'Assets/images/act6.jpg',
              image2: 'Assets/images/act7.jpg',
              image3: 'Assets/images/act8.jpg',
            ),
          ),
          activityheading(),
          PhotoContainer(
            image: 'Assets/images/act9.jpg',
          ),
          activityheading(),
          PhotoContainer(
            image: 'Assets/images/act5.jpg',
          ),
          activityheading(),
          Padding(
            padding: EdgeInsets.symmetric(vertical: ksmallSpacing),
            child: PhotoCollage(
              image1: 'Assets/images/act9.jpg',
              image2: 'Assets/images/act10.jpg',
              image3: 'Assets/images/act11.jpg',
            ),
          ),
        ],
      ),
    ),
  ),
 );
 }
}
```

6.6 CLUBSCREEN.DART

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

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

  @override
  State<ClubScreen> createState() => _ClubScreenState();
}

class _ClubScreenState extends State<ClubScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SafeArea(
        child: SingleChildScrollView(
          scrollDirection: Axis.vertical,
          child: Column(
            children: [
              Container(
                padding: EdgeInsets.all(10),
                height: 90,
                width: double.maxFinite,
                margin: EdgeInsets.only(bottom: 10),
                decoration: BoxDecoration(
                    color: Color.fromARGB(255, 7, 24, 54),
                    borderRadius: BorderRadius.only(
                        bottomRight: Radius.circular(20),
                        bottomLeft: Radius.circular(20))),
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
                  children: [
                    Row(
                      mainAxisAlignment: MainAxisAlignment.end,
                      children: [
                        Icon(Icons.notifications,
                            color: Color.fromARGB(255, 204, 232, 246)),
                        SizedBox(
                          width: 25,
                        ),
                        Icon(
                          Icons.search,
                          color: Color.fromARGB(255, 204, 232, 246),
```

```
          )
        ],
      ),
      Text(
        "KIET-CLUB",
        style: TextStyle(
            fontSize: 28,
            color: Color.fromARGB(255, 204, 232, 246),
            fontWeight: FontWeight.w500),
      )
    ],
  ),
),
ContainerBox(
  club: 'Innogeeks',
  description: 'Innogeeks is an appreciable initiative.',
  image: 'Assets/images/club1.jpg',
  location: 'E-Block – IT Department, E-311 & E-401',
  onpress: () {
    Navigator.push(context, MaterialPageRoute(
      builder: (context) {
        return ClubContainerScreen();
      },
    ));
  },
),
ContainerBox(
  club: 'Innogeeks',
  description: 'Innogeeks is an appreciable initiative.',
  image: 'Assets/images/club2.jpg',
  location: 'E-Block – IT Department, E-311 & E-401',
  onpress: () {
    Navigator.push(context, MaterialPageRoute(
      builder: (context) {
        return ClubContainerScreen();
      },
    ));
  },
),
ContainerBox(
  club: 'Innogeeks',
```

```
  description: 'Innogeeks is an appreciable initiative.',
  image: 'Assets/images/club3.jpg',
  location: 'E-Block – IT Department, E-311 & E-401',
  onpress: () {
    Navigator.push(context, MaterialPageRoute(
      builder: (context) {
        return ClubContainerScreen();
      },
    ));
  },
),
ContainerBox(
  club: 'Innogeeks',
  description: 'Innogeeks is an appreciable initiative.',
  image: 'Assets/images/club4.jpg',
  location: 'E-Block – IT Department, E-311 & E-401',
  onpress: () {
    Navigator.push(context, MaterialPageRoute(
      builder: (context) {
        return ClubContainerScreen();
      },
    ));
  },
),
ContainerBox(
  club: 'Innogeeks',
  description: 'Innogeeks is an appreciable initiative.',
  image: 'Assets/images/club5.jpg',
  location: 'E-Block – IT Department, E-311 & E-401',
  onpress: () {
    Navigator.push(context, MaterialPageRoute(
      builder: (context) {
        return ClubContainerScreen();
      },
    ));
  },
),
ContainerBox(
  club: 'Innogeeks',
  description: 'Innogeeks is an appreciable initiative.',
  image: 'Assets/images/club5.jpg',
```

```
  location: 'E-Block – IT Department, E-311 & E-401',
  onpress: () {
    Navigator.push(context, MaterialPageRoute(
      builder: (context) {
        return ClubContainerScreen();
      },
    ));
  },
),
ContainerBox(
  club: 'Innogeeks',
  description: 'Innogeeks is an appreciable initiative.',
  image: 'Assets/images/club6.jpg',
  location: 'E-Block – IT Department, E-311 & E-401',
  onpress: () {
    Navigator.push(context, MaterialPageRoute(
      builder: (context) {
        return ClubContainerScreen();
      },
    ));
  },
),
ContainerBox(
  club: 'Innogeeks',
  description: 'Innogeeks is an appreciable initiative.',
  image: 'Assets/images/club7.jpg',
  location: 'E-Block – IT Department, E-311 & E-401',
  onpress: () {
    Navigator.push(context, MaterialPageRoute(
      builder: (context) {
        return const ClubContainerScreen();
      },
    ));
  },
),
ContainerBox(
  club: 'Innogeeks',
  description: 'Innogeeks is an appreciable initiative.',
  image: 'Assets/images/club8.jpg',
  location: 'E-Block – IT Department, E-311 & E-401',
  onpress: () {
```

```
                Navigator.push(context, MaterialPageRoute(
                  builder: (context) {
                    return const ClubContainerScreen();
                  },
                ));
              },
            ),
            ContainerBox(
              club: 'Innogeeks',
              description: 'Innogeeks is an appreciable initiative.',
              image: 'Assets/images/club9.jpg',
              location: 'E-Block – IT Department, E-311 & E-401',
              onpress: () {
                Navigator.push(context, MaterialPageRoute(
                  builder: (context) {
                    return ClubContainerScreen();
                  },
                ));
              },
            ),
            ContainerBox(
              club: 'Innogeeks',
              description: 'Innogeeks is an appreciable initiative.',
              image: 'Assets/images/club10.jpg',
              location: 'E-Block – IT Department, E-311 & E-401',
              onpress: () {
                Navigator.push(context, MaterialPageRoute(
                  builder: (context) {
                    return ClubContainerScreen();
                  },
                ));
              },
            )
          ],
        ),
      ),
    ),
  );
 }
}
```

```
class ContainerBox extends StatelessWidget {
  ContainerBox(
      {super.key,
      this.club,
      this.description,
      this.image,
      this.location,
      this.onpress});
  final String? image;
  final String? club;
  final String? description;
  final String? location;
  final VoidCallback? onpress;

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onpress,
      child: Container(
        margin: EdgeInsets.only(bottom: 10, left: 10, right: 10),
        padding: EdgeInsets.all(6),
        decoration: BoxDecoration(
            borderRadius: BorderRadius.circular(12),
            color: Colors.grey.withOpacity(0.2)),
        child: Row(
          children: [
            Container(
              height: 150,
              width: 130,
              decoration: BoxDecoration(
                  borderRadius: BorderRadius.circular(12),
                  image: DecorationImage(
                      image: AssetImage(image!), fit: BoxFit.fill)),
            ),
            SizedBox(
              width: 20,
            ),
            Expanded(
                child: Column(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              crossAxisAlignment: CrossAxisAlignment.start,
```

```
            children: [
              Text(
                club!,
                style: TextStyle(
                    fontSize: 22,
                    fontWeight: FontWeight.bold,
                    color: Color.fromARGB(255, 19, 14, 47)),
              ),
              Padding(
                padding: EdgeInsets.symmetric(vertical: 6),
                child: Text(
                  description!,
                  style: TextStyle(
                      fontSize: 16,
                      fontWeight: FontWeight.w500,
                      color: Color.fromARGB(255, 19, 14, 47)),
                ),
              ),
              Text(
                location!,
                style: TextStyle(
                    fontSize: 12, color: Color.fromARGB(255, 93, 91, 91)),
              )
            ],
          ))
        ],
      ),
    ),
  );
 }
}

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

 @override
 State<ClubContainerScreen> createState() => _ClubContainerScreenState();
}

class _ClubContainerScreenState extends State<ClubContainerScreen> {
 @override
```

```
Widget build(BuildContext context) {
  return Scaffold(
     body: Container(
    margin: EdgeInsets.all(8),
    decoration: BoxDecoration(
       borderRadius: BorderRadius.circular(12),
       color: Colors.grey.withOpacity(0.2)),
    child: Column(
     crossAxisAlignment: CrossAxisAlignment.start,
     children: [
      Container(
       margin: EdgeInsets.all(8),
       height: 180,
       decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(12),
          image: DecorationImage(
             image: AssetImage('Assets/images/even1.jpg'),
             fit: BoxFit.fill)),
      ),
      Container(
       padding: EdgeInsets.all(12),
       margin: EdgeInsets.only(left: 10, right: 10, bottom: 10),
       width: double.maxFinite,
       decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(12), color: Colors.white),
       child: Column(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
         Text(
          'Innogeeks',
          style: TextStyle(
             fontSize: 22,
             fontWeight: FontWeight.bold,
             color: Color.fromARGB(255, 19, 14, 47)),
         ),
         Padding(
          padding: EdgeInsets.symmetric(vertical: 6),
          child: Text(
           'Innogeeks is an appreciable initiative.',
           style: TextStyle(
```

```
            fontSize: 16,
            fontWeight: FontWeight.w500,
            color: Color.fromARGB(255, 19, 14, 47)),
        ),
      ),
      Text(
        'E-Block – IT Department, E-311 & E-401',
        style: TextStyle(
            fontSize: 12, color: Color.fromARGB(255, 93, 91, 91)),
      ),
    ],
  ),
),
Padding(
  padding: const EdgeInsets.all(10.0),
  child: Row(
    children: [
      CircleAvatar(
        backgroundColor: Colors.grey.withOpacity(0.5),
        child: Icon(
          Icons.phone,
          color: Color.fromARGB(255, 52, 50, 50),
        ),
      ),
      SizedBox(
        width: 10,
      ),
      CircleAvatar(
        backgroundColor: Colors.grey.withOpacity(0.5),
        child: Icon(
          Icons.email,
          color: Color.fromARGB(255, 52, 50, 50),
        ),
      ),
      SizedBox(
        width: 10,
      ),
      CircleAvatar(
        backgroundColor: Colors.grey.withOpacity(0.5),
        child: Icon(
          Icons.person,
```

```
            color: Color.fromARGB(255, 52, 50, 50),
          ),
        )
      ],
    ),
  ),
  Padding(
    padding: const EdgeInsets.symmetric(vertical: 10),
    child: Divider(
      height: 1,
      color: Color.fromARGB(255, 223, 212, 212),
    ),
  ),
  Padding(
    padding: const EdgeInsets.only(left: 10, bottom: 10),
    child: Text(
      'Information:-',
      style: TextStyle(
          fontSize: 19,
          fontWeight: FontWeight.bold,
          color: Color.fromARGB(255, 19, 14, 47)),
    ),
  ),
  Expanded(
      child: Column(
    children: [
      Container(
        width: double.maxFinite,
        margin: EdgeInsets.only(left: 10, right: 10, bottom: 16),
        child: Text(
          'we are proud to share that the members of INNOGEEKS club have participated
and won in various National and International Level Hackathons & Competitions,
including Smart India Hackathon, NASA International Space Apps Challenge, ACM ICPC,
Virtual Hacks, Snapchat India Lensathons, and many more. Given below the links to club's
various achievements. ',
          style: TextStyle(
              fontSize: 16,
              fontWeight: FontWeight.w500,
              color: Color.fromARGB(255, 99, 99, 99)),
        ),
      ),
```

```
            Container(
              width: double.maxFinite,
              margin: EdgeInsets.only(left: 10, right: 10, bottom: 16),
              child: Text(
                "Innogeeks has started launching its yearly Technical E-Magazine, Bits N’
Bytes from 2020, Here is the link to latest Editions:Edition 1:
http://online.anyflip.com/oalvj/keql/mobile/index.html Edition 2:
https://drive.google.com/file/d/1OLmaneCj6uqV52eND2pku1KDfiTbalcD/view?usp=drive
sdk",
                style: TextStyle(
                    fontSize: 16,
                    fontWeight: FontWeight.w500,
                    color: Color.fromARGB(255, 99, 99, 99)),
              ),
            ),

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

CHAPTER 7

TESTING

7.1 UNIT TESTING

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process. It forms the basis for componenttesting. Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit testsare typically written and run by software developers to ensure that code meets its design and behaves as intended.

Benefits of Unit Testing -

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it offers several benefits.

1. **Find problems early:**

   Unit testing finds problems early in the development cycle. In test- driven development (TDD), which is frequently used in both extreme programming and scrum, unit tests are created before the code itself is written. When the tests pass, that code is considered complete. The same unit tests are run against that function frequently as the larger code base is developed either as the code is changed or via an automated process with the build. If the unit tests fail, it is a bug either in the changed code or the tests themselves. The unit tests then allow the location of the fault or failure to be easily traced. Since the unit tests alert the development team of the problem before handing the code off to testers or clients, it is still early in the development process.

2. **Facilitates Change:**

   Unit testing allows the programmer to refactor code or upgrade system libraries later, and make sure the module still works correctly (e.g., in regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified. Unit tests detect changes which may break a design contract.

3. **Simplifies Integration:**

   Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.

4. **Documentation:**

   Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is provided by a unit, and how to use it, can look at the unit tests to gain a basic understanding of the unit's interface (API). Unit test cases embody characteristics that are critical to the success of the unit. These characteristics can indicate appropriate/inappropriate use of a unit as well as negative behaviors that are to be trapped by the unit. A unit test case, in and of itself, documents these critical characteristics, although many software development environments do not rely solely upon code to document the product in development unit. A unit test case, in and of itself, documents these critical characteristics, although many software development environments do not rely solely upon code to document the product in development.

7.2 INTEGRATION TESTING

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e., assemblages (or groups of units), are exercised through their interfaces using black-box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface. Test cases are

constructed to test whether all the components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e., unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages. Software integration testing is performed according to the software development life cycle (SDLC) after module and functional tests. The cross-dependencies for software integration testing are schedule for integration testing, strategy and selection of the tools used for integration, define the cyclomatic complexity of the software and software architecture, reusability of modules and life cycle and versioning management. Some different types of integration testing are big-bang, top-down, and bottom-up, mixed (sandwich) and risky- hardest. Other Integration Patterns are collaboration integration, backbone integration, layer integration, client-server integration, distributed services integration and high-frequency integration.

1. Big Bang

In the big-bang approach, most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. This method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing. A type of big-bang integration testing is called "usage model testing" which can be used in both software and hardware integration testing. The basis behind this type of integration testing is to run user-like workloads in integrated user-like environments. In doing the testing in this manner, the environment is proofed, while the individual components are proofed indirectly through their use. Usage Model testing takes an optimistic approach to testing because it expects to have few problems with the individual components. The strategy relies heavily on the component developers to do the isolated unit testingfor their product. The goal of the strategy is to avoid redoing the testing done by the developers, and instead flesh-out problems caused by the interaction of the components in the environment. For integration testing, Usage Model testing can be more efficient and provides better test coveragethan traditional focused functional integration testing. To be more efficient and accurate, care must be used in defining the user-like workloads for creating realistic scenarios in exercising the environment. This gives confidence that the integrated environment will work as expected for the target customers.

2. Top-down And Bottom-up

Bottom-up testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher-level

components. The process is repeated until thecomponent at the top of the hierarchy is tested. All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower-level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage. Top-down testing is an approach to integrated testing where the top integrated modules are tested, and the branch of the module is tested step by step until the end of the related module. Sandwich testing is an approach to combine top-down testing with bottom-up testing.

7.3 SOFTWARE VERIFICATION AND VALIDATION

Software Verification and Validation (V&V) is an essential process in the development lifecycle of any software application, including the KIET Event Management App. It ensures that the software meets the specified requirements, functions correctly, and performs as intended. Here are some key aspects of software verification and validation for the app:

1. Requirement Verification:
    3. Verify that all functional and non-functional requirements for the app have been properly documented and understood.
    4. Validate that the app's features and functionalities align with the specified requirements.
    5. Conduct reviews and inspections to ensure that the requirements are clear, complete, and consistent.
2. Design Verification:
    - Verify that the app's design accurately represents the intended architecture and structure.
    - Review the app's design documents, diagrams, and prototypes to ensure they meet the functional and performance requirements.
    - Validate that the design adheres to industry standards, best practices, and usability guidelines.
3. Code Verification:
    - Conduct code reviews and inspections to ensure the quality, readability, and maintainability of the code.
    - Perform static code analysis to identify potential issues such as coding errors, security vulnerabilities, and performance bottlenecks.
    - Use automated testing tools to verify the correctness of the code and ensure

it meets the functional requirements.

4. Functional Testing:
    - Perform comprehensive functional testing to validate that all app features and functionalities work as intended.
    - Create test cases and test scenarios that cover all aspects of the app's functionality.
    - Execute various types of testing, such as unit testing, integration testing, and system testing, to ensure the app performs as expected.
5. Performance Testing:
    - Conduct performance testing to verify that the app can handle the expected user load and perform efficiently under different conditions.
    - Measure and analyze the app's response times, resource utilization, scalability, and reliability.
    - Identify and address any performance bottlenecks or issues that may affect the user experience.
6. Usability Testing:
    - Evaluate the app's usability through user testing and feedback.
    - Ensure that the user interface is intuitive, user-friendly, and meets the needs of the target users.
    - Incorporate user feedback to improve the app's usability and user satisfaction.
7. Security Testing:
    - Conduct security testing to identify and address potential vulnerabilities and risks.
    - Perform penetration testing, vulnerability scanning, and code analysis to ensure the app's security measures are in place.
    - Implement appropriate security controls, such as authentication, authorization, encryption, and data protection.
8. Validation Testing:
    - Validate the app against the user requirements and expectations.
    - Conduct user acceptance testing (UAT) to ensure that the app meets the needs of the stakeholders and end-users.
    - Obtain feedback from users and stakeholders to validate the app's functionality, usability, and overall satisfaction.
9. Documentation and Compliance:
    - Ensure that all relevant documentation, including user manuals, installation guides, and release notes, are accurate and up to date.
    - Verify compliance with industry standards, legal regulations, and data privacy requirements.
10. Maintenance and Support:

- Establish a process for ongoing maintenance and support of the app.
- Monitor the app's performance, address user feedback and bug reports, and release patches and updates as needed.
- Continuously improve the app based on user feedback, changing requirements, and emerging technologies.

It is important to note that the specific verification and validation activities and techniques may vary depending on the development methodology, project constraints, and the nature of the KIET Event Management App. Therefore, it is recommended to tailor the V&V process to the specific requirements and characteristics of the project.

7.4 BLACK-BOX TESTING

Black box testing is a technique used to test the functionality of a software application without having knowledge of its internal structure or implementation details. It focuses on the inputs and outputs of the system and verifies if the expected outputs match the desired results. Here are some examples of black box testing techniques that can be applied to the KIET Event Management App:

1. Equivalence Partitioning:
    - Identify different categories of inputs for the app, such as valid and invalid inputs, and divide them into equivalence classes.
    - Test representative values from each equivalence class to ensure the app behaves consistently within each class.
2. Boundary Value Analysis:
    - Identify the boundaries or limits for inputs in the app, such as minimum and maximum values, and test values at those boundaries.
    - Test values just above and below the boundaries to verify the app's behavior at critical points.
3. Decision Table Testing:
    - Identify the different conditions and rules that govern the behavior of the app.
    - Create a decision table with combinations of conditions and corresponding expected results.
    - Test different combinations of conditions to validate the app's decision-making process.
4. State Transition Testing:
    - Identify the different states that the app can transition between.
    - Define the valid and invalid transitions between states.
    - Test different sequences of state transitions to verify the app's behavior.
5. Error Guessing:

- Use experience and intuition to guess potential errors or issues in the app.
- Create test cases based on those guesses to verify if the app handles the errors correctly.

6. Compatibility Testing:
   - Test the app on different platforms, browsers, or devices to ensure compatibility.
   - Verify that the app functions correctly and displays appropriately across different environments.
7. Usability Testing:
   - Evaluate the app's user interface and interactions from the perspective of an end-user.
   - Test common user scenarios and assess the app's ease of use, intuitiveness, and overall user experience.
8. Security Testing:
   - Test the app for potential security vulnerabilities or weaknesses.
   - Verify if the app handles user authentication, data encryption, and access control appropriately.
9. Performance Testing:
   - Test the app's performance under different load conditions, such as a high number of concurrent users or large data sets.
   - Verify if the app responds within acceptable time limits and performs efficiently.
10. Regression Testing:
    - Re-test previously identified defects or areas of the app that have undergone changes to ensure that fixes or modifications did not introduce new issues.

During black box testing, test cases are designed based on the app's specifications, requirements, and user expectations. The focus is on validating the functionality, user interactions, and expected outputs without considering the internal implementation details of the app. By applying black box testing techniques, the KIET Event Management App can be thoroughly tested for functional correctness, usability, security, and performance.

7.5 WHITE-BOX TESTING

White box testing, also known as structural testing or glass box testing, is a software testing technique that examines the internal structure and implementation details of the application. It aims to ensure that the code functions as intended and covers all possible execution paths. Here are some examples of white box testing techniques that can be applied to the KIET Event Management App:

1. Unit Testing:

- Test individual units or components of the app, such as functions or methods, to verify their correctness.
- Use techniques like code coverage analysis (e.g., statement coverage, branch coverage) to ensure that all code paths are exercised.

2. Integration Testing:
   - Test the interaction between different components or modules of the app to ensure they work together seamlessly.
   - Verify the flow of data and control between the modules and check for any integration issues or errors.
3. Path Testing:
   - Identify and test different paths or execution flows through the app, including both positive and negative scenarios.
   - Execute test cases that cover all possible paths within the code to ensure complete coverage.
4. Decision Coverage:
   - Ensure that every decision point in the code (e.g., if statements, switch cases) is tested for both true and false conditions.
   - Validate that the app makes the correct decisions based on the specified conditions.
5. Code Review:
   - Analyze the code and its structure to identify any potential issues or vulnerabilities.
   - Review the adherence to coding standards, best practices, and potential optimizations.
6. Performance Testing:
   - Assess the app's performance from a code perspective, such as identifying any bottlenecks or inefficient algorithms.
   - Measure the execution time of critical code sections and evaluate resource usage.
7. Security Testing:
   - Review the code for potential security vulnerabilities, such as SQL injection, cross-site scripting (XSS), or authentication weaknesses.
   - Verify the implementation of secure coding practices, data encryption, and access control mechanisms.
8. Error Handling Testing:
   - Test how the app handles and recovers from unexpected errors or exceptions.
   - Validate that error messages are clear, meaningful, and do not expose sensitive information.
9. Code Coverage Analysis:

- Use tools to measure the code coverage achieved by the tests, such as statement coverage, branch coverage, or path coverage.
- Aim for high code coverage to ensure that all parts of the code are exercised.

10. Regression Testing:
    - Re-test previously tested code segments that have undergone modifications to ensure that changes did not introduce new bugs or regressions.

During white box testing, the tester has access to the application's internal code, allowing for a more detailed examination of its behavior. By applying white box testing techniques, potential issues within the code can be identified and addressed, improving the overall quality and reliability of the KIET Event Management App.

7.6 SYSTEM TESTING

System testing is a level of software testing that evaluates the complete system as a whole, rather than focusing on individual components or modules. It ensures that all components of the KIET Event Management App work together seamlessly and meet the specified requirements. Here are some examples of system testing techniques that can be applied to the app:

1. Functional Testing:
    - Verify that all functional requirements of the app are met.
    - Test various functionalities such as event creation, registration, club directory search, user login and registration, event notifications, etc.
    - Validate that the app behaves as expected and produces the correct outputs based on different inputs.
2. User Interface Testing:
    - Test the graphical user interface (GUI) of the app for usability, consistency, and responsiveness.
    - Check the layout, navigation, buttons, forms, and other UI elements to ensure they are visually appealing and intuitive.
    - Validate that the app adheres to the design guidelines and provides a seamless user experience.
3. Performance Testing:
    - Evaluate the performance of the app under different load conditions.
    - Measure response times, throughput, and resource utilization to ensure the app can handle the expected user load without significant degradation.
    - Identify and address any performance bottlenecks or scalability issues.
4. Compatibility Testing:
    - Test the app on different devices, platforms, and browsers to ensure

compatibility.

- Verify that the app works correctly on various operating systems (e.g., iOS, Android) and different screen sizes.
- Validate that the app functions properly on different web browsers (if applicable).

5. Security Testing:
    - Assess the app's security measures to protect user data and prevent unauthorized access.
    - Perform vulnerability scanning, penetration testing, and authentication testing to identify and address any security vulnerabilities.
    - Test the app's resilience against common security threats, such as cross-site scripting (XSS) and SQL injection.
6. Usability Testing:
    - Evaluate the app's usability and user-friendliness.
    - Conduct user tests and gather feedback to assess the ease of use, intuitiveness, and overall user satisfaction.
    - Identify areas of improvement and make necessary adjustments to enhance the user experience.
7. Integration Testing:
    - Test the integration of the app with external systems, such as databases, mapping services, or notification services.
    - Validate that data is exchanged correctly between the app and external systems.
    - Verify that the app's functionality remains intact when integrated with other systems.
8. Recovery Testing:
    - Simulate system failures or interruptions and evaluate the app's ability to recover and resume normal operation.
    - Test scenarios such as unexpected shutdowns, network failures, or interrupted database connections.
    - Ensure that the app can gracefully handle such situations and recover without data loss or integrity issues.
9. Regression Testing:
    - Re-test previously tested features and functionalities to ensure that recent changes or additions did not introduce new bugs or regressions.
    - Execute a set of comprehensive test cases to cover critical areas of the app and ensure that no existing functionality is compromised.
10. Documentation Testing:
    - Review the app's documentation, including user manuals, installation guides, and technical documentation.

- Validate that the documentation is accurate, up-to-date, and provides sufficient guidance for users and administrators.

System testing aims to verify the overall behavior and performance of the KIET Event Management App. By conducting thorough system testing, any issues or inconsistencies can be identified and resolved, ensuring a reliable and high-quality application for users.

CHAPTER-8

FUTURE SCOPE

The KIET Event Management App has a promising future scope with potential avenues for further development and enhancement. Here are some possible future directions for the app:

1. Advanced Analytics: Incorporating advanced analytics capabilities into the app can provide valuable insights into event attendance, user preferences, and engagement patterns. This data can be used to optimize event planning, target specific user groups, and improve overall event experiences.
2. Gamification Features: Introducing gamification elements such as badges, rewards, leaderboards, and challenges can make event participation more engaging and encourage increased user involvement. Gamification can also foster healthy competition among attendees and promote social interaction.
3. Social Media Integration: Integrating the app with popular social media platforms can facilitate seamless sharing of event details, updates, and photos, amplifying event reach and promoting user-generated content. This integration can enhance the app's social aspect and further increase event awareness.
4. Virtual and Hybrid Events: With the growing trend of virtual and hybrid events, incorporating features to support these event formats can be a valuable addition. This may include integrating virtual meeting platforms, live streaming capabilities, and interactive chat features to enable remote attendees to participate fully.
5. Ticketing and Payment Integration: Enhancing the app with integrated ticketing and secure payment systems can streamline the ticket purchase process for attendees. This can include support for various payment methods, ticket scanning, and secure transaction processing.
6. Sponsorship and Advertising Opportunities: Providing sponsorship and advertising opportunities within the app can generate additional revenue streams while promoting relevant brands and services to event attendees. This can include banner ads, sponsored event listings, and targeted promotional campaigns.
7. Feedback and Rating System: Implementing a feedback and rating system for events and clubs can provide valuable feedback to organizers and help improve future events. Attendees can provide ratings, reviews, and suggestions, contributing to continuous improvement and enhancing the overall event

experience.

8. Integration with Campus Services: Expanding the app's integration with other campus services and systems, such as library services, facility bookings, and academic calendars, can create a comprehensive platform that caters to various aspects of student life and engagement.
9. Enhanced Personalization: Developing personalized recommendations and tailored event suggestions based on user preferences, past attendance, and interests can enhance the user experience and increase engagement. This can include intelligent event suggestions, personalized notifications, and customized content delivery.
10. Multilingual Support: Incorporating multilingual support into the app can cater to a diverse user base, especially in educational institutions with international students. Offering language options can enhance accessibility and inclusivity, facilitating better user engagement.
11. New Student Orientation: Implementing a dedicated section in the app specifically designed to assist new students in exploring and joining college clubs. This section can provide comprehensive information about various clubs, including their objectives, activities, meeting schedules, and membership criteria. It can also include contact information for club leaders or advisors, making it easier for new students to reach out and express their interest in joining.
12. Club Reviews and Ratings: Introducing a club review and rating feature within the app can allow current and former club members to share their experiences and provide feedback on different clubs. This can help new students make informed decisions when choosing which clubs to join based on the experiences and opinions of their peers.
13. Club Events and Activities Calendar: Creating a shared calendar within the app specifically for club events and activities can help new students easily find and participate in club-related events. This calendar can include details such as event descriptions, dates, times, locations, and any associated costs. It can also allow users to RSVP or add events to their personal calendars for convenient planning.
14. Club Membership Requests: Integrating a feature that allows new students to submit membership requests directly through the app can streamline the process of joining clubs. This feature can include a form or online application that collects necessary information from the student and automatically forwards it to the club's designated contact person for review and approval.
15. Club Matching Algorithm: Developing an algorithm or recommendation system within the app that suggests clubs to new students based on their interests, academic program, or extracurricular preferences can assist in finding clubs that align with their individual preferences. This personalized club matching feature can help new students discover clubs that they may not have otherwise considered.
16. Virtual Club Meetups and Hangouts: In addition to physical club meetings and events, incorporating virtual meetups and hangouts within the app can provide an

opportunity for new students to interact with club members remotely. This feature can facilitate virtual club introductions, discussions, and networking, particularly for students who may not be able to attend in-person meetings regularly.

17. Club Resource Library: Creating a resource library within the app where clubs can upload documents, presentations, or other useful materials can enhance the accessibility of club-related information. This can include club constitutions, event planning guides, project templates, and other resources that can help new students understand club operations and get involved more effectively.

By incorporating these club-related features into the KIET Event Management App, new students can have a convenient and comprehensive platform to explore, join, and engage with various clubs on campus. This will not only help them integrate into the college community but also provide opportunities for personal growth, skill development, and social connections.

CHAPTER-9

CONCLUSION

In conclusion, the KIET Event Management App offers a comprehensive solution for organizing and managing events in a college setting. By considering each point discussed and incorporating club-related information, the app becomes a valuable resource for both event planning and club engagement.

The app's user-friendly interface, coupled with features such as event creation and management, RSVP tracking, and real-time updates, streamlines the event planning process and improves communication between event organizers and attendees. This ensures efficient and effective event management, leading to successful and engaging college events.

Moreover, the inclusion of a club directory with detailed information about various clubs on campus makes it easier for students to explore and join extracurricular activities. The app provides club descriptions, leadership details, eligibility criteria, and contact information, empowering students to make informed decisions about which clubs align with their interests and goals. Additionally, the integration of a club matching algorithm and club reviews/ratings feature enhances the discovery process and allows students to connect with clubs that best suit their preferences and aspirations.

Furthermore, the app offers features such as a club events and activities calendar, club membership request submissions, virtual club meetups, and a club resource library. These features facilitate seamless club engagement, enable efficient event coordination, and provide resources for club operations and growth. By combining event management functionalities with club-related information, the KIET Event Management App creates a centralized and user-friendly platform that simplifies the process of organizing events and getting involved in clubs. It promotes student engagement, fosters a sense of community, and supports personal and professional development.

Overall, the KIET Event Management App has the potential to revolutionize event planning and club engagement in a college setting, providing a convenient and efficient solution that enhances the college experience for students, faculty, and staff.

CHAPTER 10

BIBLIOGRAPHY