Blockchain University Voting System

By

Tiew Jia Jun

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY

TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY
KUALA LUMPUR

ACADEMIC YEAR
Choose an item.

Blockchain University Voting System

By

Tiew Jia Jun

Supervisor: Chai Foong Theng

A project report submitted to the
Faculty of Computing and Information Technology
in partial fulfillment of the requirement for the
Choose an item.

Choose an item.
Faculty of Computing and Information Technology
Tunku Abdul Rahman University of Management and Technology
Kuala Lumpur

Declaration

The project submitted herewith is a result of my own efforts in totality and in every aspect of the project works. All information that has been obtained from other sources had been fully acknowledged. I understand that any plagiarism, cheating or collusion or any sorts constitutes a breach of TAR University rules and regulations and would be subjected to disciplinary actions.

Tiew Jia Jun

Choose an item.

ID: 22WMR13756

Abstract

This project presents the design and development of a blockchain-based voting system tailored for educational institutions. Designed to improve transparency, security, and efficiency of institutional elections, the system leverages distributed ledger technology to mitigate vulnerabilities associated with traditional voting methods. Key objectives include ensuring data integrity through immutable records, user verification.

The project adopted an agile development approach, covering iterative planning, design, development, testing, and deployment phases. Key components included smart contract development for automated vote counting, a mobile-friendly interface using Flutter, and blockchain integration for secure voting records. The system was extensively tested for functionality, scalability, and user-friendliness, demonstrating its ability to support diverse electoral processes for institutions of varying sizes.

Survey results from potential users indicated strong support for the proposed solution, emphasizing priorities such as usability, security, and real-time transparency. The implementation demonstrated strong features such as multi-platform accessibility, multi-language support, and real-time notifications. The final deployment enabled a secure, efficient, and convenient voting process, promoting trust and engagement in the educational community. This innovation embodies the potential of blockchain technology to address modern governance challenges and sets a benchmark for the future of institutional elections.

Acknowledgement

This contains acknowledgement to those who have contributed directly or indirectly to the completion of the project. Usually the people to be acknowledged include the project supervisor(s), moderators, family, and those who have given assistance and supports to ensure the success of the project.

Table of Contents

Chapter 1

Introduction

1 Introduction

This chapter introduces a blockchain-based voting system designed for educational institutions to ensure secure, transparent, and efficient elections. It highlights the system's objectives, such as improving security through decentralized ledgers. Key features include multi-school election support, mobile accessibility, and multi-language capabilities. The document also outlines the project's advantages, including enhanced trust, scalability, and user-friendly interfaces, as well as the development plan, milestones, and team responsibilities. Overall, it presents a modern solution to address vulnerabilities in traditional voting methods, fostering trust and participation in institutional governance.

1.1 Objectives

The main goal of the project is to enhance the security and integrity of the voting process across educational institutions by developing a decentralized voting system that leverages blockchain technology. Traditional voting methods are often subject to vulnerabilities such as tampering, unauthorized access, and manipulation of results. By implementing a transparent and unchangeable ledger, the system aims to provide a secure and reliable election platform.

In addition to security, improving the overall efficiency of the voting process is also a key goal. The system will automatically record and count votes, greatly reducing delays and ensuring faster and more reliable results. This automation will not only ensure elections are secure, but also enable elections to be conducted in a timely manner, thereby minimizing disputes related to vote counting.

Accessibility and scalability are also priorities for the system. Designed to be user-friendly and accessible through a variety of devices, the system will enable all eligible voters, including remote voters or those with limited mobility, to participate in the election process.

The transparency inherent in blockchain technology allows for constant monitoring of the voting process, ensuring that the integrity of the election is maintained at all times. This capability enhances trust and gives all stakeholders confidence that the results of the election are accurate and fair.

The voting system is designed to accommodate different voting groups in educational settings. It has the flexibility to handle a variety of types of elections and decision-making processes. By ensuring accurate, secure and transparent voting, the system aims to increase trust and participation in elections and decision-making processes across multiple educational institutions, thereby changing the way schools and universities engage their communities in governance.

1.2 Project Background / Introduction

Voting is a crucial aspect of decision-making within educational institutions, influencing everything from student government leadership to important policy decisions made by administrators. However, traditional voting systems—whether paper-based or electronic—have faced significant challenges, including ballot tampering, unauthorized access, delays in counting votes, and manipulation of results. Such issues can erode trust among students and raise serious doubts about the fairness and legitimacy of election outcomes.

As educational institutions strive for more transparent, fair, and secure election processes, the demand for modern and reliable voting systems is growing. This project aims to address these challenges by developing a blockchain-based voting system specifically tailored for schools and universities. By leveraging blockchain technology, which features a decentralized and tamper-proof ledger, the system ensures that every vote is securely recorded and cannot be altered or tampered with. This transparency not only enhances security but also helps build trust among all stakeholders, including students, faculty, staff, alumni, and school board members.

A primary objective of this project is to enhance the security and integrity of the voting process by providing a decentralized platform that mitigates key vulnerabilities present in traditional voting methods.

Additionally, the project aims to improve the overall efficiency of the voting process. By automatically recording and counting votes, the system will significantly reduce delays and provide faster, more reliable results. This ensures that elections are not only secure but also timely, thus preventing disputes related to vote counting.

The voting system will also prioritize accessibility and scalability. It will feature a user-friendly interface, enabling all eligible voters—whether remote participants or those with limited mobility—to engage in the electoral process.

By adopting this blockchain voting system, schools can effectively overcome the limitations of traditional voting methods, enhance trust in the electoral process, and ensure accurate and fair results. This innovative approach not only modernizes school voting systems but also guarantees the transparency, security, and efficiency essential for all participants in the democratic process.

1.3 Features list / Function list

Table 1.1. Features of the project

No.	Features
1.	**User Registration and Login** ● Allows schools, universities or colleges to register on the platform. ● Secure user authentication using email, password verification. ● Different roles: Voters(Student), Staff, Admin.
2.	**School/Institution Management** ● Schools, universities or colleges can register, manage their profiles and organize elections. ● Admin panel to manage voting activities, participants and institution details.
3.	**Election Creation and Configuration** ● Institution administrators can create and configure different types of elections. ● Ability to set election rules, eligibility, start/end dates and candidate lists.
4.	**Voter Authentication** ● Prevent unauthorized users from voting.
5.	**Blockchain-based Voting System**

	● Securely and immutably record votes using blockchain technology. ● Voters can submit their ballots securely and the results are recorded on-chain for transparency. ● Ensures integrity and immutability of the voting process.
6.	**Real-time Voting Status and Updates** ● Provides real-time information on the voting process, such as ongoing elections and voter participation rates. ● Shows the current status of elections and voting progress.
7.	**Counting and Results** ● Votes are counted automatically using blockchain's consensus mechanism. ● Results are generated and immutably stored on-chain.
8.	**Mobile-friendly Voting** ● The platform is available on mobile devices (Android and iOS), allowing voters to participate from anywhere. ● Native mobile app interface (based on Flutter) for ease of use.
9.	**Notifications and Reminders** ● Notifications for election updates, deadlines, and voting reminders. ● Push notifications for mobile users and email notifications for web users.
10.	**Multi-language Support** ● Multi-language capabilities are available to cater to different student and staff groups in different regions.
11.	**Custom Reports and Analytics** ● Generate custom reports for administrators on voter turnout, election statistics, and results. ● Visual dashboards for in-depth insights into election performance.
12.	**Data Privacy and Security** ● Strong encryption and privacy measures ensure voter data and votes are secure. ● No personally identifiable data is stored on the blockchain, ensuring user

	identities are safe.
13.	**User-friendly interface** ● The platform's web and mobile interfaces are simple and intuitive. ● Suitable for users of all skill levels.

1.4 Advantages & Contributions

1. **Enhanced Security and Integrity**

   The system utilizes blockchain technology to securely record every vote in a decentralized ledger, making it virtually impossible to alter or tamper with voting data. This security feature addresses concerns about fraud and manipulation, promoting trust among voters and stakeholders.

2. **Transparency**

   With the ability to monitor the voting process in real time, stakeholders can verify the accuracy and integrity of election results at any time. This transparency enhances confidence in the election process and ensures that decisions are made fairly.

3. **Efficient Voting Process**

   The system automatically records and counts votes, streamlining the election process and minimizing delays. This efficiency reduces the likelihood of disputes related to vote counting and enables schools to make timely decisions.

4. **User-Friendly Interface**

   Designed with a focus on user experience, the system allows students, faculty, staff, and alumni to easily complete the voting process. This intuitive interface encourages participation from all eligible voters, including those with limited technical skills.

5. **Promotes Democratic Participation**

   By providing a secure and transparent voting platform, the system encourages greater participation in school elections and decision-making processes. Increased participation can lead to a more informed and active student body, fostering a vibrant school community.

6. **Competitive Advantages**

   The integration of blockchain authentication sets this voting system apart from traditional methods, offering a modern, secure, and efficient alternative. This innovative approach enhances the educational institution's reputation as a leader in adopting cutting-edge solutions.

7. **Educational Value**

   As an educational tool, the system demonstrates the principles of democratic participation, technology integration, and the importance of secure voting practices. It provides students with hands-on experience participating in a transparent electoral process, preparing them for their future civic responsibilities.

8. **Social Impact**

   By addressing the limitations of traditional voting systems, the project helps schools build more trustworthy and reliable election processes. It empowers students and staff to make their voices heard, ensuring their participation in decision-making is respected and valued.

1.5 Project plan / Milestones / Gantt Chart

Table 1.2. Project plan table

ACTIVITIES	EXPECTED OUTCOME	COMPLETION DATE
Initial Research	Understanding of needs, and the project requirements.	September 22, 2024
Requirement Analysis	Detailed requirements document and use cases.	October 20, 2024
System Design & Architecture	System architecture, detailed design, and technology stack.	December 1, 2024
Prototyping	Working prototype and user feedback.	January 5, 2025
Smart Contract Development	Functional, secure smart contracts ready for deployment.	February 2, 2025

Frontend and Backend Development	Developed user interfaces, backend services, and APIs.	April 1, 2025
Integration and Testing	Integrated system with thorough test reports and bug fixes.	May 15, 2025
Deployment & Maintenance	Live system with user training, support plan, and performance monitoring.	June 15, 2025

Gantt Chart

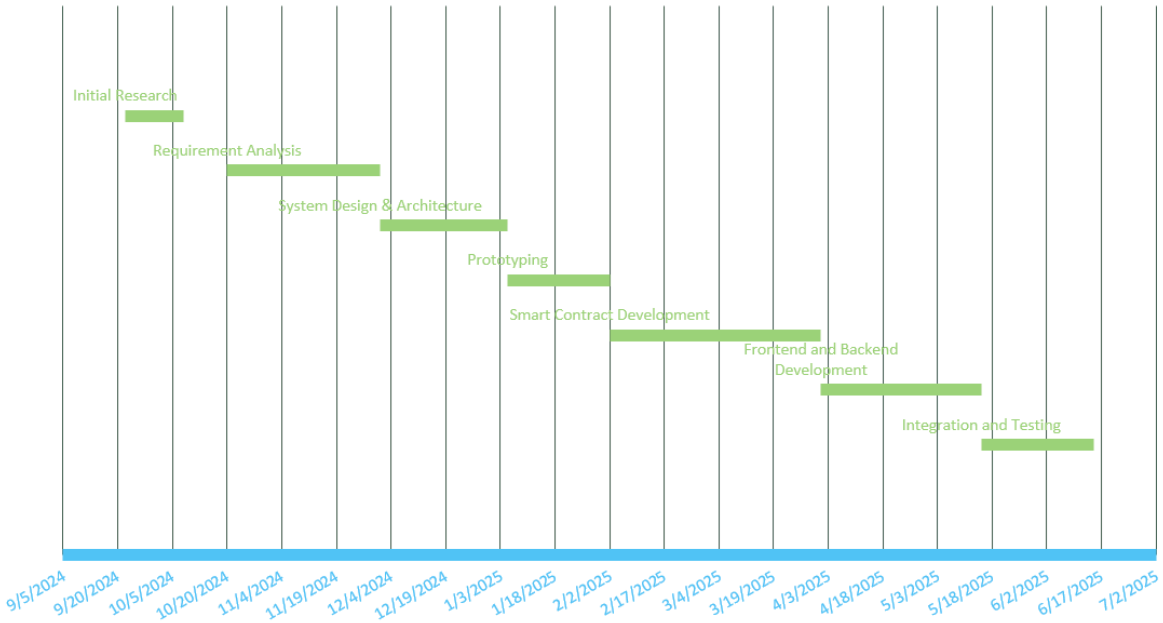


Figure 1.1. Project Gantt Chart

1.6 Project Team & Organization

1. **Smart Contract Development (Blockchain)**
    - Design and implement smart contracts using Solidity to handle key functions such as voting, counting results, and verifying voter authenticity.
    - Ensure the security and immutability of contracts to prevent tampering or manipulation of votes.
    - Test smart contracts on a global blockchain environment (e.g., Remix) and deploy them on the Ethereum blockchain.

2. **Backend Development**

- Develop server-side logic to manage user registration, election creation, and vote submission.
- Implement APIs to communicate between the blockchain and the front end to perform operations such as voting and getting election results.
- Integrate database systems to store non-sensitive data that does not require blockchain-level security (e.g., election information).

3. **Mobile Application Development**
   - Develop a mobile application using Flutter to provide cross-platform access for Android and iOS users.
   - Ensure that the mobile application seamlessly syncs with the backend and blockchain to enable real-time voting and results updates.

4. **Testing and Debugging**
   - Perform thorough testing to ensure the system can smoothly handle multiple users, votes, and election types.
   - Test edge cases for potential security vulnerabilities, such as unauthorized access and duplicate votes.
   - Implement debugging mechanisms to log and track errors to ensure a smooth user experience.

5. **Security Enhancements**
   - Integrate advanced security features such as encryption to protect voter data.
   - Implement measures to prevent DDoS attacks or hacker attacks that could compromise the platform.

6. **Deployment and Maintenance**
   - Deploy the blockchain-based voting system into web and mobile production environments.
   - Provide ongoing monitoring and maintenance to ensure the system operates efficiently during the election.
   - Update the system as needed to improve performance, scalability, and security.

1.7 Chapter Summary and Evaluation

The main goal of the project was to develop a secure, transparent, and efficient voting system tailored for schools using blockchain technology. The project successfully achieved this goal by addressing key vulnerabilities in traditional voting systems, such as tampering, unauthorized access, and vote manipulation. By implementing smart contracts, and decentralized data storage, the system ensures the integrity of the voting process, greatly enhancing trust in election results.

One of the main achievements of the project is its ability to support different voting groups (students, faculty, and school board members), allowing for various types of elections, from student government to policy decisions. In addition, the system's scalability and user-friendly interface ensure that it can handle elections for schools of any size, making it highly adaptable to different institutions.

In summary, the system not only achieves the goal of modernizing the school voting process, but also provides users with a valuable educational experience. It sets a benchmark for secure, reliable, and transparent elections, allowing schools to foster greater trust and participation in decision-making.

Chapter 2

Literature Review

2 Literature Review

This chapter delves into the development of a blockchain-based voting system tailored for Student Representative Council elections in educational institutions. It explores key technological components such as blockchain, smart contracts, and private blockchains, emphasizing their role in enhancing the transparency, security, and efficiency of electoral processes. The chapter also assesses the feasibility of implementing such a system through technical, financial, and operational lenses. By addressing vulnerabilities in traditional voting systems, the study highlights the potential of blockchain technology to foster trust, participation, and accountability in democratic governance within academic environments.

2.1 Research Background and Related Work

The advent of blockchain technology has revolutionized sectors as diverse as finance, supply chain, and governance. Its most significant promise lies in its ability to improve transparency, security, and efficiency of operations. In the context of democratic processes, the integration of blockchain with voting systems has attracted significant attention from researchers, policymakers, and practitioners. Traditional voting methods—whether paper-based or electronic—often suffer from vulnerabilities such as tampering, voter fraud, and inefficiencies in counting and verifying votes (Huang et al., 2022; Yang et al., 2021). These weaknesses can lead to a lack of trust in election results and erode public confidence in democratic institutions.

As educational institutions increasingly turn to technology to increase transparency and trust in elections, challenges associated with traditional voting practices require innovative solutions that leverage emerging technologies. Blockchain technology provides a decentralized and immutable ledger that ensures that all votes are securely recorded and cannot be altered once submitted. This transparency not only enhances the integrity of the voting process but also builds confidence among stakeholders, including students, faculty, staff, and alumni (Ch et al., 2022).

Additionally, the use of smart contracts in blockchain systems facilitates automated vote counting, further streamlining the election process (Weiss et al., 2022). By eliminating human intervention, smart contracts reduce the potential for human error and increase the speed and accuracy of election results. Blockchain technology therefore makes a compelling case for improving the election process in educational institutions, providing a path to more democratic and participatory governance.

2.2 Project Background

The project aims to develop a blockchain-based voting system specifically for Student Representative Council elections in educational institutions. The main goal is to create a platform that will not only secure and streamline the voting process but also promote participation and trust in the democratic process (Rathnayake, 2022). By addressing key vulnerabilities associated with traditional voting methods, the proposed system aims to ensure that every vote is valid and election results are transparent and reliable.

2.3 Literature Review

2.3.1 Blockchain Technology

Blockchain technology is the foundation of the proposed voting system. Its decentralized nature prevents any single entity from controlling the voting process, thereby reducing the risks associated with tampering and manipulation (Yang et al., 2021). Research shows that blockchain's transparency allows for continuous monitoring of the voting process, which is essential for maintaining integrity (Huang et al., 2022).

In the voting field, blockchain can facilitate the entire election process, from voter registration to voting and counting. Every transaction is securely recorded on a distributed ledger, ensuring that once a vote is cast, it cannot be changed or deleted without network consensus. This immutable record of voting can effectively defend against common election integrity threats such as ballot tampering and double voting. In addition, the decentralized nature of blockchain reduces the possibility of system failure, making the voting process more resilient to attacks.

2.3.2 Smart Contracts

Smart contracts are self-executing contracts where the agreement is written directly into the code. In the context of electronic voting, smart contracts can automate various aspects of the voting process, including recording and counting votes (Weiss et al., 2022). This not only improves efficiency, but also reduces the possibility of human error, making the process more reliable and accurate (Rosasooria et al., 2020).

By implementing smart contracts, voting systems can ensure that once the voting period ends, the results are automatically calculated and made immediately available to stakeholders. This instant vote counting capability allows for faster announcement of results, which is often an important issue during elections. Additionally, smart contracts can facilitate the enforcement of voting rules, ensuring that only eligible voters can participate and that each voter only casts one vote. This reduces the risks associated with voter fraud and further increases the credibility of the electoral process.

2.3.3 Private Blockchains

While public blockchains offer transparency, they also face challenges related to performance and scalability. Private blockchains, or permissioned blockchains, offer an alternative that addresses these limitations (Rathnayake, 2022). This study explored the suitability of private blockchains for electronic voting, highlighting their ability to handle large numbers of voters while ensuring compliance with legal requirements (Yang et al., 2021).

Private blockchains allow institutions to control who can participate in the network, which can be particularly beneficial in educational settings. By limiting access to authorized users,

educational institutions can ensure that only legitimate students can vote. This not only enhances security, but also simplifies the voting process as the network can be optimized for performance without the overhead of accommodating larger public blockchain infrastructure.

2.4 Feasibility Study

2.4.1 Technical Feasibility

The technical feasibility of the proposed blockchain voting system is supported by the existing blockchain infrastructure that supports decentralized data storage and secure transactions. The implementation of smart contracts allows for automated counting, which simplifies the voting process and reduces errors associated with manual counting (Huang et al., 2022). The selection of an appropriate blockchain platform (e.g., Ethereum or Hyperledger Fabric) will depend on the specific requirements of the voting system. These platforms provide a robust framework for developing blockchain applications and can support the functionality required for a secure and efficient voting process.

2.4.2 Financial Feasibility

The financial feasibility of implementing a blockchain-based voting system involves analyzing initial development costs, ongoing operating expenses, and identifying potential sources of funding. Although blockchain technology may require significant upfront investments, its long-term benefits, such as reduced administrative costs and increased voter participation, may justify these initial expenses (Fadhilnor Rahmad et al., 2024).

2.4.3 Operational Feasibility

Operational feasibility depends on stakeholder acceptance and understanding of the technology. Training programs are essential to educate users on the system's capabilities and ensure smooth implementation (Ch et al., 2022). The platform's user-friendly design will facilitate easy access by voters, making the system more likely to be accepted by the community.
In addition, clear communication and outreach efforts are needed to inform students and teachers of the benefits and capabilities of the new voting system. Building positive perceptions of technology is essential to encourage participation and foster trust in the electoral process.

2.5 Chapter Summary and Evaluation

The literature review revealed several key technologies for implementing a blockchain-based voting system. The integration of blockchain technology, smart contracts enhances security, transparency, and efficiency, effectively addressing the vulnerabilities inherent in traditional voting methods (Weiss et al., 2022).

The decision to use smart contracts is particularly advantageous as it can automate the voting process, reduce human errors, and ensure accurate results (Rosasooria et al., 2020). Additionally, the adoption of a private blockchain model allows for greater control over the voting environment while ensuring compliance with legal requirements (Rathnayake, 2022).

In summary, the proposed blockchain-based voting system represents an innovative approach to enhance the electoral process in educational institutions. By leveraging these technologies, the

project aims to promote trust and participation in student elections, ultimately transforming the way schools and universities engage their communities in governance. The integration of secure voting mechanisms and efficient processes has the potential to reshape the landscape of student representation, ensuring that elections are fair, accessible, and reflect the wishes of the student body.

Chapter 3

Methodology and Requirements Analysis

3 Methodology and Requirements Analysis

This chapter presents the methodology and requirements analysis for developing a blockchain-based voting system. It adopts the Agile methodology to enable flexibility, iterative development, and continuous feedback. The chapter covers key phases, including planning, design, development, testing, review, and launch. It also outlines findings from user surveys to understand preferences and expectations, focusing on security, transparency, and usability. Detailed functional and non-functional requirements are specified to ensure the system's scalability, reliability, and performance. This foundation supports the development of a secure and transparent voting platform.

3.1 Methodology

For this project, **Agile development methodology** chose because it is flexible and iterative, well suited for projects where requirements change over time and regular user feedback is critical for continuous improvement. Agile allows for rapid adaptation to new requirements or unforeseen challenges, especially in complex projects such as integrating blockchain technology. Development is structured in sprints, with each sprint focused on a specific deliverable, such as user authentication, smart contract deployment and token trading functionality.

Plan phase:

Plan phase, where initial research is conducted to understand project requirements. Stakeholder interviews are performed, market trends are analyzed and the capabilities of technologies such as blockchain, Flutter and Firebase are reviewed. This phase results in a clear understanding of needs and a preliminary feature list.

Key activities in the **Plan phase**:

- Conduct initial research to understand project requirements.
- Perform stakeholder interviews to gather inputs.
- Analyze market trends and existing solutions.
- Review capabilities of technologies like blockchain, Flutter and Firebase.
- Define project scope, goals and timelines.
- Develop a preliminary feature list and prioritize requirements.

Design phase:

During the design phase, requirements are analyzed in detail. Functional and non-functional requirements are collected and use cases are created. System architecture is designed including technology stack selection, database schema and API design. Mockups and wireframes are developed to visualize the user interface.

Working prototypes are built and feedback from stakeholders is incorporated to refine the design.

Key activities in the **Design phase**:

- Gather and document functional and non-functional requirements.
- Create user stories and detailed use cases.
- Design system architecture and select the technology stack.
- Develop database schemas and API structures.
- Create mockups and wireframes for UI design.
- Build a working prototype for early user feedback.
- Incorporate stakeholder feedback to refine designs.

Develop phase:

The development phase focused on building the application. Smart contracts were written and tested for security and functionality conducted to verify performance. The user interface was developed using Flutter, while backend services and APIs were implemented to handle data management. Features user authentication and MetaMask integration were also developed during this phase.

Key activities in the **Develop phase**:

- Write and test smart contracts to ensure security and functionality.
- Develop user interfaces using Flutter.
- Build backend services and APIs for data handling.
- Implement features like user authentication and MetaMask wallet integration.
- Perform code reviews and version control management.

Test phase:

After development is complete, the project enters the testing phase. The system is integrated and fully tested including unit testing, integration testing and system testing. Errors are identified and fixed to ensure that the system is stable and runs as expected.

Key activities in the **Test phase**:

- Conduct unit testing to validate individual components.
- Perform integration testing to ensure modules work together seamlessly.
- Execute system testing to check overall performance and security.
- Identify and fix bugs to enhance system stability.
- Document test cases, results and issues for future reference.

Review phase:

The review phase involves evaluating system performance based on user feedback and analysis. Issues found during this phase are promptly addressed to ensure reliability.

Key activities in the **Review phase**:

- Evaluate system performance using feedback and analytics.
- Collect and analyze performance metrics and user feedback.
- Identify areas for improvement and apply updates or fixes.
- Ensure the system meets user expectations and performance goals.

Launch phase:

Finally, the launch phase ensures live deployment of the application and a support and maintenance plan is in place. Continuous monitoring and updates are performed to maintain performance and security.

Key activities in the **Launch phase**:

- Finalize deployment and transition to full production.
- Set up a support and maintenance plan for ongoing operations.
- Monitor system performance and security post-launch.
- Continuously update and improve the application based on user needs. Ensure customer satisfaction through responsive support.

3.2 Fact Gathering

A survey conducted 20 participants using a Google Form to understand their preferences and expectations for a blockchain-based voting system. The survey was designed to understand the demographics of users, their previous election experience, familiarity with blockchain technology, and their preferences for voting system features. Here are some questions that covered in the questionnaire:

- Have you participated in any university elections before.
- What kind of elections you have participated.
- Do you familiar with blockchain technology.
- What device do you most commonly use for online activities.
- What features do you find most important in a university voting system.
- What are your top concerns about voting online.

Google form (Questionnaire) link: https://forms.gle/incgVSmtwHm3fYAu6

3.3 Fact Recording

3.3.1 Section 1 - General Information

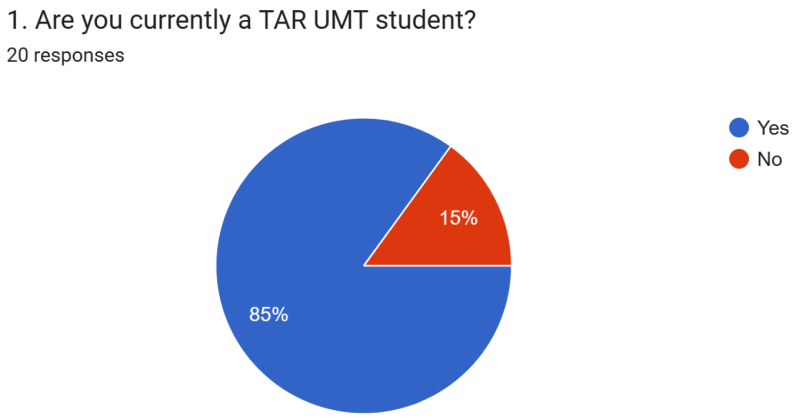


Figure 3.1. Are you currently a TAR UMT student

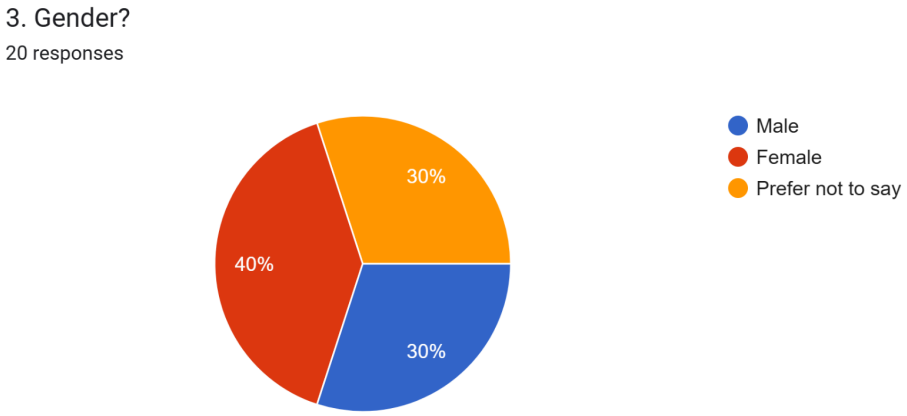


Figure 3.2. Gender

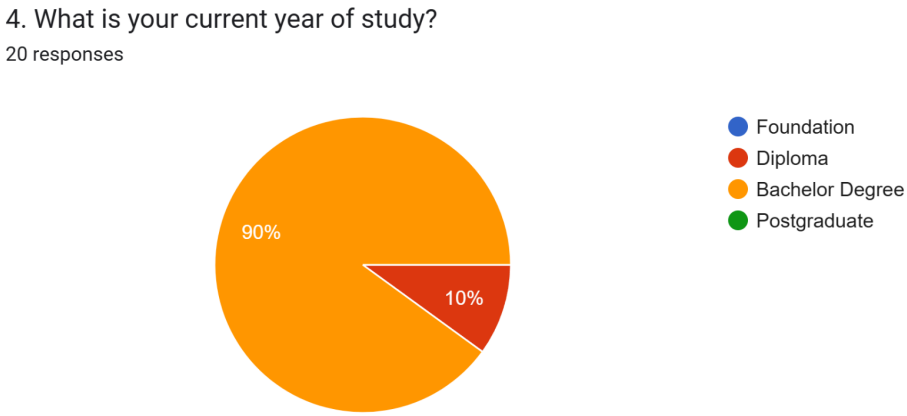


Figure 3.3. What is your current year of study

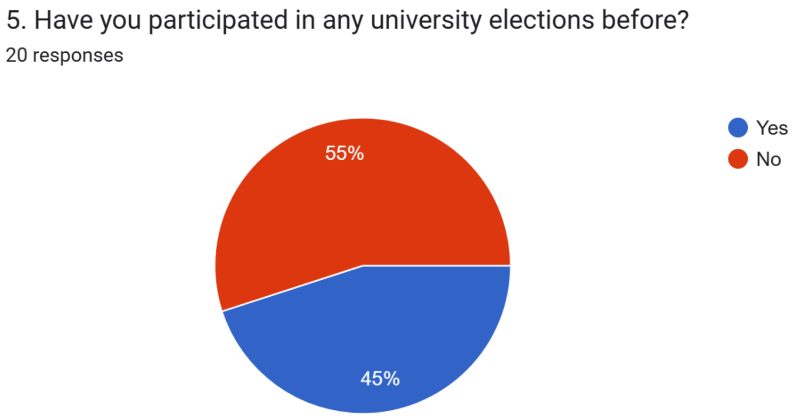


Figure 3.4. Have you participated in any university elections before

3.3.2 Section 2 - Election Part

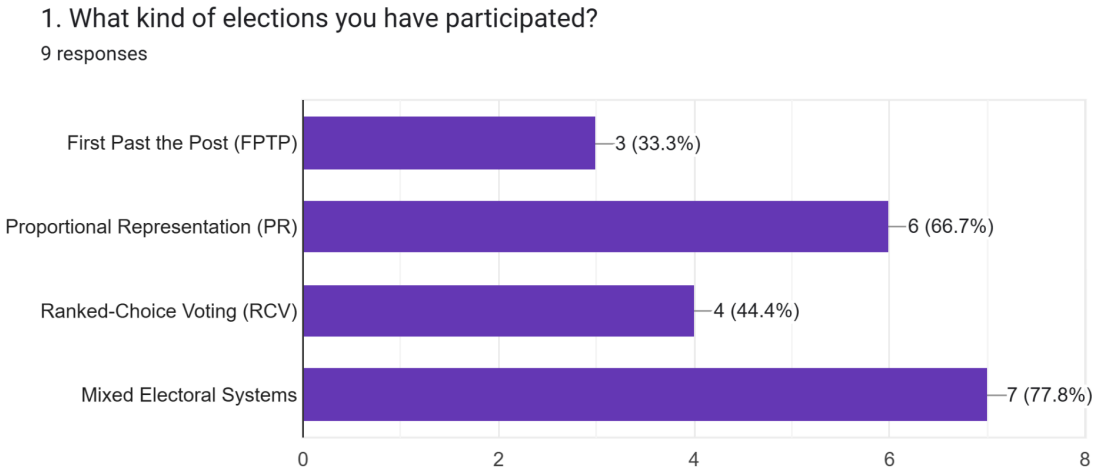


Figure 3.5. What kind of elections you have participated

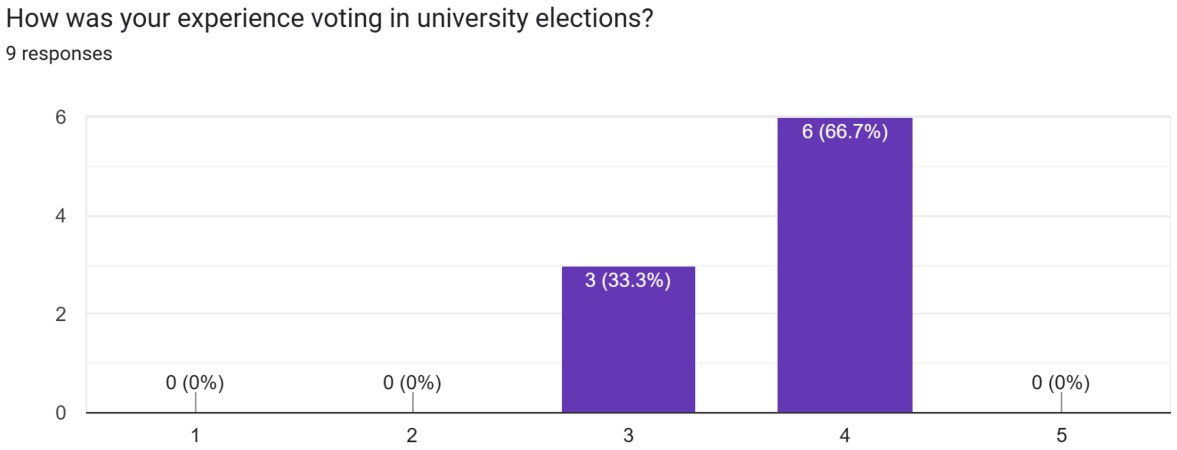


Figure 3.6. How was your experience voting in university elections

3.3.3 Section 3 - Technology Familiarity

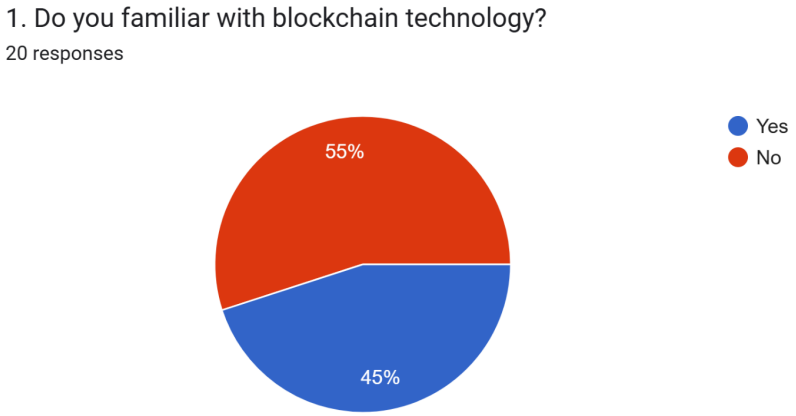


Figure 3.7. Do you familiar with blockchain technology

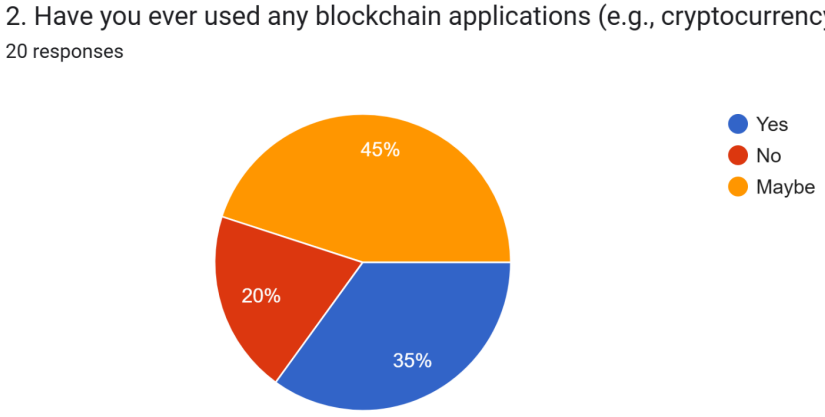


Figure 3.8. Have you ever used any blockchain applications (e.g., cryptocurrency wallets)?

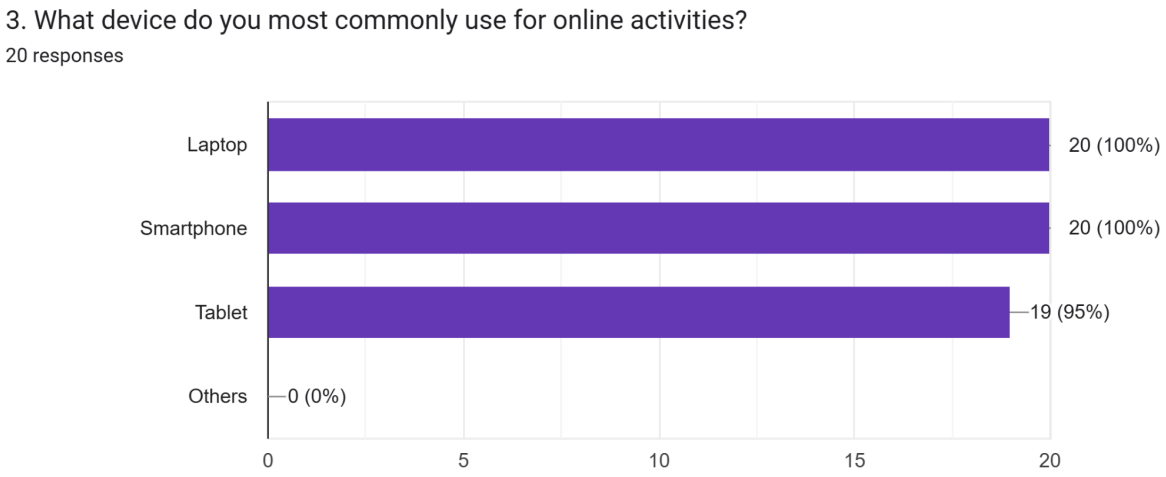


Figure 3.9. What device do you most commonly use for online activities

3.3.4 Section 4 - Voting System Preferences

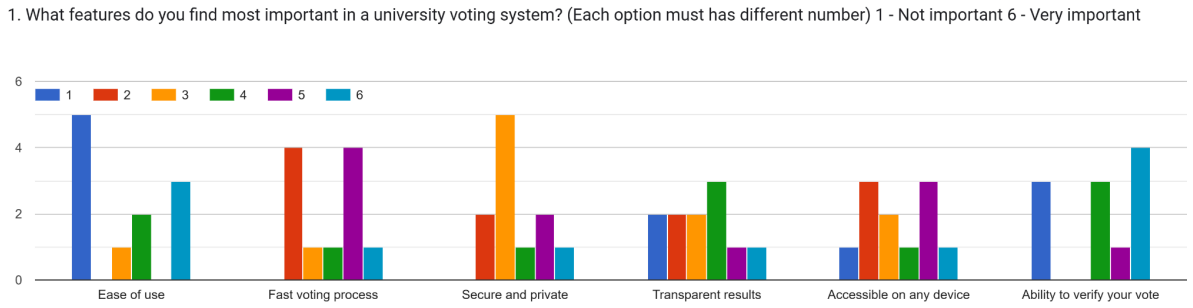


Figure 3.10. What features do you find most important in in a university voting system

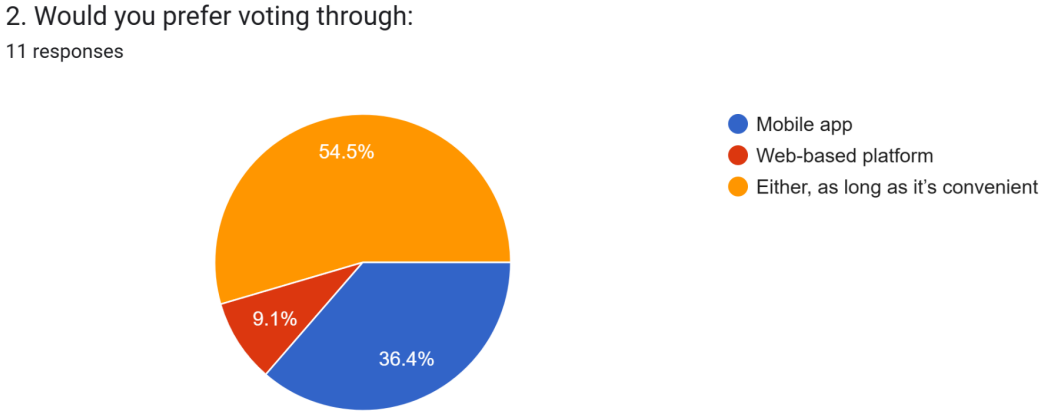


Figure 3.11. Would you prefer voting through

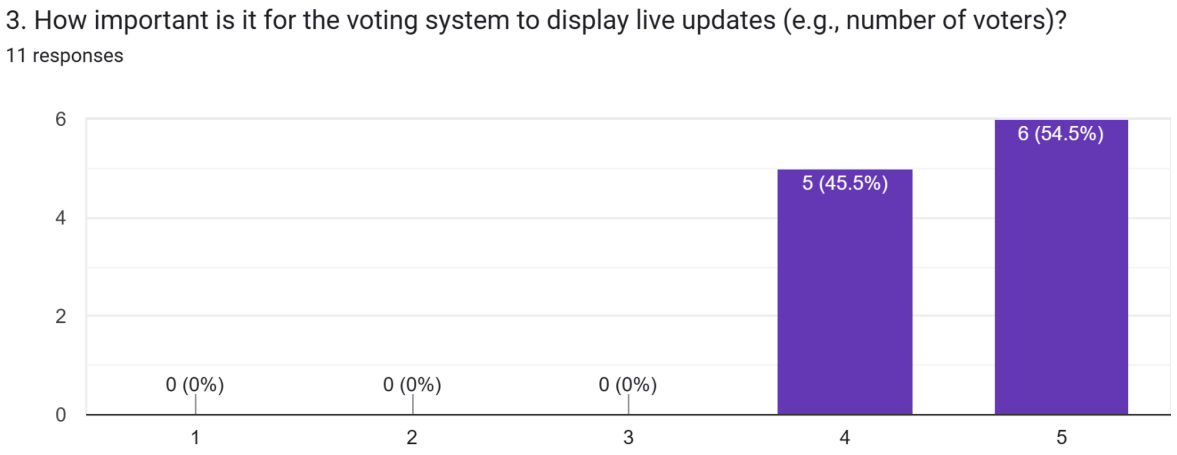


Figure 3.12. How important is it for the voting system to display live updates (e.g., number of voters)

3.3.5 Section 5 - Security and Transparency

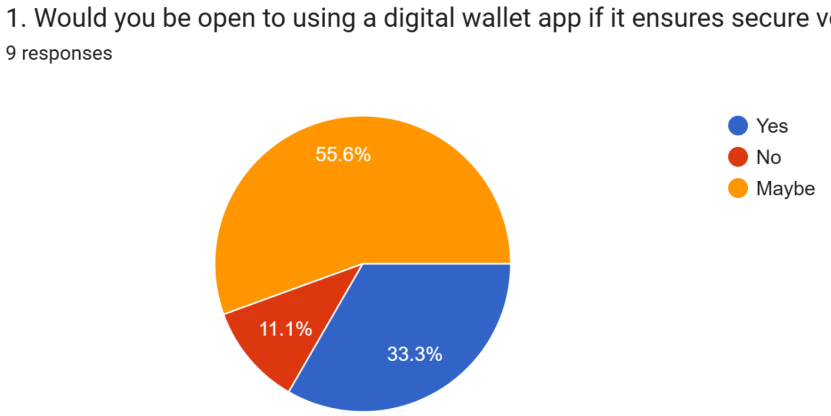


Figure 3.13. Would you be open to using a digital wallet app if it ensure secure voting

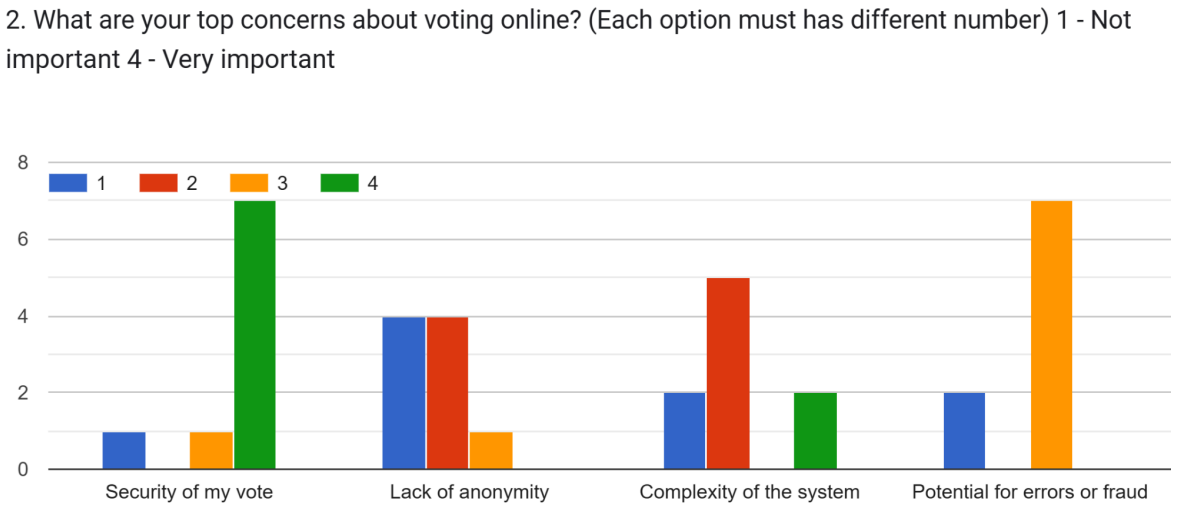


Figure 3.14. What are your top concerns about voting online

3.3.6 Section 6 - Feedback and Suggestions

3.4 Fact Analysis

3.4.1 Section 1 - General Information

1. **Are you currently a TAR UMT student?**
   Majority (85%) of respondents are TAR UMT students, indicating a high relevance of the survey results to the university community. The remaining 15% can provide external perspectives, broadening insights into general preferences.

2. **Gender?**
   Gender distribution is relatively balanced, with 30% male, 40% female, and 30% preferring not to disclose. Gender differences may not play a major role in influencing survey outcomes.

3. **What is your current year of study?**
   Most respondents (90%) are bachelor's degree students, reflecting perspectives primarily from this group. Only 10% are diploma-level students, while no postgraduate or foundation students participated.

4. **Have you participated in any university elections before?**
   A near-even split shows 45% have prior election experience, providing insights into experienced and inexperienced voters, which may impact preferences and expectations.

3.4.2 Section 2 - Election Part

1. **What kind of elections you have participated?**
   Mixed electoral systems (77.8%) and proportional representation (66.7%) are the most common types, highlighting people's familiarity with modern voting mechanisms. Ranked-choice voting (44.4%) and first past the post (33.3%) are less popular.

2. **How was your experience voting in university elections?**
   The majority of participants (66.7%) rated their experience a 4 (positive), indicating an overall satisfactory experience. No one rated it below a 3, indicating very low dissatisfaction.

3.4.3 Section 3 - Technology Familiarity

1. **Do you familiar with blockchain technology?**
   Just almost half (45%) are familiar with blockchain technology, while 55% are not. This suggests that educational resources are needed to increase understanding and adoption.

2. **Have you ever used any blockchain applications (e.g., cryptocurrency wallets)?**
   35% have experience, 45% are unsure, and 20% have never used a blockchain application. The results show that people have a moderate level of familiarity with blockchain applications, but highlight the need for user-friendly tools.

3. **What device do you most commonly use for online activities?**
   All participants used laptops and smartphones, with 95% using tablets. This confirms the importance of designing accessible solutions across platforms.

3.4.4 Section 4 - Voting System Preferences

1. **What features do you find most important in a university voting system?**
   - **Ease of use**: Rated 6 by 3 participants, highlighting the need for a user-friendly interface.
   - **Fast voting process**: Mixed responses with emphasis on moderate importance (rating 3-5), suggesting efficiency is appreciated but not prioritized.
   - **Secure and private**: Rated 6 by 1 participant and 5 by 2 participants, emphasizing security and privacy as top priorities.
   - **Transparent results**: Responses were spread across the scale, indicating some interest but less priority compared to security.
   - **Accessible on any device**: Moderate ratings suggest multi-device accessibility is important but secondary.
   - **Ability to verify votes**: Rated 6 by 4 participants, showing strong demand for verifiability.

2. **Would you prefer voting through:**
   Most respondents were open to different platforms (54.5%) as long as that platform gave them comfort or mobile apps (36.4%), indicating that voting platform was not a priority.

3. **How important is it for the voting system to display live updates (e.g., number of voters)?**
   A combined 100% rated this feature as 4 or 5, highlighting a strong demand for transparency and real-time updates.

3.4.5 Section 5 - Security and Transparency

1. **Would you be open to using a digital wallet app if it ensures secure voting?**
   55.6% said maybe, 33.3% said yes, and 11.1% said no. While most respondents supported or were skeptical of the idea, further user education could increase acceptance.

2. **What are your top concerns about voting online?**
   - **Security**: Rated 4 by 7 participants, indicating it as the primary concern.
   - **Complexity**: Rated 2 by 5 participants, showing usability concerns must be addressed.
   - **Lack of anonymity**: Mixed ratings, suggesting divided opinions.
   - **Errors or fraud**: Rated 3 by 7 participants, reinforcing the importance of error prevention.

   Security and fraud prevention are the top concerns requiring focused solutions.

3.4.6 Section 6 - Feedback and Suggestions

1. **What improvements would you like to see in the current voting process?**
   Suggestions included clearer instructions, online voting options, flexible/extended voting hours, legal protections and better security. These all demonstrate people's desire for modernity, safety and convenience.

2. **If you had the opportunity, would you like to try a blockchain-based voting system?**
   All participants (100%) expressed interest, confirming strong acceptance and enthusiasm for the proposed solution.

3.5 Requirements Analysis

3.5.1 Development Environment

Component	Description
Languages	Dart (Flutter) for building mobile applications and Solidity for developing smart contracts.
Frameworks	Flutter is used for creating cross-platform mobile applications and React.js for web apps.
Blockchain Tools	Remix for deploying smart contract and MetaMask is a wallet for interacting with Ethereum.
Database	Firebase is used for storing basic user data, ensuring scalability and real-time updates.

3.5.2 Operational Environment

Component	Description
Platforms	Supports Android, iOS and web browsers to provide multi-platform accessibility.
Blockchain	Ethereum testnet is used during development for testing smart contracts and blockchain features.
Devices	Includes one laptop for development and physical smartphones for testing mobile compatibility.
Operational Fees	Includes salary costs for one developer at 4500 MYR per month to cover development and support activities.

3.5.3 Non-functional Requirement

1. Authentication Module

- The authentication process should complete within 2 seconds under normal network conditions. (**Performance**)
- The login and Metamask integration should be user-friendly and provide clear error messages. (**Usability**)
- Authentication services should be available for up to 99.99% time. (**Reliability**)
- All user credentials and authentication data should be encrypted securely by using the industry standard (SHA-256). (**Security**)
- The system should handle a large number of concurrent logins during peak times without performance degradation. (**Scalability**)

2. User Management Module

- Only authorized personnel (admin) can make changes to the user roles or data. (**Security**)
- Modifying and getting user information should take no longer than 3 seconds under normal conditions. (**Performance**)
- The system should ensure that user data is consistently available and up-to-date with a maximum of 99.9% availability. (**Reliability**)
- User interface interactions should be simple and responsive for ease of use. (**Usability**)

3. Voting Event Management Module

- Creating and modifying a voting event should not take more than 5 seconds. (**Performance**)
- Events should prevent unauthorized access or tampering by securing data via blockchain immutability. (**Security**)
- The system should scale to support multiple simultaneous voting events without performance degradation. (**Scalability**)
- Event management features should be intuitive and require minimal training. (**Usability**)

4. Blockchain Integration Module

- Transactions to and from blockchain must be processed within 10 seconds under normal network conditions. (**Performance**)
- Blockchain data should be immutable and tamper-proof to ensure data integrity. (**Security**)
- The system should support compatibility with Ethereum networks and Metamask wallets. (**Compatibility**)
- Blockchain interactions should be scalable to support high transaction volumes. (**Scalability**)

5. Voting Interface Module

- The voting interface should load and display data within 3 seconds. (**Performance**)
- The voting process should be secured against unauthorized access. (**Security**)

- The interface should be designed to support mobile and web responsiveness. (**Usability**)
- Voters should receive real-time feedback upon submitting their votes. (**Reliability**)

6. Voting Results Module

- The results should be immutable and verifiable via blockchain. (**Security**)
- The system should support exporting results in multiple formats. (**Compatibility**)
- The results should be accessible to authorized users only. (**Security**)
- Results should be available in real-time without delays. (**Performance**)

7. Reporting Module

- Reports should be exportable in multiple formats. (**Compatibility**)
- The reporting interface should be user-friendly and customizable. (**Usability**)

8. Notifications Module

- Notifications should be encrypted for secure delivery. (**Security**)
- The system should support multiple notification channels (email, push). (**Compatibility**)
- Users should be able to configure notification preference easily. (**Usability**)
- Notifications should be delivered within 5 seconds after an event is triggered. (**Performance**)

9. Dashboard Module

- The dashboard should load data within 3 seconds under normal conditions. (**Performance**)
- Dashboard components should be interactive and responsive. (**Usability**)
- Data displayed on the dashboard should always be up-to-date. (**Reliability**)
- The dashboard should scale to handle a high number of concurrent users. (**Scalability**)

3.5.4 Functional Requirement

1. Authentication Module

Admin:

1. The system should allow the admin to log in to the system using a username and password for secure access.
2. The system should allow the admin to log in using a MetaMask account to perform blockchain-related operations.
3. The system should allow the admin to authenticate using MetaMask for blockchain-related operations, including smart contract management.
4. The system should allow admin to logout the account from the system.

University Staff:

1. The system should allow university staff to log in using university-issued credentials (username and password) for secure access.
2. The system should allow university staff to log in using a MetaMask account to connect their wallet and manage voting events.
3. The system should allow university staff to connect their MetaMask wallet to manage voting events and interact with the blockchain.
4. The system should allow university staff to reset their own password if forgotten to regain access.
5. The system should allow university staffs to logout their account from the system.

University Student:

1. The system should allow university students to register for an account using a username and password before logging in.
2. The system should allow university students to log in to their account using MetaMask for blockchain transactions and voting events.
3. The system should allow university students to log in using credentials (username and password) for accessing their voting status and profile.
4. The system should allow university students to recover their password via email or mobile if forgotten.
5. The system should allow university students to connect their MetaMask wallet to participate in voting events and interact with the blockchain.
6. The system should allow university students to logout their account from the system.

2. User Management Module

Admin:

1. The system should allow the admin to add new university staff and students to the system.
2. The system should allow the admin to edit or delete user accounts (e.g., staff or students).
3. The system should allow the admin to assign or update roles for users (e.g., staff to admin).

University Staff:

1. The system should allow university staff to manage student account statuses (activate, deactivate, or suspend).
2. The system should allow university staff to edit their own account information, such as personal details or passwords in their profile.
3. The system should allow university staff to assign roles to students for voting eligibility.
4. The system should allow university staff to add new students to the system.

University Student:

1. The system should allow university students to edit personal information in their profile.

3. *Voting Event Management Module*

Admin:

1. The system should allow the admin to approve or reject voting events created by university staff.
2. The system should allow the admin to create new voting events including specifying voting criteria and timelines.
3. The system should allow the admin to schedule voting event timelines.
4. The system should allow the admin to cancel or reschedule voting events if necessary.
5. The system should allow the admin to add candidates to the voting events they create.
6. The system should allow the admin to edit candidates details in the voting events.
7. The system should allow the admin to remove candidates from voting events they create.
8. The system should allow admin to monitor the progress of ongoing voting events.

University Staff:

1. The system should allow university staff to create new voting events including specifying voting criteria and timelines.
2. The system should allow university staff to edit or cancel voting events.
3. The system should allow university staff to monitor the progress of ongoing voting events.
4. The system should allow university staff to add candidates to the voting events they create.
5. The system should allow university staff to edit candidates details in the voting events.
6. The system should allow university staff to remove candidates from voting events they create.

University Student:

1. The system should allow university students to view a list of voting events they are eligible to participate in.
2. The system should allow university students to register their interest in upcoming voting events.

4. *Blockchain Integration Module*

Admin:

1. The system should allow admin to trigger Blockchain transactions for creating voting events.

University Staff:

1. The system should allow university staff to trigger Blockchain transactions for creating voting events.

University Student:

1. The system should allow university student to trigger Blockchain transactions for performing vote submission.
2. The system should allow university students to submit votes securely via blockchain.

5. *Voting Interface Module*

Admin:

1. The system should allow admin to view a live voting dashboard showing progress and participation statistics.

University Staff:

1. The system should allow university staff to view a live voting dashboard showing progress and participation statistics.
2. The system should allow university staff to customize the visibility of certain voting options.

University Student:

1. The system should allow university students to access the voting interface to view voting options.
2. The system should allow university students to vote and confirm their vote before submission.

6. Voting Results Module

Admin:

1. The system should allow admin to view the results of the voting events they participated in after they conclude.
2. The system should allow admin to export voting results for reporting.

University Staff:

1. The system should allow university staffs to view the results of the voting events they participated in after they conclude.
2. The system should allow university staff to export voting results for reporting.

University Student:

1. The system should allow university students to view the results of the voting events they participated in after they conclude.

7. Reporting Module

Admin:

1. The system should allow the admin to generate summary reports for specific voting events.

University Staff:

1. The system should allow university staff to generate summary reports for specific voting events.

8. Notifications Module

Admin:

1. The system should allow the admin to configure global notifications for all users, such as system maintenance notifications and update notifications.
2. The system should allow admin to receive notifications about the system maintenance and updates with the system.
3. The system should admin to set automated reminders for important notifications.
4. The system should allow admin to enable or disable notifications.

University Staff:

1. The system should allow university staff to send notifications to students regarding upcoming or ongoing voting events.

2. The system should allow university staff to receive notifications from admin about the system maintenance and updates with the system.
3. The system should allow university staff to set automated reminders for voting deadlines.
4. The system should allow university staff to enable or disable specific types of notifications.

University Student:

1. The system should allow university students to receive notifications about their voting eligibility and event schedules.
2. The system should allow university students to enable or disable specific types of notifications.

9. Dashboard Module

Admin:

1. The system should allow the admin to view an overview of all system activities, such as the number of active users and ongoing voting events.
2. The system should allow the admin to restrict module access for specific users or user roles.
3. The system should allow the admin to remove access restriction for specific modules, restoring access to users or user roles.
4. The system should allow the admin to view all summary of the voting events.

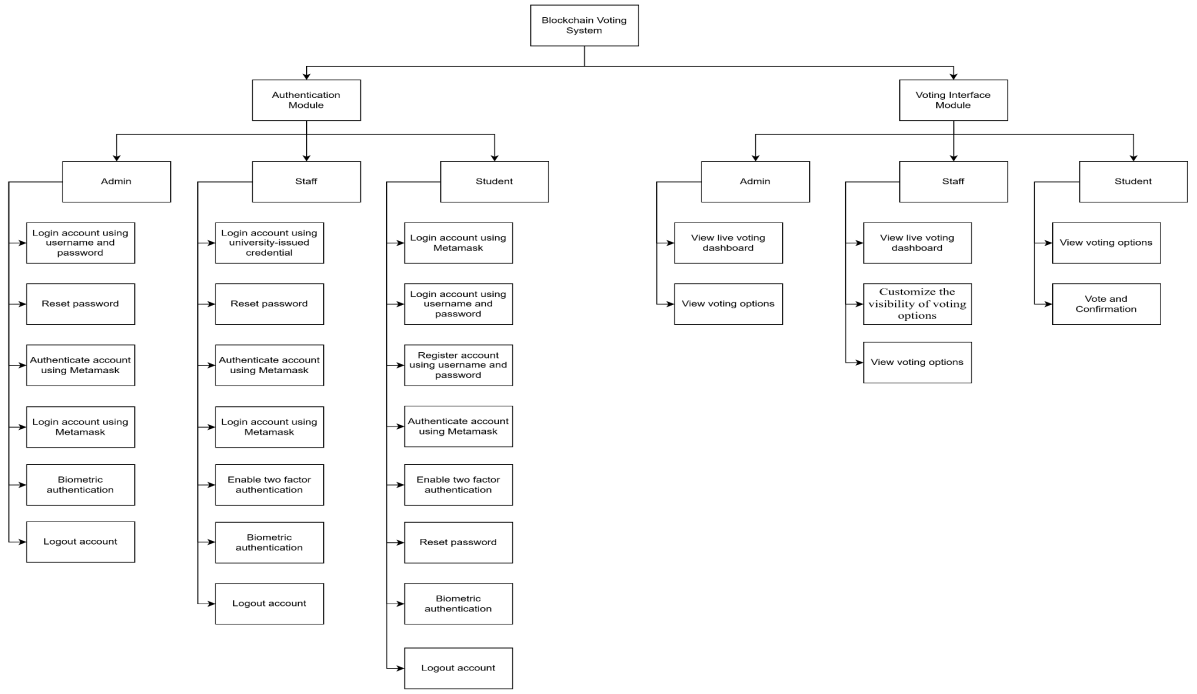
University Staff:

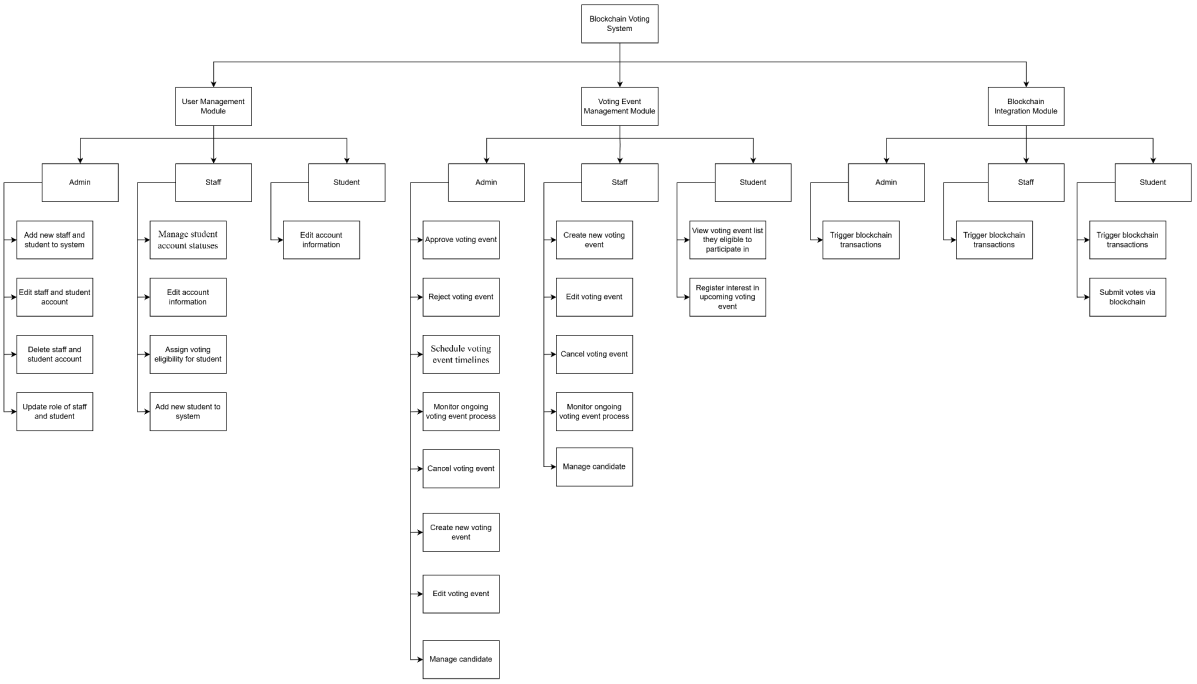
1. The system should allow university staff to view a summary of the voting events they manage.

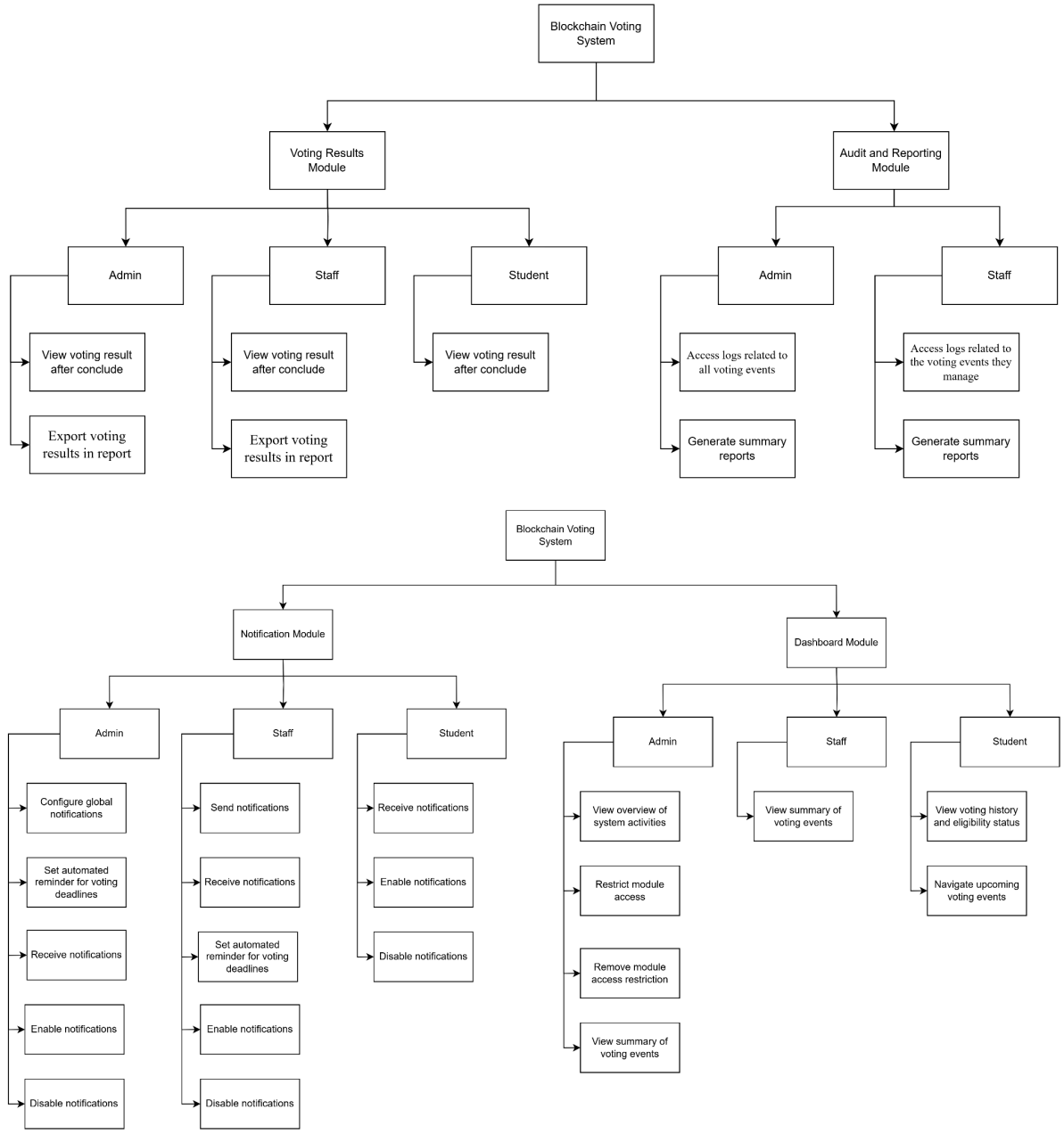
University Student:

1. The system should allow university students to view their voting history and eligibility status.
2. The system should allow university students to navigate upcoming voting events directly from the dashboard.

3.6 Requirements Diagram (Module Diagram)







3.7 Chapter Summary and Evaluation

This chapter outlines the methodology and requirements analysis for a blockchain-based voting system, using an agile approach to enable flexibility and iterative development. Key phases include:

Planning - research, stakeholder interviews, and feature prioritization.

Design - system architecture, database schemas, and models.

Develop - smart contract implementation, UI development, and feature integration.

Test - unit, integration, and system testing.

Review and Launch - performance evaluation, deployment, and maintenance plans.

A survey of 20 participants revealed priorities such as security, usability, transparency, and real-time updates. The requirements analysis detailed the functional and non-functional requirements of the module, focusing on scalability, security, and usability.

It effectively defined a structured approach and comprehensive requirements. It ensured alignment with user expectations and system goals. Recommendations include adding risk analysis, requirements validation, and performance benchmarks for further refinement.

Chapter 4

System Design

4 System Design

This provides an overview of the system's architecture, covering key modules, UML diagrams, data design, UI layouts, and software architecture. It outlines functionality, workflows, and user interactions to guide system implementation and evaluation.

4.1 System Design

4.1.1 Authentication Module

The Authentication Module ensures that only authorized users can access the system and perform actions based on their roles, while also enhancing security through user authentication and blockchain integration.

Admin:

- **Login using Metamask account:** Admin can log in using their MetaMask account to perform blockchain-related operations.
- **Login using credentials:** Admin can log in using their username and password for secure access to the system.
- **MetaMask Authentication:** Admin can authenticate using MetaMask for blockchain-related operations, allowing them to interact with smart contracts and manage blockchain voting tasks.
- **Logout account:** Admin can logout the account from the system.

University Staff:

- **Login using Metamask account:** University staff can log in using their MetaMask account to connect their wallet and manage voting events, enabling blockchain interactions.
- **Login using credentials:** University staff can log in using their university-issued credentials (username and password), ensuring access to their respective features for managing events and users.
- **MetaMask Integration:** University staff can connect their MetaMask wallet to manage voting events and interact with the blockchain.
- **Password Recovery:** University staff can reset their passwords if forgotten to regain access to their accounts.
- **Logout account:** University staffs can logout their account from the system.

University Student:

- **Login using Metamask account:** University students can log in using their MetaMask account, which integrates their wallet for blockchain transactions, allowing them to vote in elections.
- **Login using credentials:** University students can use their university credentials (e.g., student ID and password) to log in to the system and access their voting status and profile.

- **Registration with username and password:** University students can register for the system using a username and password, creating an account before logging in.
- **Password Recovery:** University students can recover their passwords via email or mobile, ensuring account recovery if they forget their login information.
- **MetaMask Integration**: University student can connect their MetaMask wallet to participate in voting events and interact with the blockchain.
- **Logout account:** University students can logout their account from the system.

4.1.2 User Management Module

The User Management Module allows for the management of all user accounts in the system, ensuring proper access control and role assignment.

Admin:

- **User Addition and Management**: Admin can add new university staff and students to the system, ensuring that users can access the platform.
- **User Editing/Deletion**: Admin have the ability to modify or remove users if necessary, ensuring that outdated or inactive accounts are properly handled.
- **Role Assignment**: Admin can assign or update roles, such as promoting a staff member to admin or assigning new roles for other users.

University Staff:

- **Student Account Status Management**: University staff can manage student accounts, enabling, disabling, or suspending accounts as necessary.
- **Profile Management**: University staff can update their personal details and reset their passwords, ensuring their information is accurate and secure.
- **Role Assignment for Students**: University staff can assign roles to students, such as confirming their eligibility for voting in specific events.
- **Student Addition**: University staff can add new students to the system, providing them with access to voting events and other features.

University Student:

- **Profile Management**: Students can update their personal details and view or modify their profiles as needed.

4.1.3 Voting Event Module

The Voting Event Management Module enables the creation, editing and management of voting events, ensuring smooth and controlled elections.

Admin:

- **Event Approval**: Admin can approve or reject voting events created by staff to ensure they meet the necessary criteria.
- **Event Creation**: Admin can create new voting events, specifying the voting criteria, timelines and other key event details.

- **Event Scheduling**: Admin can schedule voting events, ensuring that they are held within predefined timelines.
- **Event Modification**: Admin can cancel or reschedule voting events if necessary, accommodating changes in event planning.
- **Candidate Management**: Admin can add, edit or remove candidates for voting events they manage, ensuring that candidate lists are up to date.
- **Event Monitoring**: Admin can monitor the progress of voting events including participation and election status.

University Staff:

- **Event Creation**: Staff can create voting events, setting timelines and criteria for eligible voters.
- **Event Modification**: Staff can edit or cancel events as needed, ensuring that changes are communicated effectively.
- **Event Monitoring**: Staff can monitor the progress of voting events including participation and election status.
- **Candidate Management**: Staff can manage candidates by adding, editing, or removing them from the event.

University Student:

- **View Events**: Students can view a list of voting events they are eligible to participate in, based on their role and eligibility.
- **Event Registration**: Students can register their interest in upcoming events to stay informed about their participation.

4.1.4 Blockchain Integration Module

This module handles the interaction between the application and the blockchain, ensuring that voting transactions are securely processed.

Admin:

- **Blockchain Transaction Triggering**: Admin can initiate blockchain transactions for creating and managing voting events, ensuring that blockchain-based voting processes are correctly recorded.

University Staff:

- **Blockchain Transaction Triggering**: University staff can initiate blockchain transactions for creating and managing voting events, ensuring that blockchain-based voting processes are correctly recorded.

University Student:

- **Blockchain Transaction Triggering**: University student can initiate blockchain transactions for vote submission, ensuring that blockchain-based voting processes are correctly recorded.

- **Vote Submission**: University students can submit their votes via the blockchain, ensuring their vote is securely recorded and immutable.

4.1.5 Voting Interface Module

The Voting Interface Module provides an easy-to-use interface for voters to interact with during elections.

Admin:

- **Voting Dashboard**: Admin can access a live voting dashboard to view real-time statistics, progress and participation rates for ongoing voting events.
- **Voting Interface**: Admin can access a user-friendly interface to view voting options and candidates in the events.

University Staff:

- **Voting Dashboard**: University staff can view the voting dashboard to monitor event progress and participation, helping them to ensure the election is proceeding smoothly.
- **Customization**: University staff can customize the visibility of certain voting options, ensuring that the voting interface is appropriate for different roles or events.
- **Voting Interface**: University staff can access a user-friendly interface to view voting options and candidates in the events.

University Student:

- **Voting Interface**: University students can access a user-friendly interface to view voting options and candidates in the events they are eligible to participate in.
- **Vote and Confirmation**: University students can vote candidate and confirm their vote before submission to ensure accuracy and intentional voting.

4.1.6 Voting Result Module

This module allows for the real-time and post-event viewing of voting results, ensuring transparency and accountability.

Admin:

- **View Results**: Admin can view the results of the voting events they participated in, once the event concludes.
- **Export Results**: Admin can export results for reporting and record-keeping.

University Staff:

- **View Results**: Admin can view the results of the voting events they participated in, once the event concludes.
- **Export Results**: University staff can export results for reporting and record-keeping.

University Student:

- **View Results**: University student can view the results of the voting events they participated in, once the event concludes.

4.1.7 Reporting Module

The Reporting Module ensures reports can be generated for transparency and future reference.

Admin:

- **Summary Reports**: Admin can generate detailed summary reports for specific events to analyze voting trends and results.

University Staff:

- **Event Reports**: University staff can generate summary reports for the voting events they oversee, helping them to manage event data efficiently.

4.1.8 Notifications Module

This module ensures that users receive timely updates relevant to their role.

Admin:

- **Global Notifications**: Admin can configure global notifications for all users, such as notifications for system maintenance, updates with the system or important announcements.
- **Automated Reminders**: Admin can set up automated reminders for important notifications on time.
- **Receive Notifications**: Admin can receive notifications about the system maintenance notifications or important system updates notifications through global notifications.
- **Notification Customization**: Admin can enable or disable specific types of notifications, allowing for personalized notification preferences.

University Staff:

- **Student Notifications**: University staff can send notifications to students regarding voting events, ensuring they stay informed about upcoming deadlines or participation details.
- **Receive Notifications**: University staff can receive notifications about the system maintenance notifications or important system updates notifications from admin through global notifications.
- **Automated Reminders**: University staff can set up automated reminders for voting deadlines, ensuring students are prompted to vote on time.
- **Notification Customization**: University staff can enable or disable specific types of notifications, allowing for personalized notification preferences.

University Student:

- **Event Notifications**: Students can receive notifications about their eligibility and event schedules, ensuring they don't miss important voting opportunities and the system maintenance or updates notifications.
- **Notification Customization**: Students can enable or disable specific types of notifications, allowing for personalized notification preferences.

4.1.9 Dashboard Module

The Dashboard Module offers an overview of system activities and allows for easy navigation between various modules.

Admin:

- **System Overview**: Admin can view an overview of all system activities, such as active users and ongoing voting events.
- **Restrict Module Access**: Admin can restrict access to specific modules for users or user roles, ensuring that only authorized individuals can access sensitive system functionality.
- **Remove Module Access Restriction:** Admin can remove previously applied module access restrictions, allowing users or user roles to regain access to modules that were previously restricted.
- **Event Overview**: Admin can view all summary of the voting event including details about event status, participation rates and timelines.

University Staff:

- **Event Overview**: Staff can view a summary of the voting events they manage, helping them stay organized and informed about their responsibilities.

University Student:

- **Voting History and Eligibility**: Students can view their voting history and eligibility status from the dashboard.
- **Event Navigation**: Students can easily navigate to upcoming voting events directly from the dashboard.

4.2 UML Diagram

4.2.1 Use Case Diagram

1. *Overall Use Case Diagram*

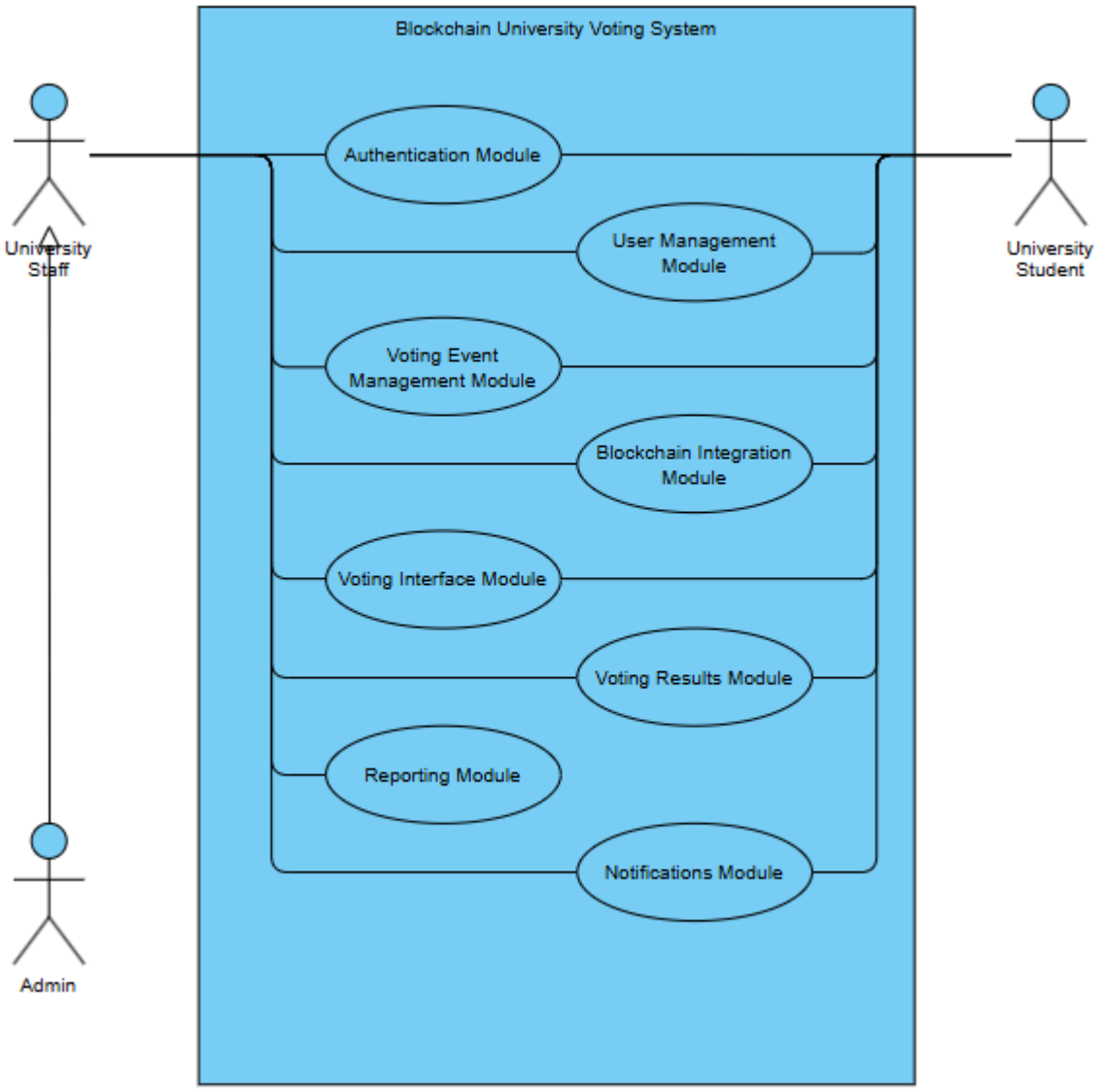


Figure 4.1. Blockchain University Voting System

2. *Authentication Module*

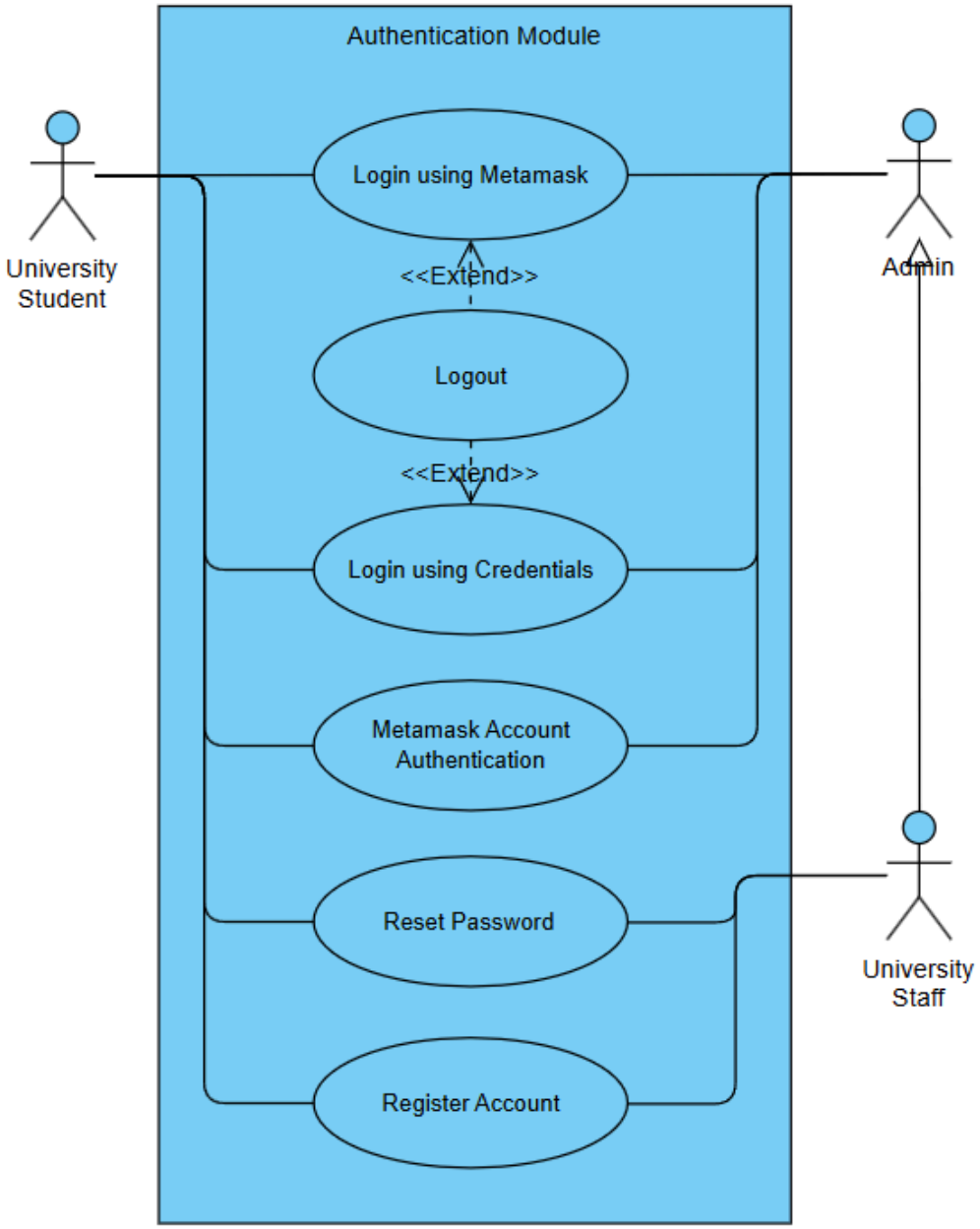


Figure 4.2. Authentication Module

Use case name: Login using credentials	
Actor: Admin, University Staff, University Student	
Brief description: The system should allow the user to login their account using credentials (username, password).	
Pre-condition: N/A	
Main flow of Events:	
Actor Action	System Response

1. Open application.	2. Load the login page and display.
3. Fill in the username and password. 4. Click 'Login' button.	5. Verify the account' credentials in the system. 6. Send verification code to email. 7. Display verification code page.
8. Fill in the verification code (email). 9. Click 'Verify' button.	10. Verify the code. 11. Display success message and redirects to dashboard page (home page).
Alternative Flow of Events: **A1 Step 5:** If the credentials is unmatch with the data or doesn't exist in the system or the system fails to connect to the system due to network issues, it will display an error message and return to step 3. **A2 Step 5:** If the user account does not enable two-factor authentication, it will redirects to step 11. **A3 Step 10:** If the verification code is wrong with the code that system sent to the email, it will display an error message and return to step 8.	
Post-condition: The user has login to the account using credentials successfully.	

Use case name: Login using Metamask	
Actor: Admin, University Staff, University Student	
Brief description: The system should the user to login their account using Metamask.	
Pre-condition: The user account wanted to login has authenticated with Metamask account.	
Main flow of Events:	
Actor Action	System Response
1. Open application.	2. Load the login page and display.
3. Click 'Connect Wallet' button.	4. Display modal view of the wallet connection.
5. Select 'Metamask' tab in the modal view.	6. Open Metamask app. 7. Display confirmation message.
8. Click 'Connect' button.	9. Connect to the Metamask account. 10. Display success message and redirects to dashboard page (home page).
Alternative Flow of Events: **A1 Step 4:** If the system fails to connect to the modal view due to network issues, it will display an error	

message and return to step 3.

A2 Step 6:
If the device does not has the Metamask application, it will display an error message and return to step 3.

A3 Step 8:
If the user cancelled the connection, it will return to step 3.

A4 Step 9:
If the system fails to connect to Metamask account due to network issues, it will display an error message and return to step 3.

Post-condition: The user has login to the system using Metamask account successfully.

Use case name: Metamask account authentication

Actor: Admin, University Staff, University Student

Brief description: The system should allow the user to authenticate their account with Metamask.

Pre-condition:
- The user must be logged in to the system.
- The user account has not authenticate before with Metamask.

Main flow of Events:

Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select 'Profile' option located at the bottom navigation bar.	4. Display the current user account information.
5. Click 'Authenticate account with Metamask' button.	6. Display modal view of the wallet connection.
7. Select 'Metamask' tab in the modal view.	8. Open Metamask app. 9. Display confirmation message.
10. Click 'Connect' button.	11. Connect the user account with the Metamask account. 12. Display success message.

Alternative Flow of Events:
A1 Step 4:
If the system fails to load the current user information due to network issues, it will display an error message and return to step 3
A2 Step 6:
If the system fails to connect to the modal view due to network issues, it will display an error message and return to step 5.

A3 Step 8:
If the device does not has the Metamask application, it will display an error message and return to step 5.

A4 Step 10:
If the user cancelled the connection, it will return to step 5.

A5 Step 11:
If the system fails to connect to Metamask account due to network issues, it will display an error message and return to step 5.

Post-condition: The user account is connected and authenticated with Metamask account.

Use case name: Reset password	
Actor: University Staff, University Student	
Brief description: The system should allow the user to reset their account password if they forget it.	
Pre-condition: N/A	
Main flow of Events:	
Actor Action	System Response
1. Open application.	2. Load the login page and display.
3. Click 'Forgot password' text.	4. Display reset password page.
5. Enters username or email. 6. Click 'Request Verification Code'.	7. Check is the username or email exists in the system. 8. Send verification code to the email that binded with the username. 9. Display new password and verification code request in a page.
10. Enters new password and verification code. 11. Click 'Reset password'.	12. Validate the password with the password requirements. 13. Verify the verification code. 14. Display success message and redirects to login page.
Alternative Flow of Events: **A1 Step 7:** If the username does not exists in the system, it will display an error message and return to step 5. **A2 Step 12:** If the password does not match the requirements, it will display an error message and return to step 10.	

A3 Step 13:
If the verification code does not match with the verification code that system sent to the email, it will display an error message and return to step 10.

Post-condition: The new password has set successfully for the user account.

Use case name: Register account

Actor: University Student

Brief description: The system should allow the university student to register account in the system under their university.

Pre-condition: N/A

Main flow of Events:

Actor Action	System Response
1. Open application.	2. Load the login page and display.
3. Click 'Doesn't have an account? Register here'	4. Display register page with new university student form.
5. Fill in the information to the new university student form and select the university.	6. Validates the information in the form with syntax and logic checking.
7. Click 'Register' button.	8. Validate the form information. 9. Send verification code to the new university student email.
10. Enters the verification code.	11. Verify the verification code. 12. Create the new university student account with unverified status under the university selected. 13. Display success message and redirects to login page.

Alternative Flow of Events:
A1 Step 6:
If the parameters in the form detected syntax or logic errors, it will display an error message and return to step 5.

A2 Step 11:
If the verification code does not match with the verification code that system sent to the email, it will display an error message and return to step 10.

A3 Step 12: If the new university student account is repeated in the system or it was suspend or deactivate before or fails to create the account due to network issues, it will display an error message and return to step 5.
Post-condition: The new university student account created with unverified status successfully in the system.

Use case name: Logout	
Actor: Admin, University Staff, University Student	
Brief description: The system should allow the user to logout their account.	
Pre-condition: The user must be logged in to the system.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Click ‘Settings’ option located at the bottom navigation bar.	4. Display settings page.
5. Click ‘Logout Account’ tab.	6. Display confirmation message.
7. Click ‘Confirm’ button.	8. Logout the account. 9. Display success message and redirects to login page.
Alternative Flow of Events: **A1 Step 7:** If the user cancelled the confirmation, it will return to step 5.	
Post-condition: The user has logout their account successfully.	

3. *User Management Module*

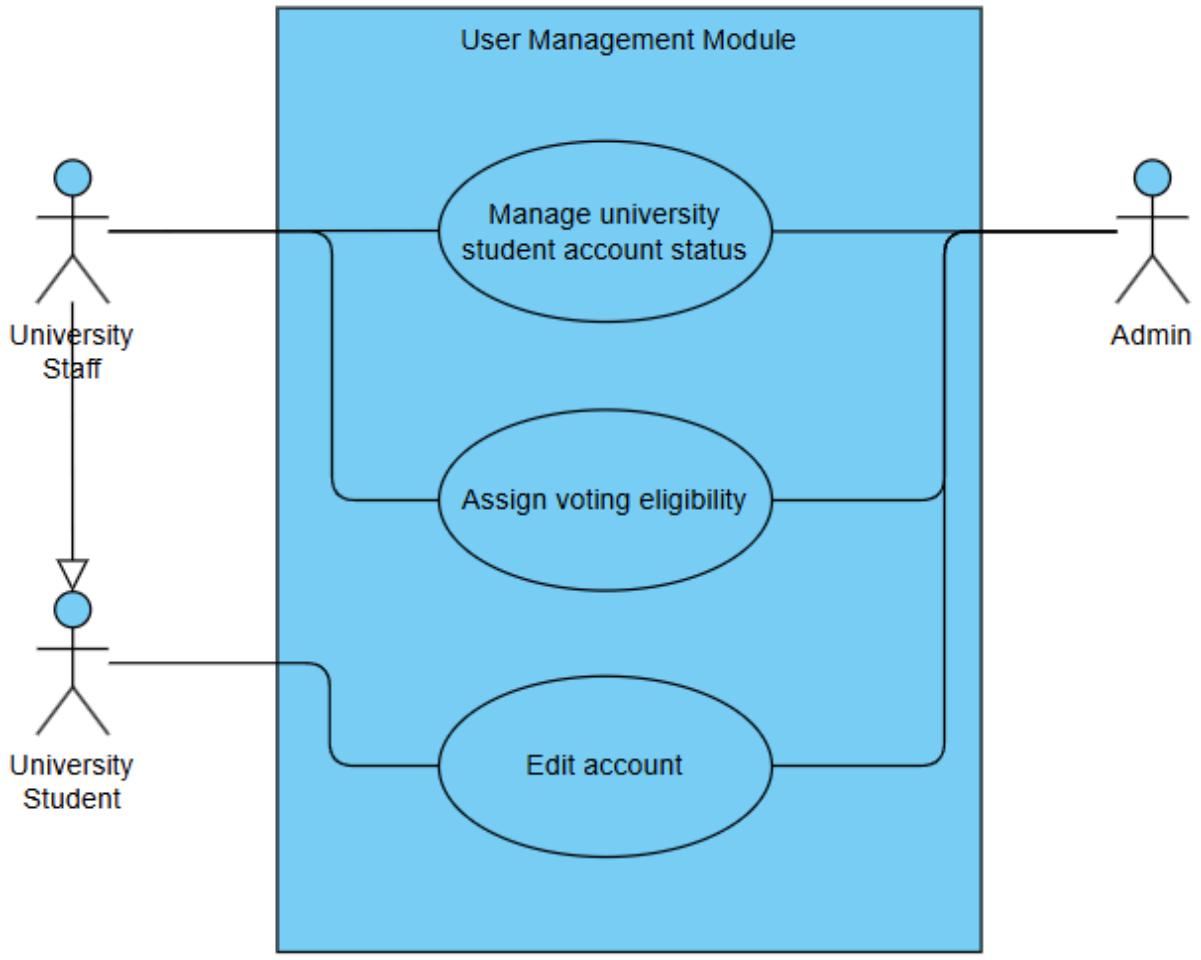


Figure 4.3. User Management Module

Use case name: Manage university student account status	
Actor: University Staff	
Brief description: The system should allow university staff to manage the account status of university student such as active, freezed.	
Pre-condition: The university staff must be logged into the system.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select user management section.	4. Display the university staffs and

	students in the system with sorted list and section.
5. Select a university student.	6. Display the university student information page.
7. Click 'Edit Status' button.	8. Display a dialog with multiple choice option to update the university student account status.
9. Select the new status. 10. Click 'Confirm' button.	11. Update the university student account status with the new status selected. 12. Display success message.
Alternative Flow of Events: **A1 Step 4:** If the system fails to load the user list due to network issues, it will display an error message and return to step 3. **A2 Step 11:** If the selected status is same with the current university student status or the system cannot update the university student account status with new status due to network issues, it will display an error message and return to step 5.	
Post-condition: The university student account status updated with the new status successfully.	

Use case name: Assign voting eligibility	
Actor: University Staff	
Brief description: The system should allow assign the voting eligibility for university student has the permission to vote.	
Pre-condition: The university staff must be logged into the system.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select user management section.	4. Display the university staffs and students in the system with sorted list and section.

5. Select a university student.	6. Display the university student information page.
7. Click ‘Edit Voting Eligibility’ button.	8. Display a dialog with yes or no options for university student account voting eligibility.
9. Select the new option. 10. Click ‘Confirm’ button.	11. Update the university student account voting eligibility with the new option selected. 12. Display success message.
Alternative Flow of Events: **A1 Step 4:** If the system fails to load the user list due to network issues, it will display an error message and return to step 3. **A2 Step 11:** If the selected option is same with the current university student voting eligibility status or the system cannot update the university student account voting eligibility status with new status due to network issues, it will display an error message and return to step 5.	
Post-condition: The university student account voting eligibility updated with new status option successfully.	

Use case name: Edit account	
Actor: Admin, University Staff, University Student	
Brief description: The system should allow the user to edit their account information.	
Pre-condition: The user must be logged into the system.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Click the ‘Profile’ option located in the bottom navigation bar.	4. Display the current user account information.
5. Click edit icon.	6. Change the user account information to editable form.
7. Fill with new information.	8. Validates the user account

	information form with syntax and logic checking.
9. Click ‘Edit Account’ button.	10. Update the user account with new user account information form. 11. Display success message.

Alternative Flow of Events:

A1 Step 5:

If the device does not has internet connection, it will display an error message and return to step 4.

A2 Step 8:

If the system detects the syntax or logic errors in the editable user account information form, it will display an error message and return to step 7.

A3 Step 10:

If the system fails to update the user account information due to network issues, it will display an error message and return to step 7.

Post-condition: The user account information updated with new information successfully.

4. *Voting Event Management Module*

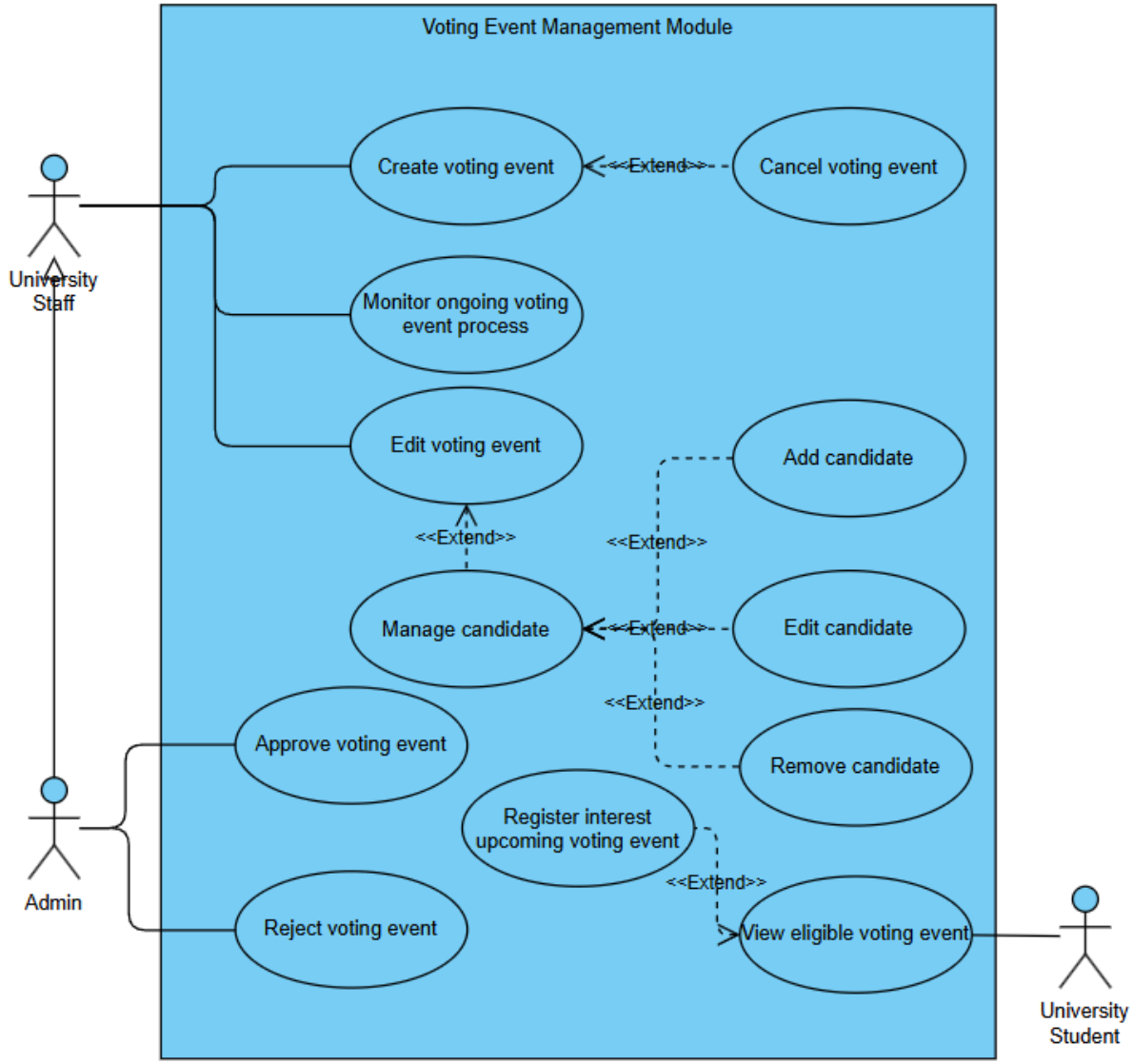


Figure 4.4 Voting Event Management Module

Use case name: Approve voting event	
Actor: Admin	
Brief description: The system should allow the admin to approve the voting event that created by admin or university staff.	
Pre-condition: The admin is logged into the system.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home	2. Display dashboard page.

page).	
3. Select pending voting event section.	4. Redirects to pending voting event page. 5. Load the pending list and display.
6. Select the voting event that is waiting to approve or reject. 7. Click 'Approve' button.	8. Display a dialog with confirmation and the voting event information.
9. Click 'Confirm' button.	10. Process and approve the voting event to available and remove from pending list. 11. Display success message.
Alternative Flow of Events: **A1 Step 5:** If the system fails to load the pending list due to network issues, it will display an error message and return to step 3. **A2 Step 9:** If the admin cancelled the confirmation, it will return to step 6. **A3 Step 10:** If the system fails to process the approvement of the voting event due to network issues, it will display an error message with retry button, if retry failed, it will return to step 6 and display retry failed message.	
Post-condition: The voting event status updated to available and removed from pending list.	

Use case name: Reject voting event	
Actor: Admin	
Brief description: The system should allow the admin to reject the voting event that created by admin or university staff.	
Pre-condition: The admin is logged into the system.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.

3. Select pending voting event section.	4. Redirects to pending voting event page. 5. Load the pending list and display.
6. Select the voting event that is waiting to approve or reject. 7. Click 'Reject' button.	8. Display a dialog with confirmation and the voting event information.
9. Click 'Confirm' button.	10. Process and reject the voting event to rejected and remove from pending list. 11. Display success message.
Alternative Flow of Events: **A1 Step 5:** If the system fails to load the pending list due to network issues, it will display an error message and return to step 3. **A2 Step 9:** If the admin cancelled the confirmation, it will return to step 6. **A3 Step 10:** If the system fails to process the rejection of the voting event due to network issues, it will display an error message with retry button, if retry failed, it will return to step 6 and display retry failed message.	
Post-condition: The voting event status updated to rejected and removed from pending list.	

Use case name: Create voting event	
Actor: Admin, University Staff	
Brief description: The system should allow the user to create the voting event by specifying criteria and timelines.	
Pre-condition: The user must be logged in to the system.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select voting event section.	4. Redirects to voting event page.

5. Click ‘Create New’ button.	6. Display create voting event page with empty voting event form.
7. Fill the voting event form and set the voting event timeline.	8. Validate the voting event form and the timeline set with syntax and logic validation.
9. Click ‘Create New’ button.	10. Create the voting event with pending status and add to the pending voting event list. 11. Display success message.
Alternative Flow of Events: **A1 Step 8:** If the system detects the syntax or logic errors, it will display an error message and return to step 7. **A2 Step 10:** If the system fails to create the voting event due to network issues, it will display an error message and return to step 7.	
Post-condition: The voting event is created and added to the pending voting event list.	

Use case name: Cancel voting event	
Actor: Admin, University Staff	
Brief description: The system should allow the user to cancel the voting event.	
Pre-condition: ● The user must be logged in to the system. ● The voting event must be available or ongoing process.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select voting event section.	4. Redirects to voting event page.
5. Hold to select voting event.	6. Display a dialog with confirmation message.
7. Click ‘Confirm’ button.	8. Process and cancel the voting event and remove from the system. 9. Remove from pending voting event

	list if haven't approved or rejected. 10. Display success message.
Alternative Flow of Events: **<u>A1 Step 8:</u>** If the system fails to process the cancellation of the voting event due to network issues or the voting event is already cancelled by other user due to network latency, it will display an error message and return to step 5.	
Post-condition: The voting event is cancelled and removed from the system successfully.	

Use case name: Edit voting event	
Actor: Admin, University Staff	
Brief description: The system should allow the user to edit the details of the voting event.	
Pre-condition: ● The user must be logged in to the system. ● The voting event must be available or ongoing process.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select voting event section.	4. Redirects to voting event page.
5. Hold to select voting event.	6. Display a dialog with part of voting event details.
7. Click 'Edit' button.	8. Display complete voting event details in editable form.
9. Edit the voting event details.	10. Validate the logic and syntax of the voting event details.
11. Click 'Edit' button.	12. Update the voting event with new details. 13. Display success message.
Alternative Flow of Events: **<u>A1 Step 8:</u>** If the voting event is rejected, unavailable, it will display an error message and return to step 7.	

A2 Step 10:
If the system detects the syntax or logic errors of the voting event details, it will display an error message and return to step 9.

A3 Step 12:
If the system fails to update the voting event details due to network issues, it will display an error message and return to step 9.

Post-condition: The voting event is edited and updated with new details.

Use case name: Add candidate

Actor: Admin, University Staff

Brief description: The system should allow the user to add the candidate to participate in the voting event.

Pre-condition:
- The user must be logged in to the system.
- The voting event must be available or ongoing process.
- The candidate account must be available and eligible.

Main flow of Events:

Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select voting event section.	4. Redirects to voting event page.
5. Select voting event.	6. Display the voting event details page.
7. Click ‘Add Candidate’ button.	8. Display add candidate page with empty candidate form.
9. Fill the candidate information.	10. Validate the candidate information form.
11. Click ‘Add Candidate’ button.	12. Add the candidate to the voting event. 13. Display success message.

Alternative Flow of Events:
A1 Step 6:
If the system fails to load the voting event details page, it will display an error message and return to step 5.

A2 Step 10:
If the system detects the error such as syntax or logic errors from the candidate information form, it will display an error message and return to step 9.

A3 Step 12:
If the system fails to add the new candidate to the voting event due to network issues or repeated candidate, it will display an error message and return to step 9.

Post-condition: The candidate is added to the voting event successfully.

Use case name: Edit candidate

Actor: Admin, University Staff

Brief description: The system should allow the user to edit the information of the candidate displayed in the voting event.

Pre-condition:
- The user must be logged in to the system.
- The voting event must be available or ongoing process.
- The candidate account must be available and eligible.

Main flow of Events:

Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select voting event section.	4. Redirects to voting event page.
5. Select voting event.	6. Display the voting event details page.
7. Select an existed candidate in the voting event.	8. Display editable candidate details form.
9. Edit the candidate information.	10. Validate the candidate information form.
11. Click ‘Edit Candidate’ button.	12. Update the new candidate information in the voting event. 13. Display success message.

Alternative Flow of Events:
A1 Step 6:
If the system fails to load the voting event details page, it will display an error message and

return to step 5.

A2 Step 10:
If the system detects the error such as syntax or logic errors from the candidate information form, it will display an error message and return to step 9.

A3 Step 12:
If the system fails to update the new information of the candidate in the voting event due to network issues, it will display an error message and return to step 9.

Post-condition: The candidate information in the voting event has been updated with new information successfully.

Use case name: Remove candidate

Actor: Admin, University Staff

Brief description: The system should allow the user to remove the candidate from the voting event.

Pre-condition:
- The user must be logged in to the system.
- The voting event must be available or ongoing process.

Main flow of Events:

Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select voting event section.	4. Redirects to voting event page.
5. Select voting event.	6. Display the voting event details page.
7. Select an existed candidate in the voting event.	8. Display editable candidate details form.
9. Click 'Remove Candidate' button.	10. Display a dialog with confirmation message.
11. Click 'Confirm' button.	12. Remove the candidate from the voting event. 13. Display success message.

Alternative Flow of Events:
A1 Step 6:

If the system fails to load the voting event details page, it will display an error message and return to step 5.

A2 Step 12:
If the system fails to remove the candidate from the voting event due to network issues, it will display an error message and return to step 9.

Post-condition: The candidate removed from the voting event successfully.

Use case name: Register interest to upcoming voting event

Actor: University Student

Brief description: The system should allow university student to register their interest to join the upcoming voting event for viewing the voting event details and voting to the candidate.

Pre-condition:
- The university student is logged into the system.
- The university student status is eligible.

Main flow of Events:

Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select voting event section.	4. Redirects to voting event page and display upcoming voting event that is eligible to participate.
5. Select a upcoming voting event.	6. Display the upcoming voting event details.
7. Click 'Register Interest' button.	8. Process and add the university student to the upcoming voting event registration list. 9. Display success message.

Alternative Flow of Events:
A1 Step 6:
If the system fails to load the upcoming voting event details due to network issues, it will display an error message and return to step 5.

A2 Step 8:
If the system fails to add the university student to the upcoming voting event registration

list or the student already in the list, it will display an error message and return to step 5.
Post-condition: The university student successful to register the interest to the upcoming voting event registration list.

5. Blockchain Integration Module

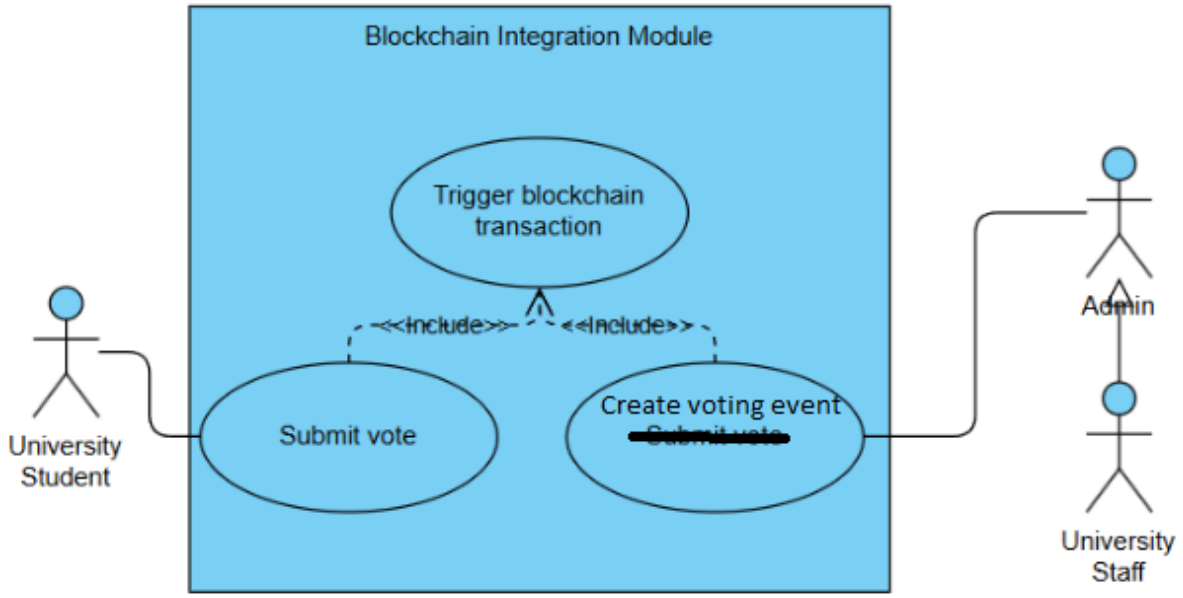


Figure 4.5. Blockchain Integration Module

Use case name: Trigger blockchain transaction for create voting event.	
Actor: Admin, University Staff	
Brief description: The system should allow the user to initiate blockchain transactions for creating and managing voting events, ensuring that blockchain-based voting processes are correctly recorded.	
Pre-condition: ● The user must be logged into the system. ● The user should accessed and confirmed create voting event in 'Voting Event Management Module'.	
Main flow of Events:	
Actor Action	System Response
1. Click the 'Confirm' button.	2. Triggers the blockchain transaction to pop up with transaction details

	and confirmation button.
3. Click ‘Transact’ button.	4. Securely process the transaction and record on the blockchain. 5. Create voting event. 6. Display success message with transaction details.
Alternative Flow of Events: **A1 Step 2:** If the system fails to trigger the transaction due to network issues, it will display an error message and end the transaction process. **A2 Step 4:** If the system fails to process the transaction due to network issues, it will display and error message and end the transaction process.	
Post-condition: The voting event is created and transaction is recorded on the blockchain successfully.	

Use case name: Trigger blockchain transaction for vote submission.	
Actor: University Student	
Brief description: The system should allow the university student to initiate blockchain transactions for vote submission, ensuring that blockchain-based voting processes are correctly recorded.	
Pre-condition: ● The university student must be logged into the system. ● The university student should accessed and confirmed vote submission in ‘Voting Interface Module’.	
Main flow of Events:	
Actor Action	System Response
1. Click the ‘Confirm’ button.	2. Triggers the blockchain transaction to pop up with transaction details and confirmation button.
3. Click ‘Transact’ button.	4. Securely process the transaction and record on the blockchain. 5. Submit the vote to the blockchain. 6. Display success message with transaction details.

Alternative Flow of Events: **A1 Step 2:** If the system fails to trigger the transaction due to network issues, it will display and error message and return to step 1. **A2 Step 4:** If the system fails to process the transaction due to network issues, it will display and error message and return to step 1.
Post-condition: The vote is submitted to and transaction is recorded on the blockchain successfully.

6. *Voting Inteface Module*

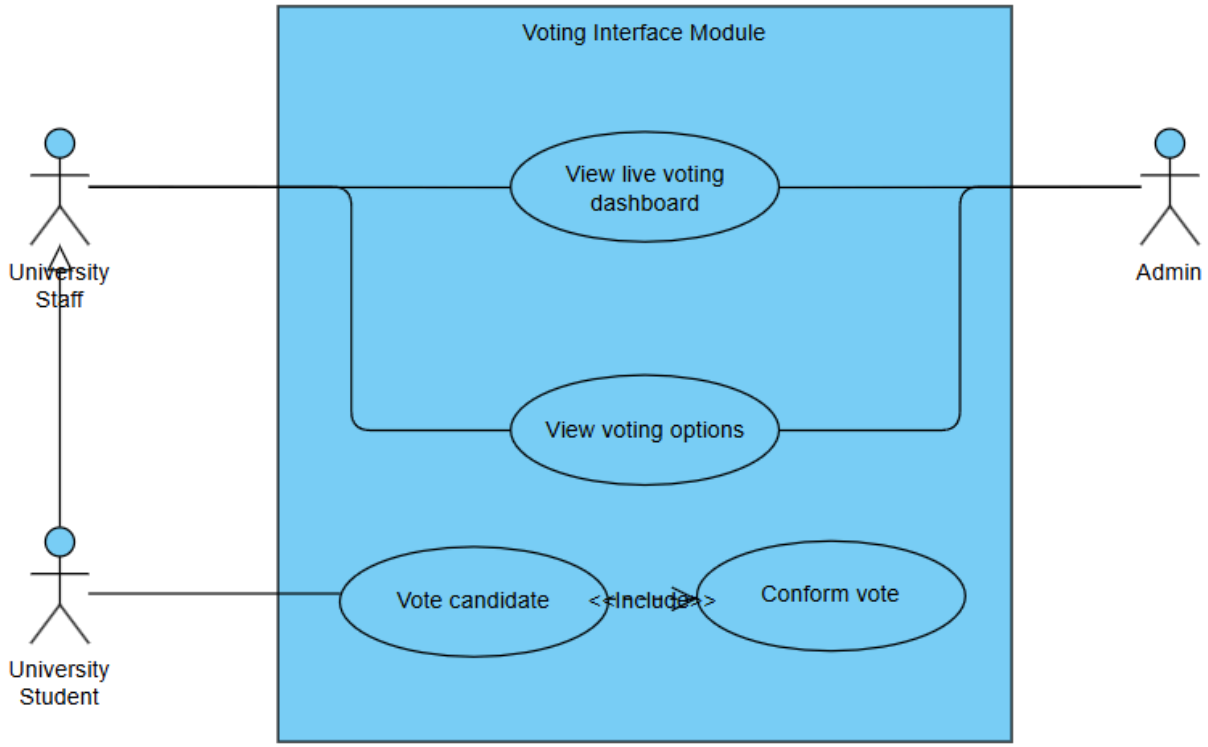


Figure 4.6. Voting Interface Module

Use case name: View live voting dashboard
Actor: Admin, University Staff
Brief description: The system should allow the user to view a live voting dashboard showing progress and participation statistics.
Pre-condition:

<table>
<tr><td colspan="2">

- The users must be logged into the system.
- The voting event must be active or in progress.
- The users must have permission to access the results (University Staff manage the voting event).

</td></tr>
<tr><td colspan="2">Main flow of Events:</td></tr>
<tr><td>Actor Action</td><td>System Response</td></tr>
<tr><td>1. Login to dashboard page (home page).</td><td>2. Display dashboard page.</td></tr>
<tr><td>3. Select the voting event section.</td><td>4. Retrieve voting event list.
5. Redirect to and display voting event list page.</td></tr>
<tr><td>6. Select an active or in progress voting event.</td><td>7. Display the voting event’s details with voting dashboard, statistics and progress status.</td></tr>
<tr><td colspan="2">Alternative Flow of Events:
<u>A1 Step 4:</u>
If the system fails to retrieve voting event list data due to network issues, it will display an error message and return to step 3.

<u>A2 Step 7:</u>
If the selected voting event is not available, it will display an error message and return to step 6.</td></tr>
<tr><td colspan="2">Post-condition: User has access to live voting dashboard.</td></tr>
</table>

<table>
<tr><td colspan="2">Use case name: View voting options</td></tr>
<tr><td colspan="2">Actor: Admin, University Staff, University Student</td></tr>
<tr><td colspan="2">Brief description: The system should allow the users to access the voting interface to view voting options.</td></tr>
<tr><td colspan="2">

Pre-condition:

- The users must be logged into the system.
- The voting event must be active or in progress.

</td></tr>
<tr><td colspan="2">Main flow of Events:</td></tr>
<tr><td>Actor Action</td><td>System Response</td></tr>
<tr><td>1. Login to dashboard page (home</td><td>2. Display dashboard page.</td></tr>
</table>

page).	
3. Select the voting event section.	4. Retrieve voting event list. 5. Redirect to and display voting event list page.
6. Select an active or in progress voting event.	7. Display the voting event's details.
8. Click info icon.	9. Display the voting options of the selected voting event.
Alternative Flow of Events: **A1 Step 4:** If the system fails to retrieve voting event list data due to network issues, it will display an error message and return to step 3. **A2 Step 7:** If the selected voting event is not available, it will display an error message and return to step 6.	
Post-condition: The user has successfully access the voting options of the voting event.	

Use case name: Vote and confirmation	
Actor: University Student	
Brief description: The system should allow the university student to vote candidate and confirm the vote before submission.	
Pre-condition: ● The university student must be logged into the system. ● The voting event must be active or in progress. ● The university student hasn't vote in this voting event before.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select the voting event section.	4. Retrieve voting event list. 5. Redirect to and display voting event list page.
6. Select an active or in progress voting	7. Display the voting event's details.

event.	
8. Click 'Vote' button in the specific candidate tab.	9. Display confirmation of submission vote with the candidate information.
10. Click 'Confirm' button.	11. Submit the vote and store into blockchain. 12. Display successful message.
Alternative Flow of Events: **A1 Step 4:** If the system fails to retrieve voting event list data due to network issues, it will display an error message and return to step 3. **A2 Step 7:** If the selected voting event is not available, it will display an error message and return to step 6. **A4 Step 11:** If the system fails to submit the vote data due to network issues, it will display an error message and return to step 8.	
Post-condition: The university student's vote has submitted, hashed the vote's voter's details and to the blockchain successfully.	

7. *Voting Results Module*

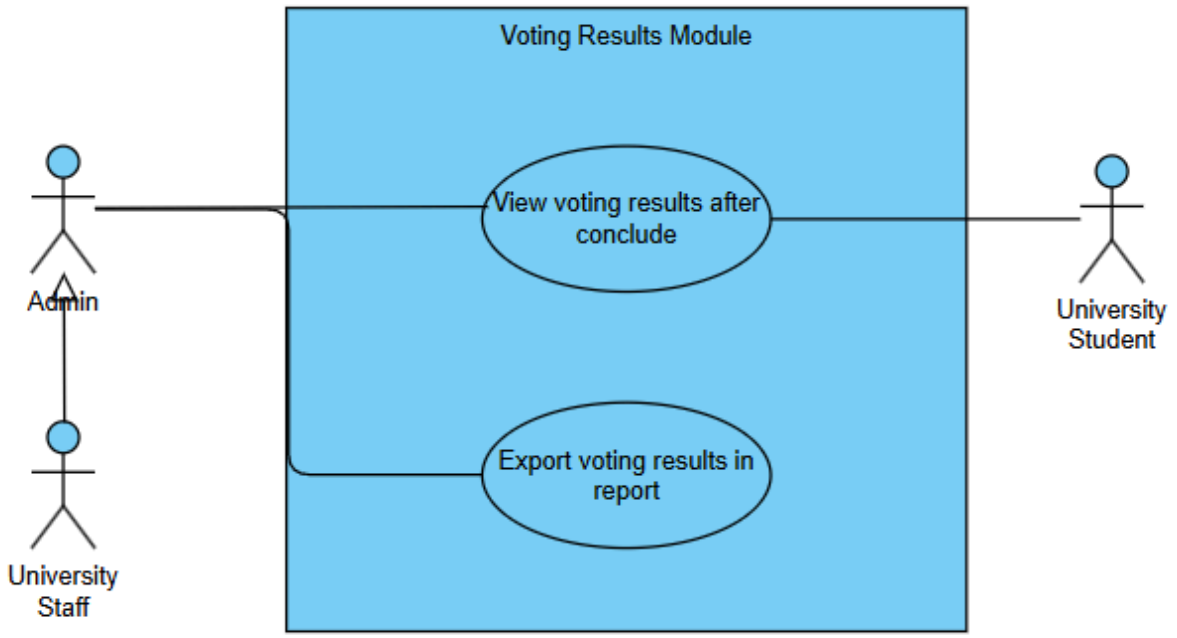


Figure 4.7. Voting Results Module

<table>
<tr><td colspan="2">Use case name: View voting results after conclude</td></tr>
<tr><td colspan="2">Actor: Admin, University Staff, University Student</td></tr>
<tr><td colspan="2">Brief description: The system should allow the users to view the results of the voting events they participated in after they conclude.</td></tr>
<tr><td colspan="2">Pre-condition:
● The users must be logged into the system.
● The voting event must have concluded, and results must be available.
● The users must have permission to access the results (University Staff manage the voting event, University Student for their own participation).</td></tr>
<tr><td colspan="2">Main flow of Events:</td></tr>
<tr><td>Actor Action</td><td>System Response</td></tr>
<tr><td>1. Login to dashboard page (home page).</td><td>2. Display dashboard page.</td></tr>
<tr><td>3. Select voting event section.</td><td>4. Retrieve voting event list.
5. Redirect to and display voting event list page.</td></tr>
<tr><td>6. Select concluded voting event section.</td><td>7. Display concluded voting events.</td></tr>
<tr><td>8. Select a concluded voting event.</td><td>9. Display the voting event results.</td></tr>
<tr><td colspan="2">Alternative Flow of Events:
<u>A1 Step 4:</u>
If the system fails to retrieve voting event list data due to network issues, it will display an error message and return to step 3.

<u>A2 Step 7:</u>
If the selected voting event is not available, it will display an error message and return to step 6.</td></tr>
<tr><td colspan="2">Post-condition: Voting results for the selected voting event are displayed for the user.</td></tr>
</table>

<table>
<tr><td>Use case name: Export voting results in report</td></tr>
<tr><td>Actor: Admin, University Staff</td></tr>
<tr><td>Brief description: The system should allow the users to export voting results for reporting</td></tr>
</table>

purposes.	
Pre-condition: The user must logged in to the system.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select voting event section.	4. Retrieve voting event list. 5. Redirect to and display voting event list page.
6. Select concluded voting event section.	7. Display concluded voting events.
8. Select a concluded voting event.	9. Display the voting event results.
10. Click 'Export to Report' button.	11. Display a dialog for users to choose type of report format.
12. Select report format. 13. Click 'Export' button.	14. Generates the voting results in the selected format.
Alternative Flow of Events: N/A	
Post-condition: The voting results file is generated in the chosen format.	

8. Reporting Module

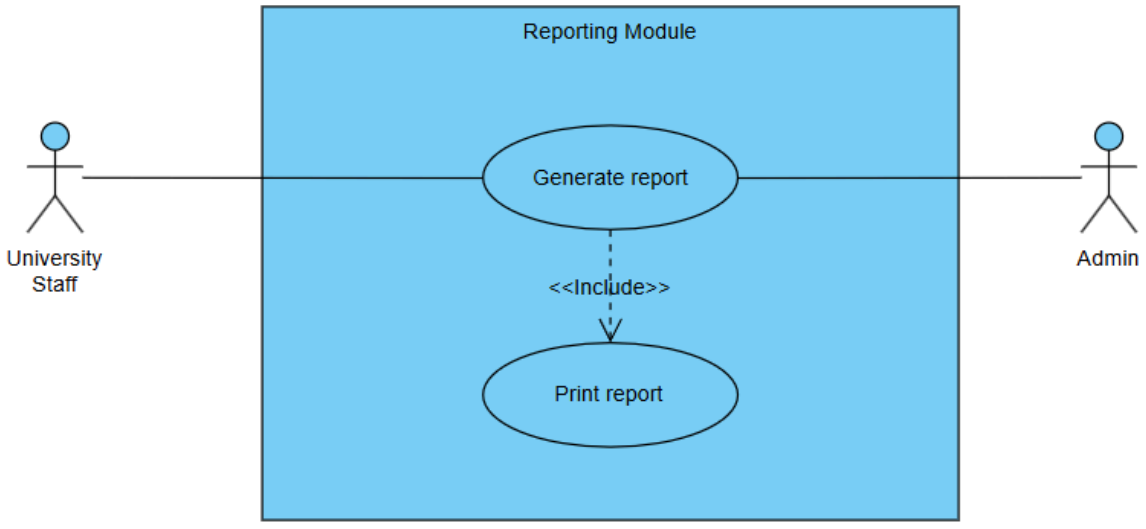


Figure 4.8. Reporting Module

<table>
<tr><td colspan="2">Use case name: Generate report</td></tr>
<tr><td colspan="2">Actor: Admin, University Staff</td></tr>
<tr><td colspan="2">Brief description: Allow the users to generate summary reports for specific voting events.</td></tr>
<tr><td colspan="2">Pre-condition: The user must logged into the system.</td></tr>
<tr><td colspan="2">Main flow of Events:</td></tr>
<tr><td>Actor Action</td><td>System Response</td></tr>
<tr><td>1. Login to dashboard page (home page).</td><td>2. Display dashboard page.</td></tr>
<tr><td>3. Select the report section.</td><td>4. Redirects to report page.</td></tr>
<tr><td>5. Select a voting event from available list of voting events.
6. Click ‘Generate Report’ button.</td><td>7. Retrieve information related to the voting event selected
8. Generates the summary report.
9. Display summary report.</td></tr>
<tr><td colspan="2">Alternative Flow of Events:
<u>A1 Step 6:</u>
If no voting event selected, it will display an error message and return to step 5.

<u>A2 Step 7:</u>
If the system fails to retrieve information due to network issues, it will display an error and return to step 5.</td></tr>
<tr><td colspan="2">Post-condition: Selected summary report generated successfully and displayed.</td></tr>
</table>

9. *Notifications Module*

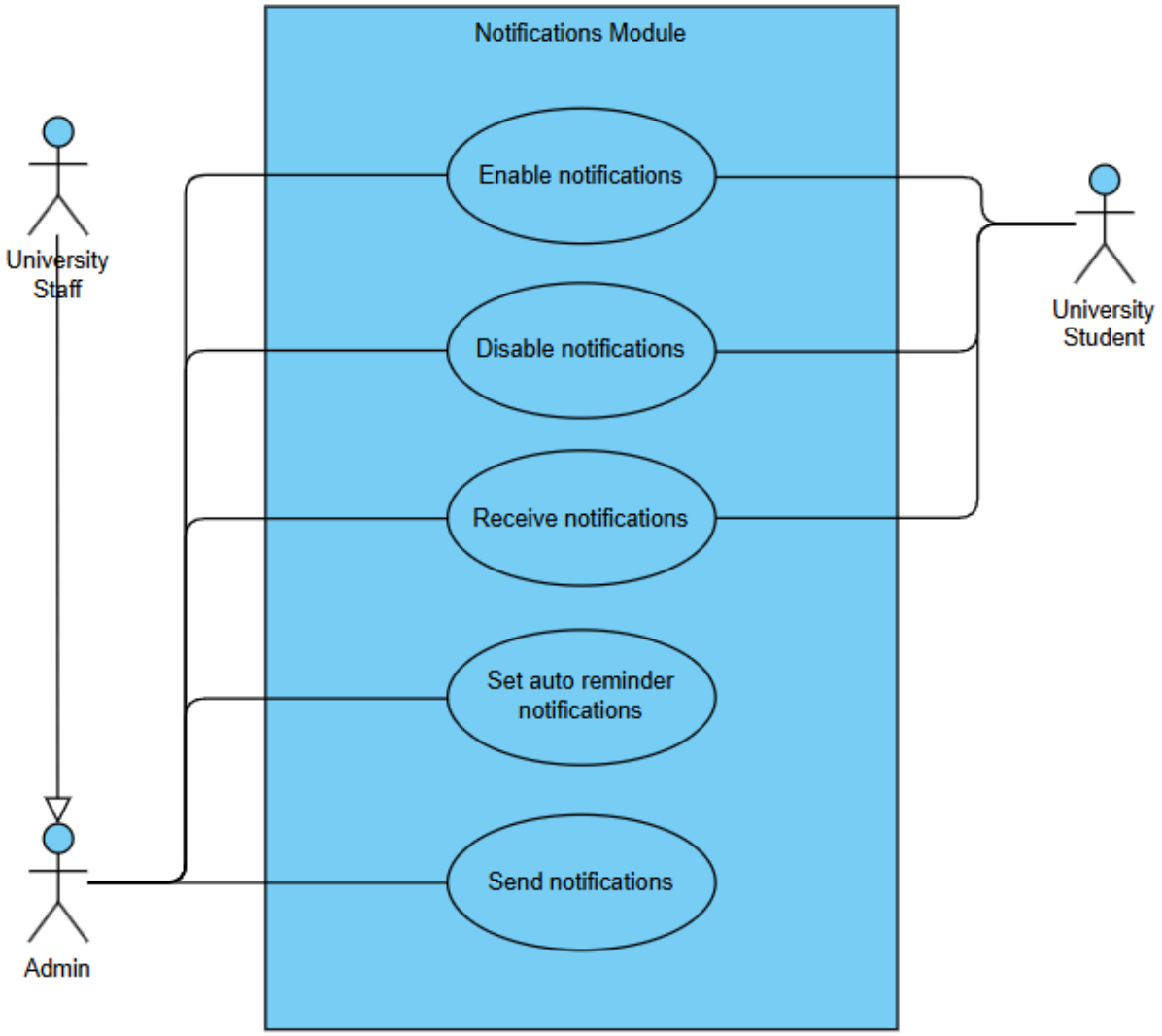


Figure 4.9. Notifications Module

Use case name: Receive notification	
Actor: Admin, University Staff, University Student	
Brief description: The system should allow the users receive notifications about important updates, reminders, and events based on their roles.	
Pre-condition: The user must be logged into the system.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page. 3. Receive the notifications after login

	done. 4. Display the notifications.
Alternative Flow of Events: **<u>A1 Step 3:</u>** If the system fails to retrieve the notifications due to network issues, it will not display any notifications.	
Post-condition: Notifications received successfully.	

Use case name: Send notification	
Actor: Admin, University Staff	
Brief description: The system should allow the users send notifications to other users about updates, reminders, or event schedules.	
Pre-condition: The user must be logged into the system.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select the notification section.	4. Redirects to notification page.
5. Click ‘Send Notification’ button.	6. Redirects to send notification page with notification form.
7. Input parameters to the notification form and select the specific users to receive the notification.	8. Validates the correctness of parameter syntax and logic.
9. Click ‘Send’ button.	10. Sends the notification to the selected users. 11. Display success message.
Alternative Flow of Events: **<u>A1 Step 8:</u>** If the system detects the parameters in the notification form has syntax or logic errors or the selected users does not exist or available, it will display an error message and return to step 7. **<u>A2 Step 10:</u>** If the system fails to send the notification due to network issues, it will display an error message and return to step 7.	

Post-condition: Notification sent to the selected users successfully.

Use case name: Set automation reminder notification	
Actor: Admin, University Staff	
Brief description: The system should allow the users to configure automated reminders for important tasks and deadlines.	
Pre-condition: The user must be logged into the system.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select the notification section.	4. Redirects to notification page.
5. Click 'Set Automation Reminder' button.	6. Redirects to set automation reminder page.
7. Select the notifications or voting events created. 8. Set the time reminder.	9. Validates the logic of selected time.
10. Click 'Set' button.	11. Automation reminder notification set. 12. Display success message.
Alternative Flow of Events: **<u>A1 Step 9:</u>** If the system detects the notifications or voting events not available or the time set is not logically, it will display an error message and return to step 7.	
Post-condition: Automation reminder notification set successfully and notify when the time reached.	

Use case name: Enable notification
Actor: Admin, University Staff, University Student

Brief description: The system should allow the users to enable notifications for receiving updates and reminders.	
Pre-condition: The user must be logged into the system.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select the notification section.	4. Redirects to notification page.
5. Click settings icon of the notification	6. Display dialog with types of notification.
7. Check the types of notification. 8. Click 'Confirm' button.	9. Enable the types of notification to receive the notifications. 10. Display success message.
Alternative Flow of Events: **A1 Step 8:** If the user clicked 'Cancel' button, it will return to step 3.	
Post-condition: Notifications are enabled successfully.	

Use case name: Disable notification	
Actor: Admin, University Staff, University Student	
Brief description: The system should allow the users to disable notifications for specific updates or reminders.	
Pre-condition: The user must be logged into the system.	
Main flow of Events:	
Actor Action	System Response
1. Login to dashboard page (home page).	2. Display dashboard page.
3. Select the notification section.	4. Redirects to notification page.
5. Click settings icon of the notification	6. Display dialog with types of notification.

7. Uncheck the types of notification. 8. Click ‘Confirm’ button.	9. Disable the types of notification to receive the notifications. 10. Display success message.
Alternative Flow of Events: **A1 Step 8:** If the user clicked ‘Cancel’ button, it will return to step 3.	
Post-condition: Notifications are disabled successfully.	

4.2.2 Activity Diagram

*Note

- User is for all roles (Admin, University Staff, University Student)

1. Authentication Module

Login using credentials

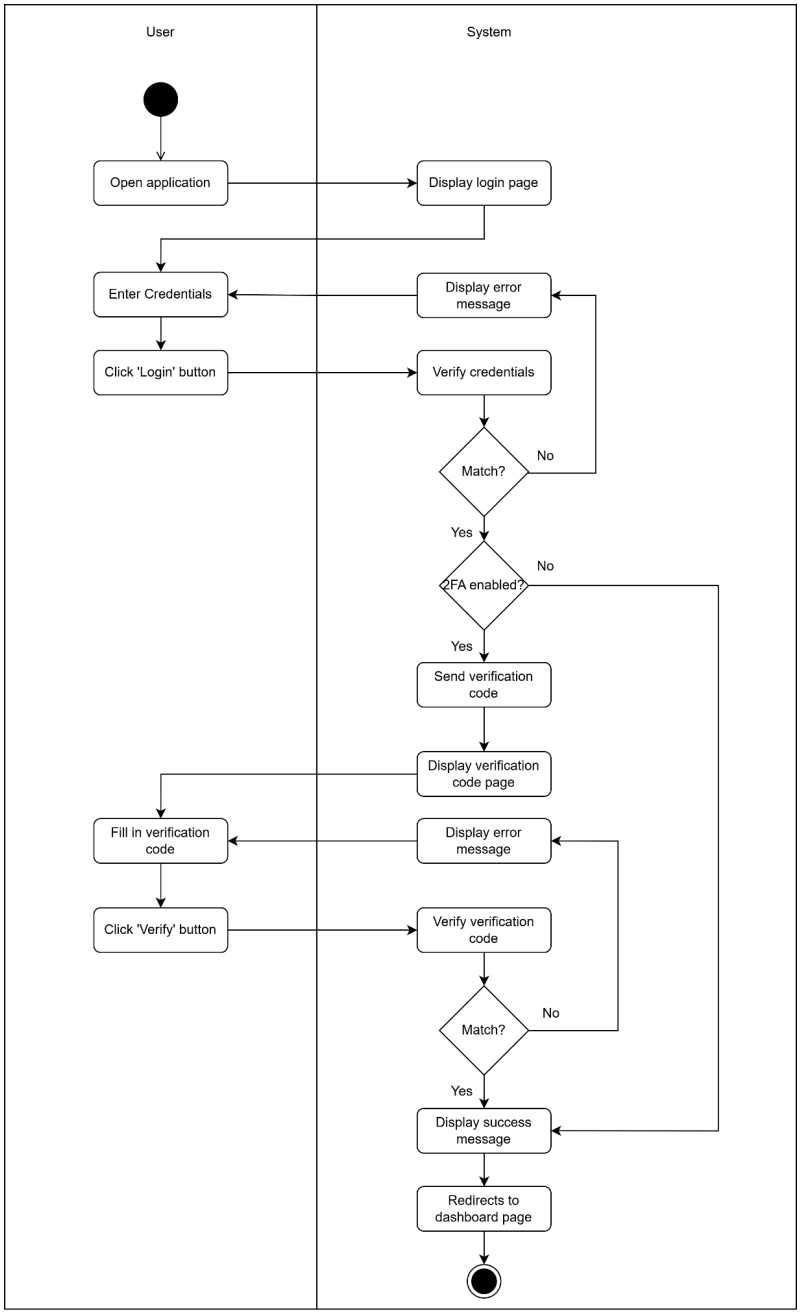


Figure 4.11 Login using Credentials

Login using Metamask

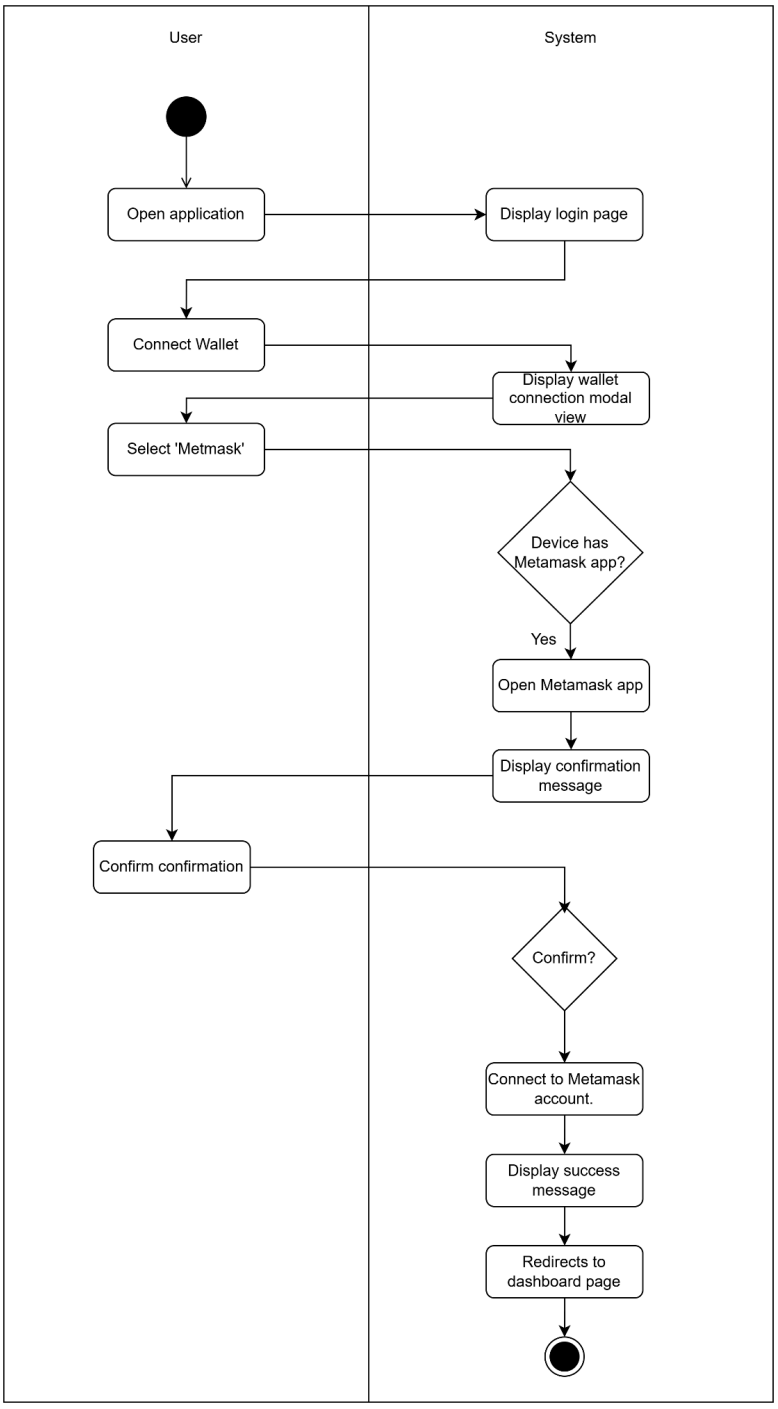


Figure 4.12 Login using Metamask

Metamask account authentication

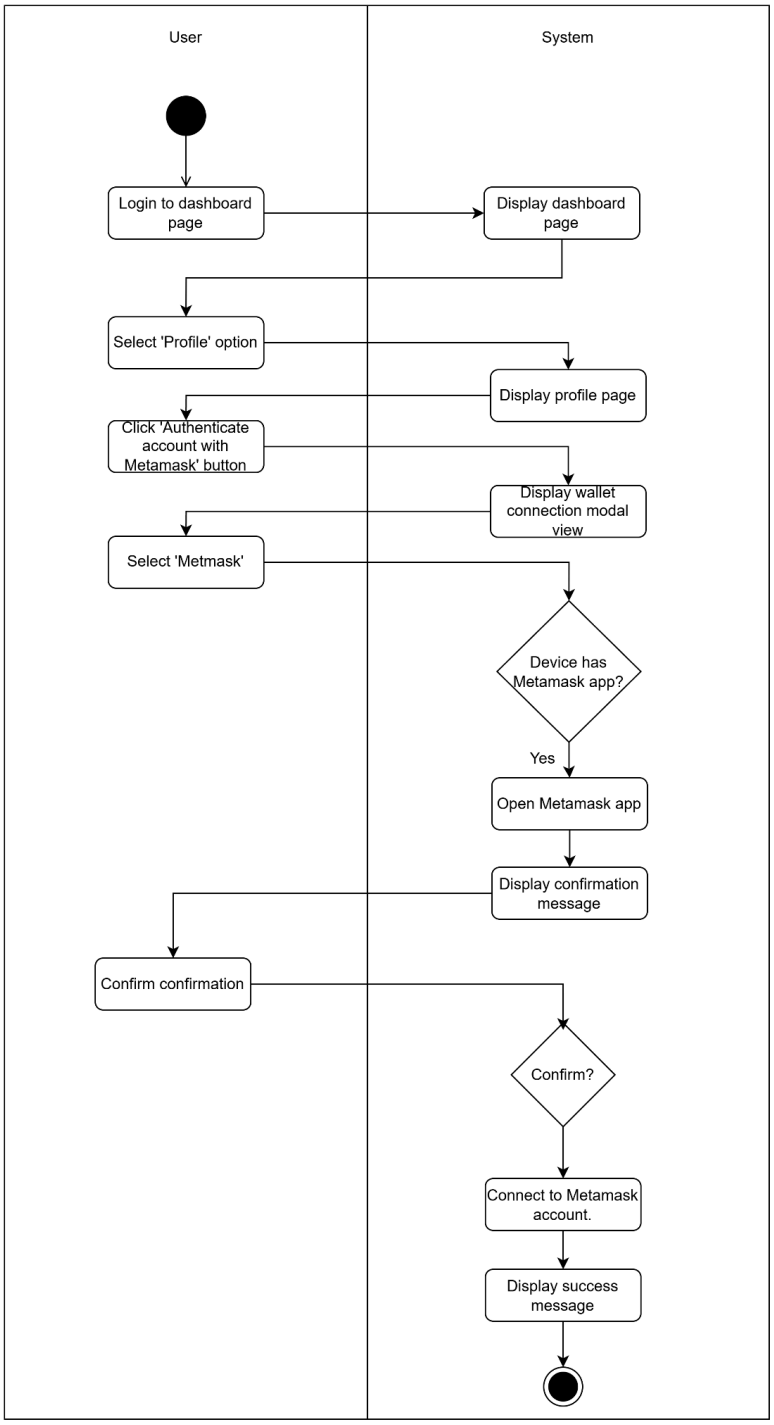


Figure 4.13 Authenticate Metamask Account

Reset password

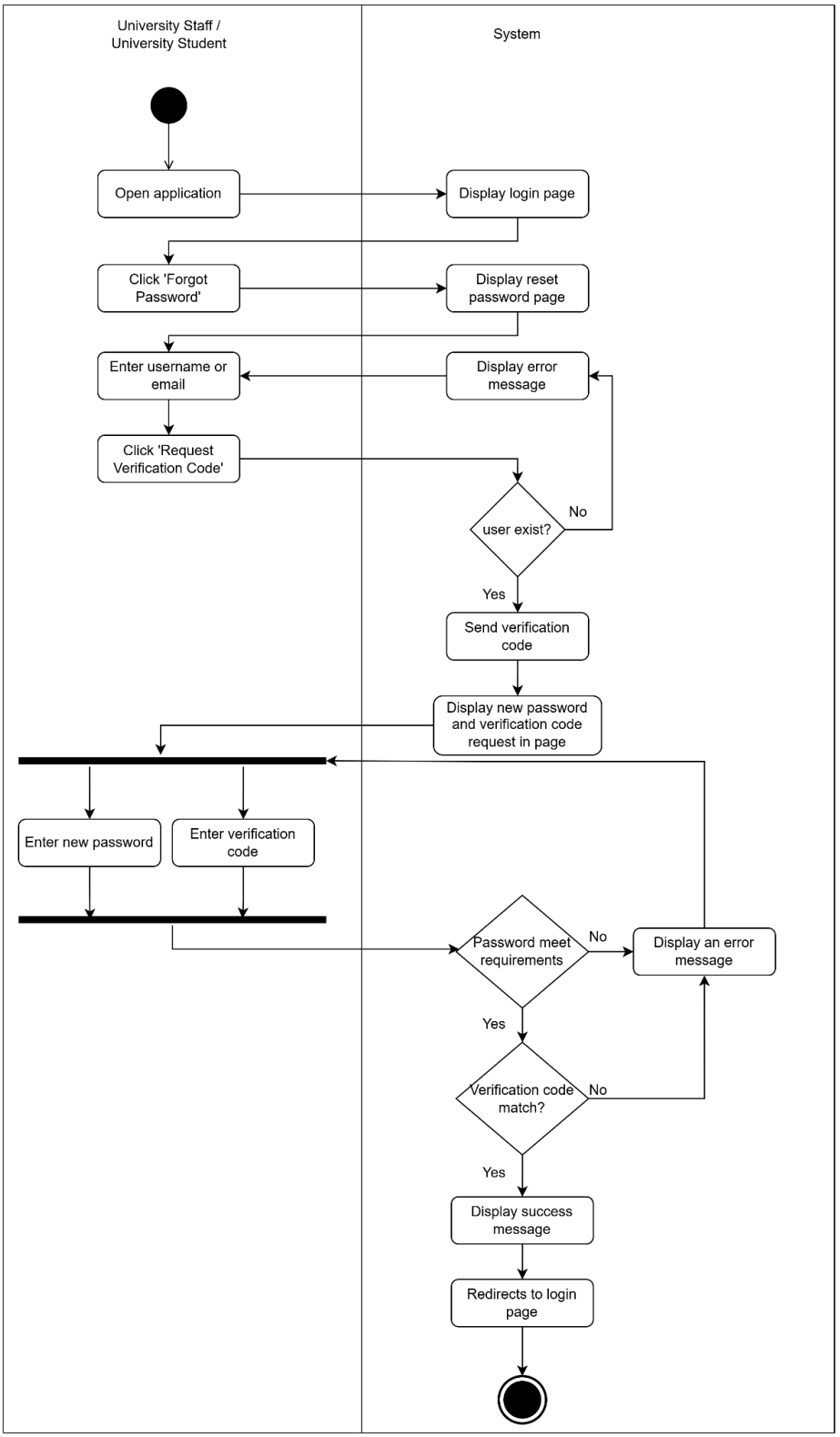


Figure 4.14 Reset password

Register account

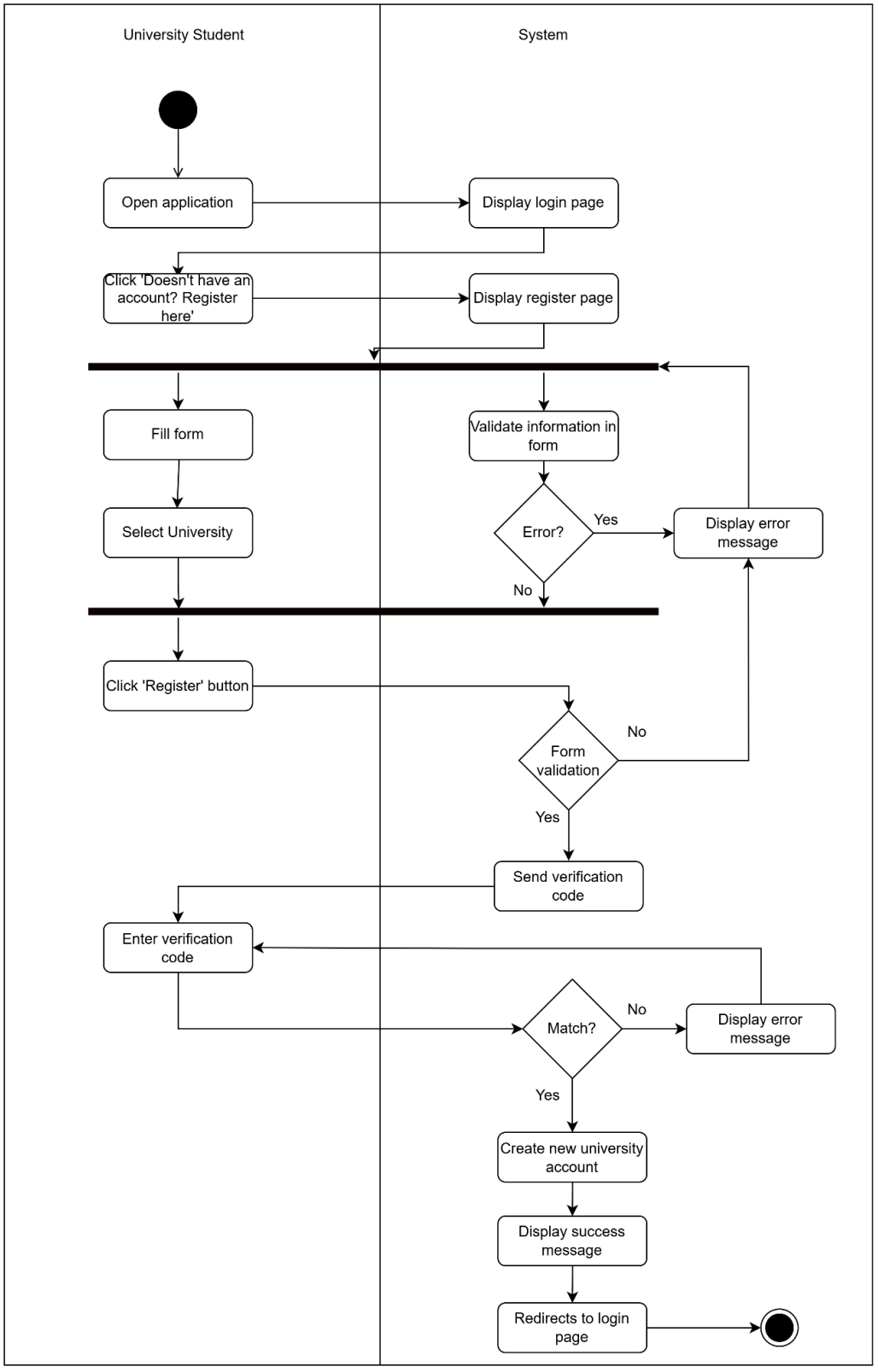


Figure 4.15. Register account

Logout

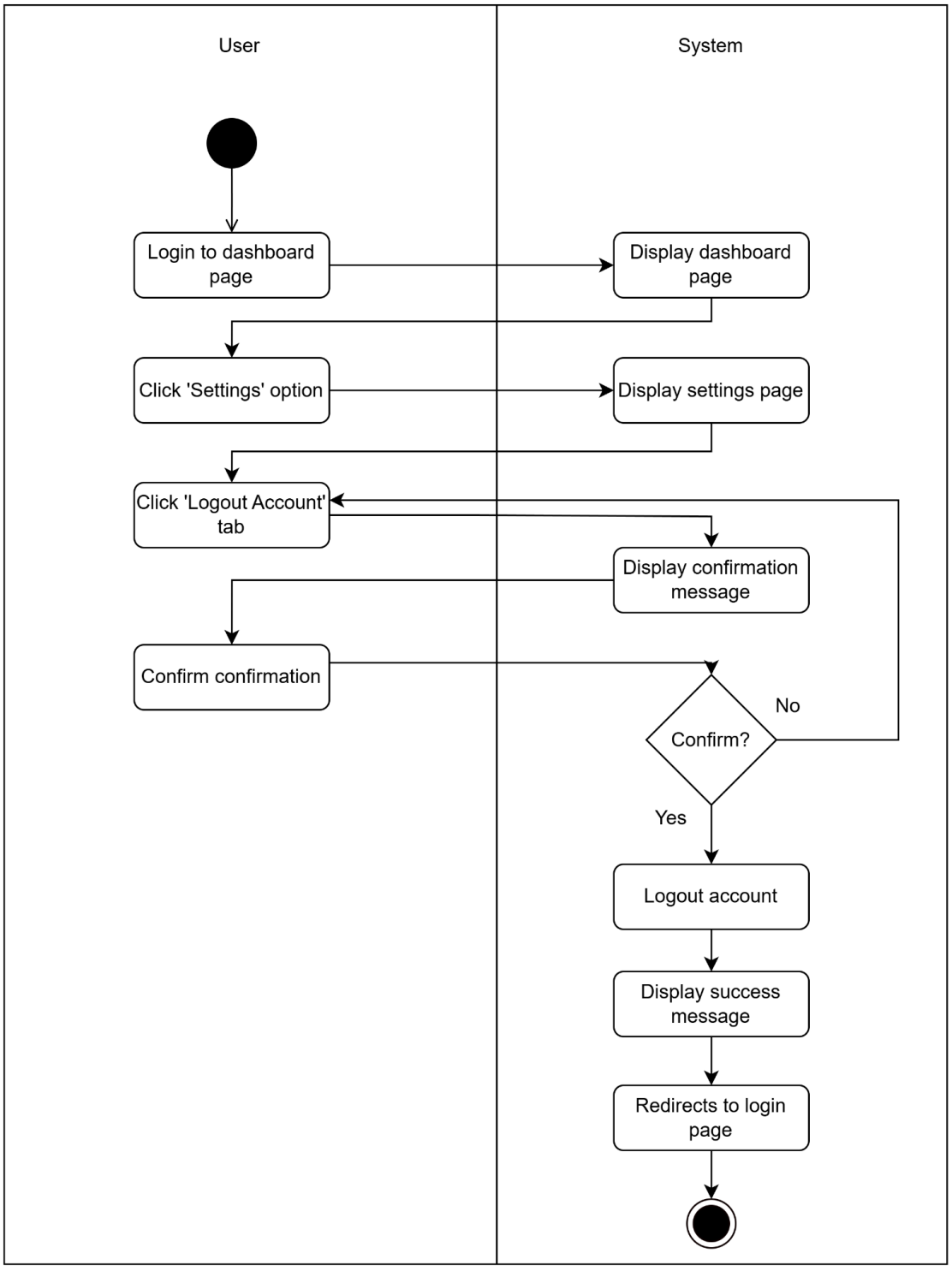


Figure 4.16. Logout

2. User Management Module

Manage university student account status

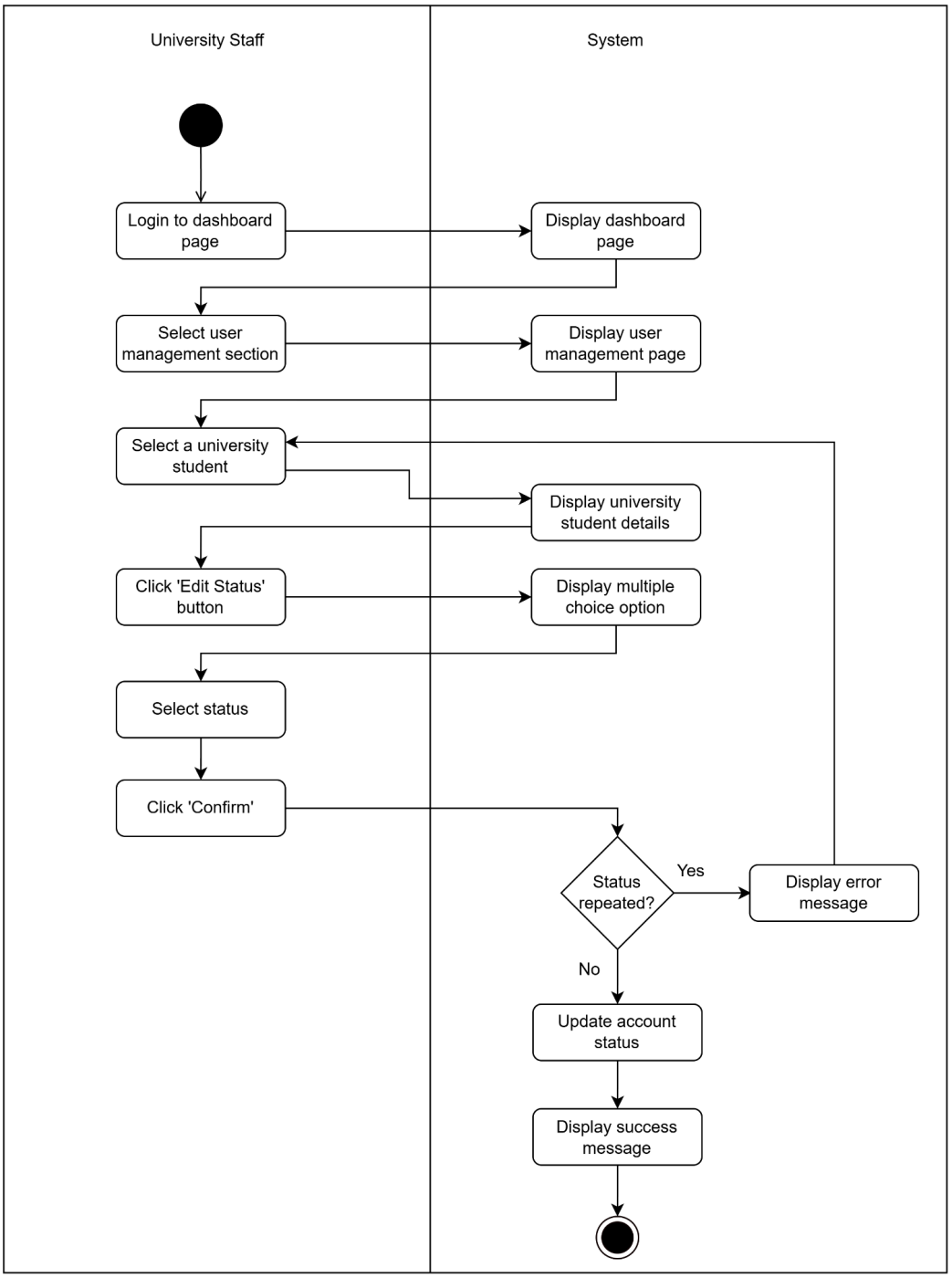


Figure 4.18. Manage university student account status

Edit Account

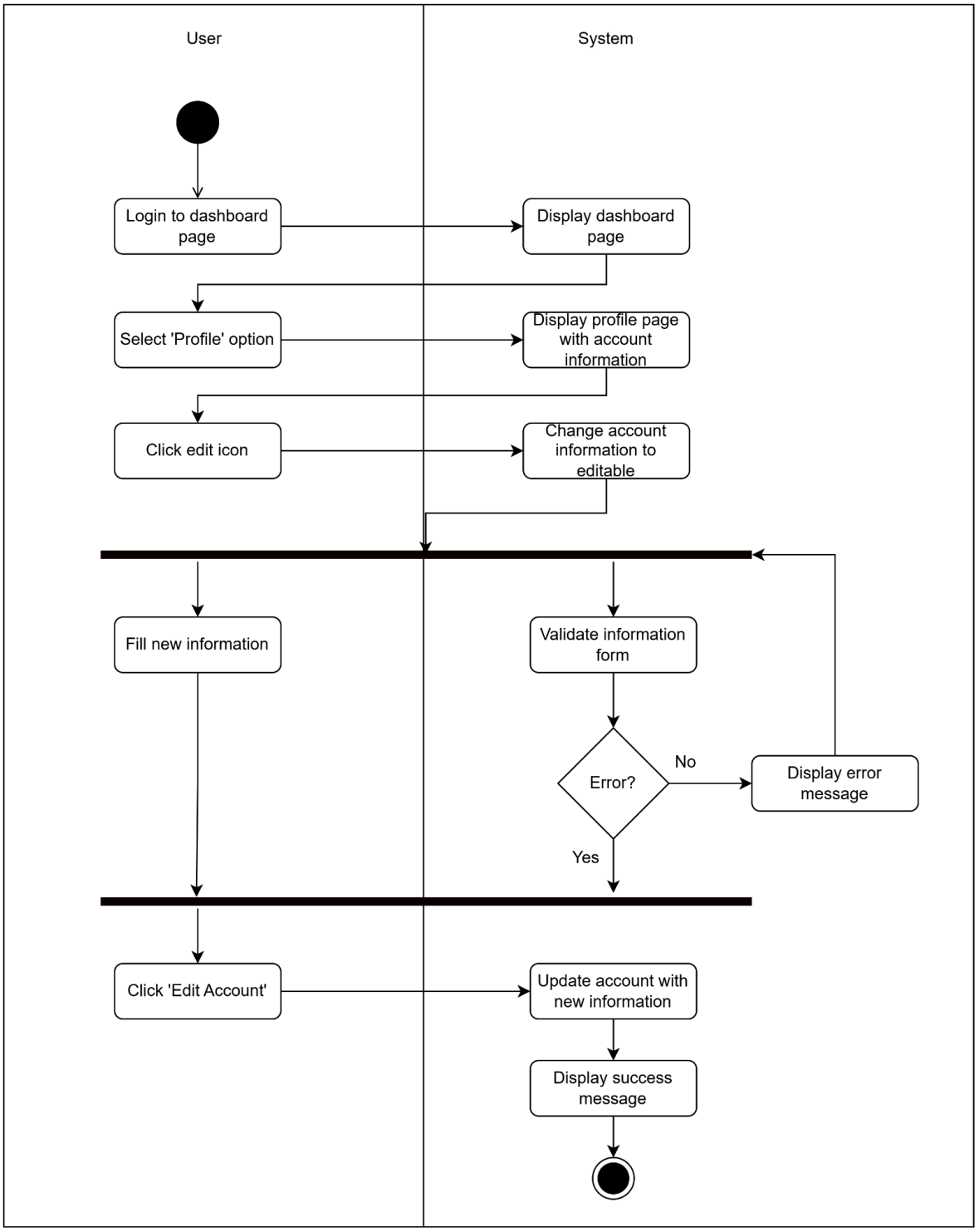


Figure 4.19. Edit account

3. *Voting Event Management Module*

Approve voting event

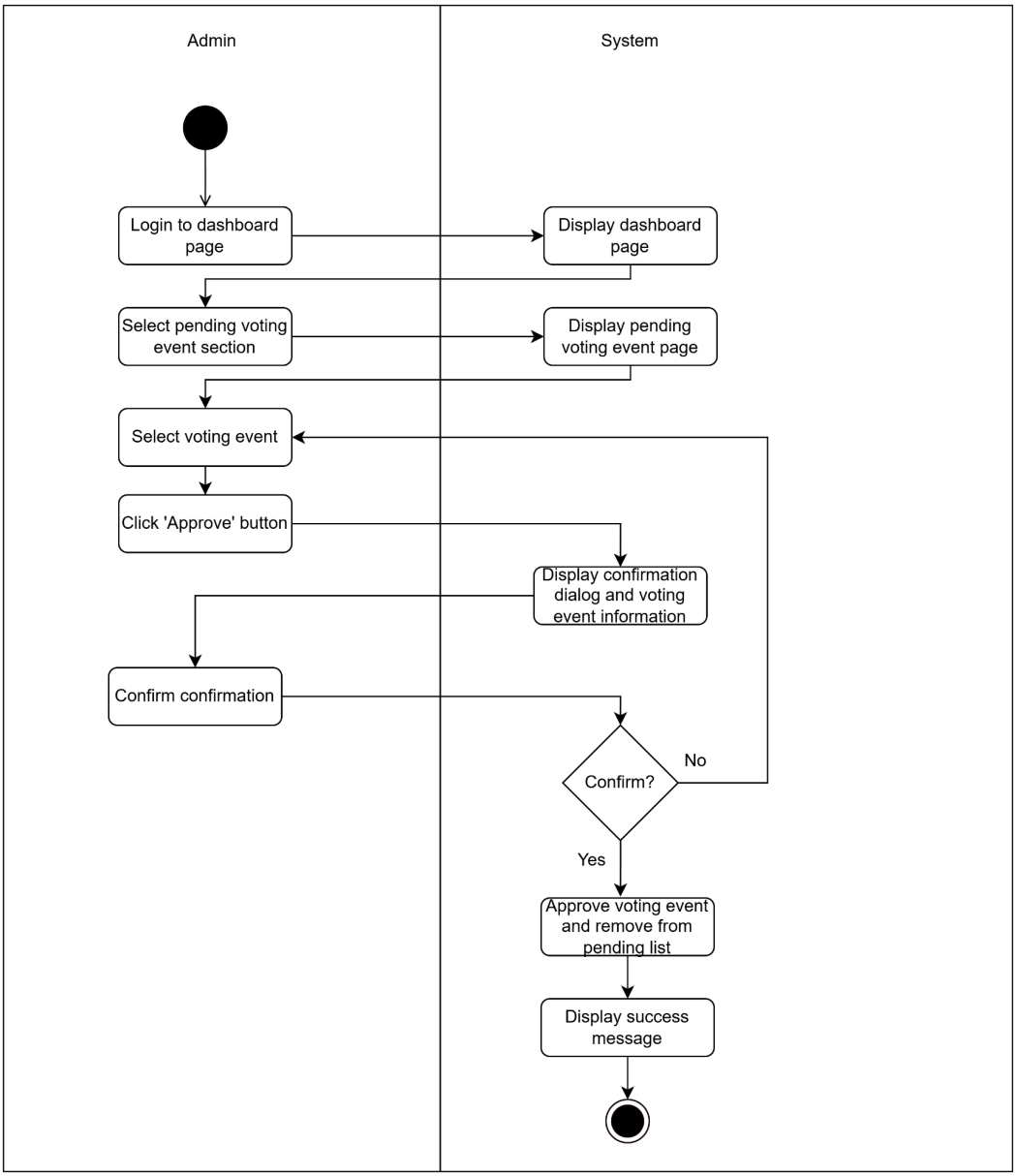


Figure 4.20. Approve voting event

Reject voting event

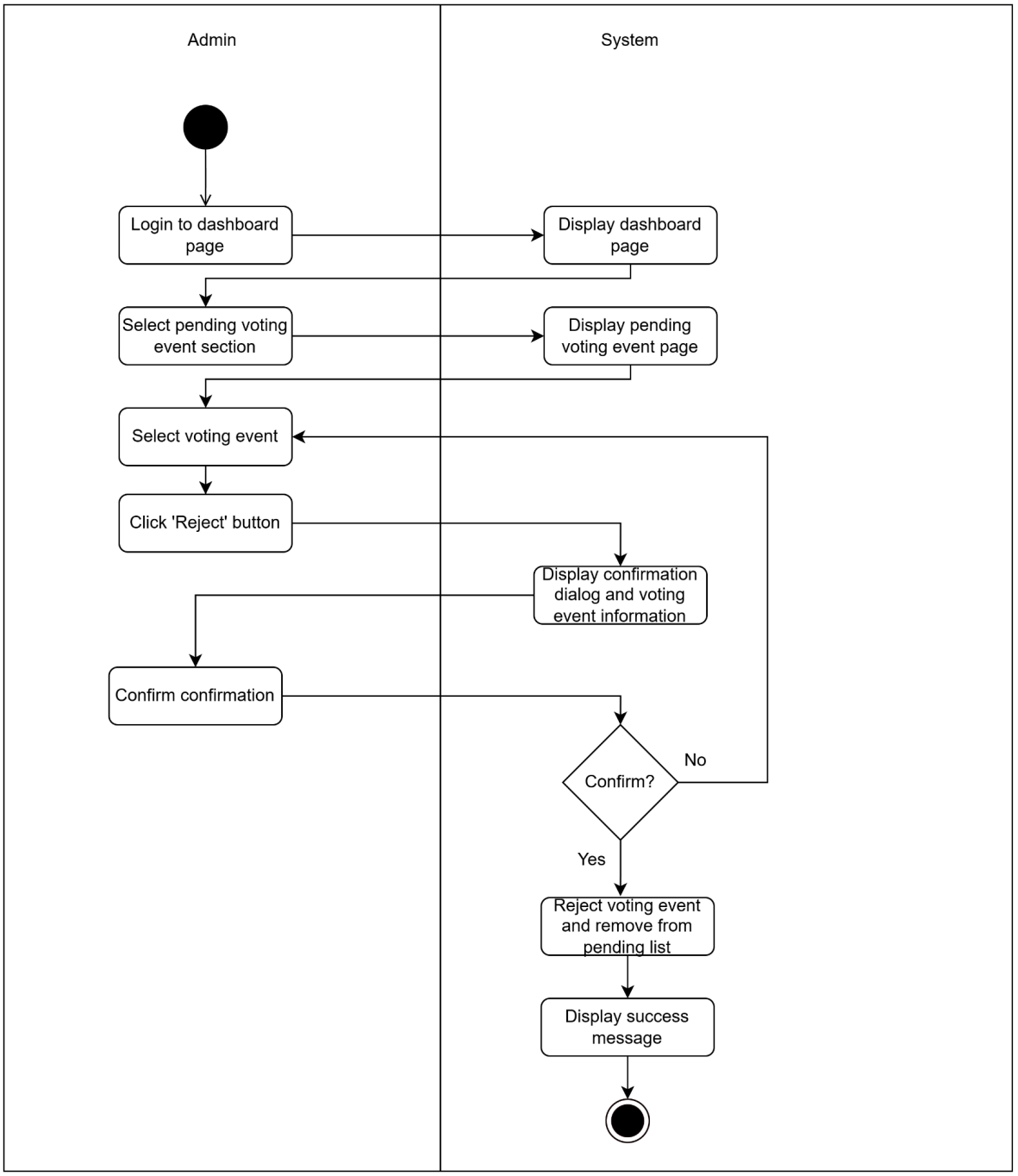


Figure 4.21. Reject voting event

Create voting event

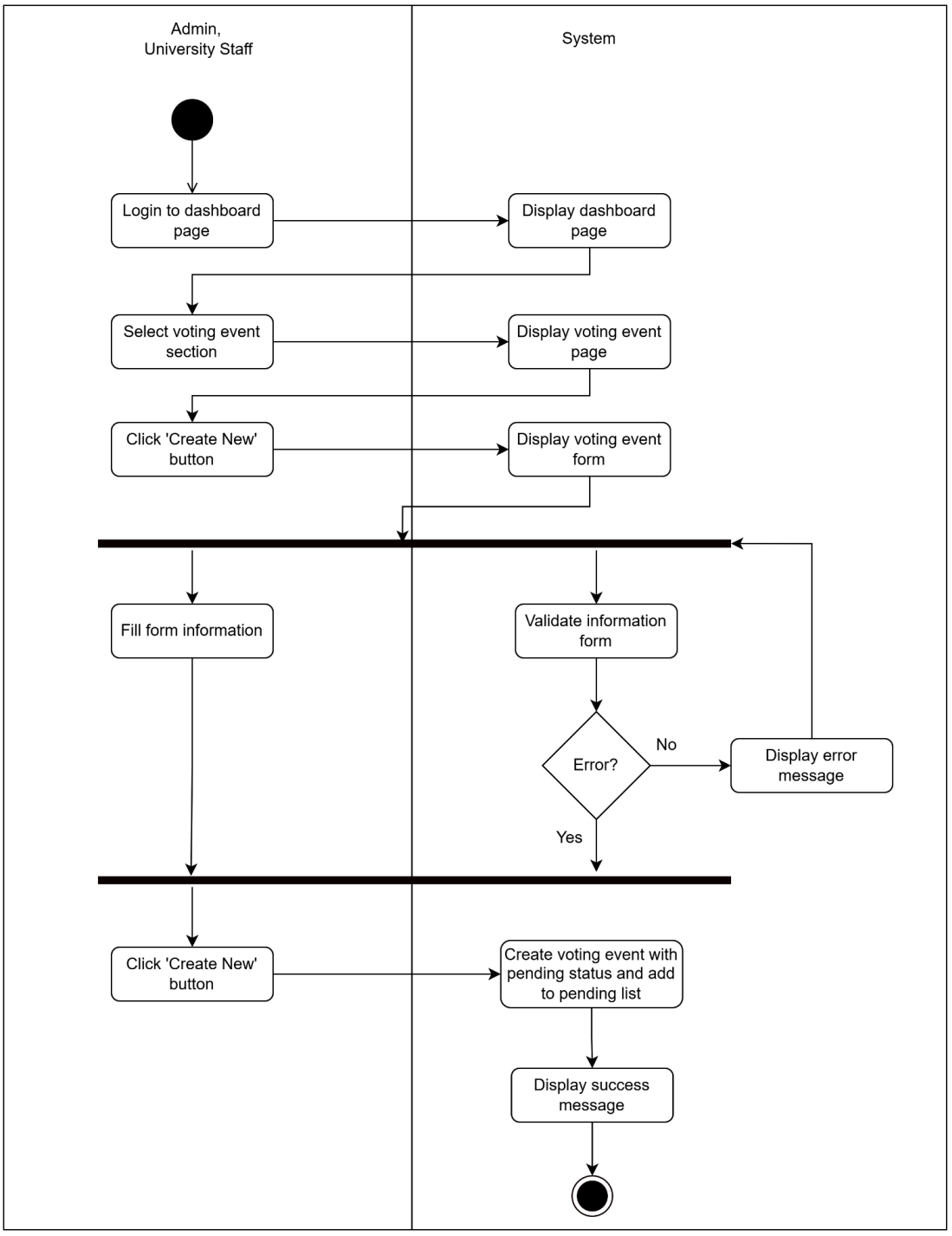


Figure 4.22. Create voting event

Cancel voting event

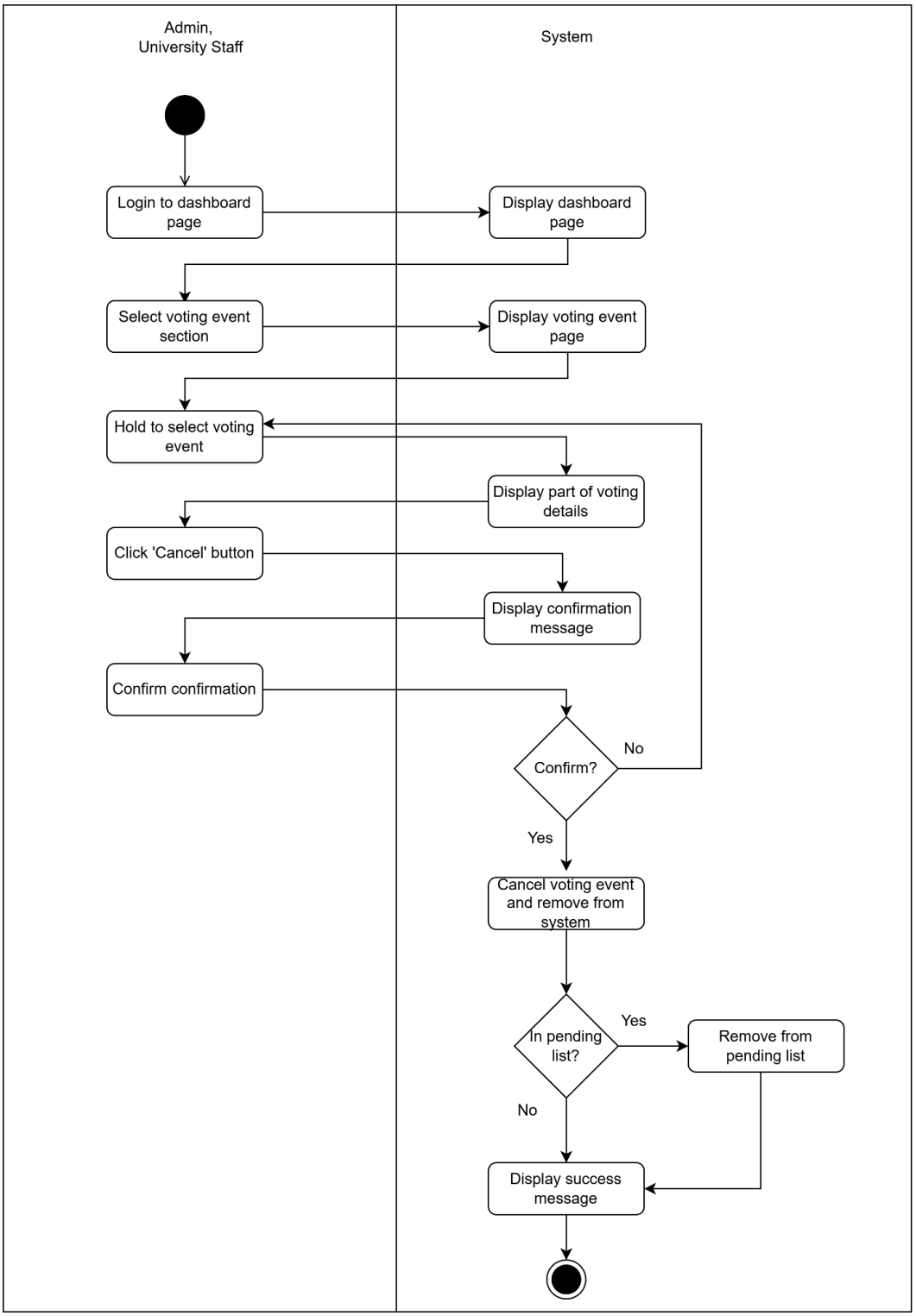


Figure 4.23. Cancel voting event

Edit voting event

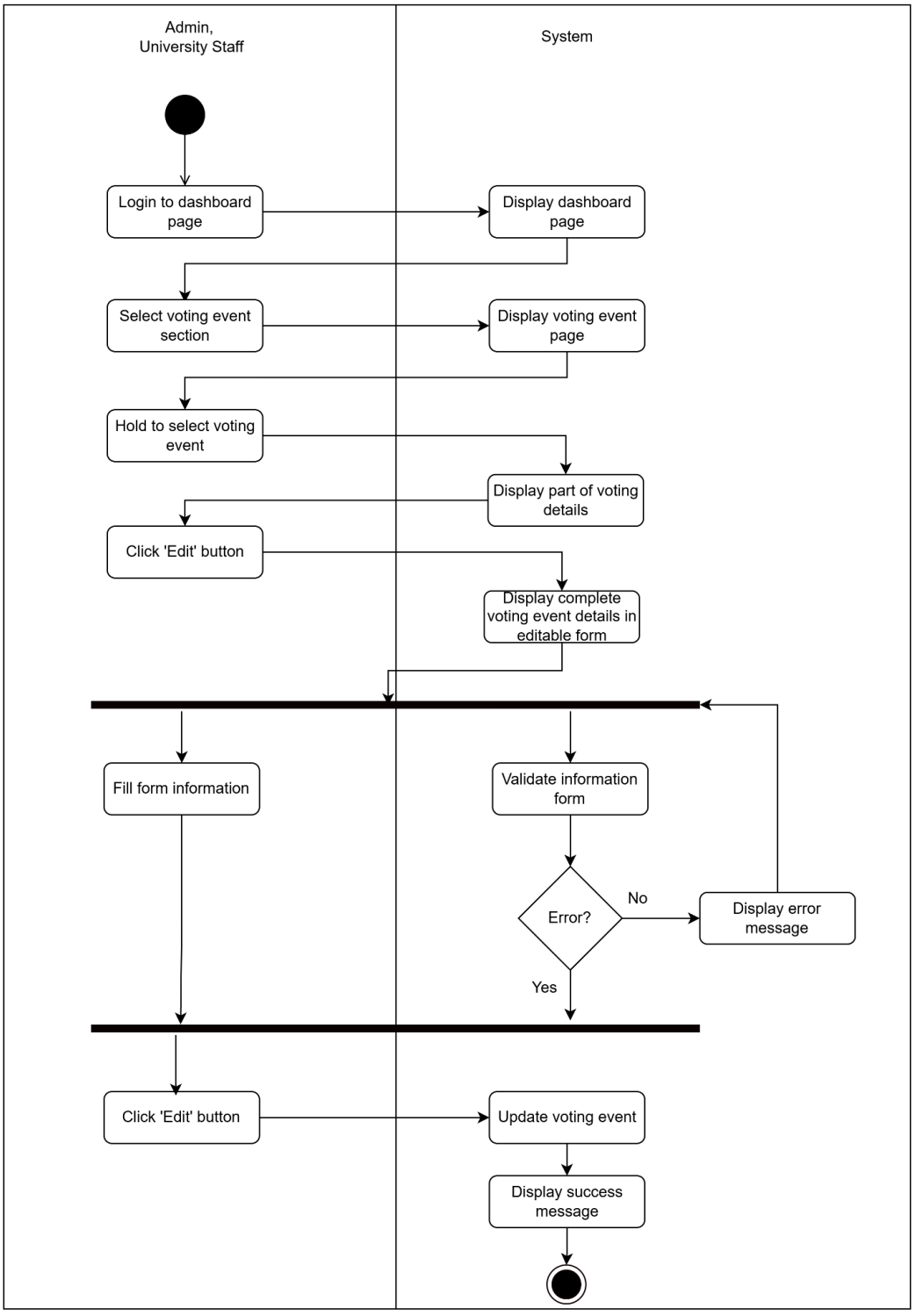


Figure 4.24. Edit voting event

Add candidate

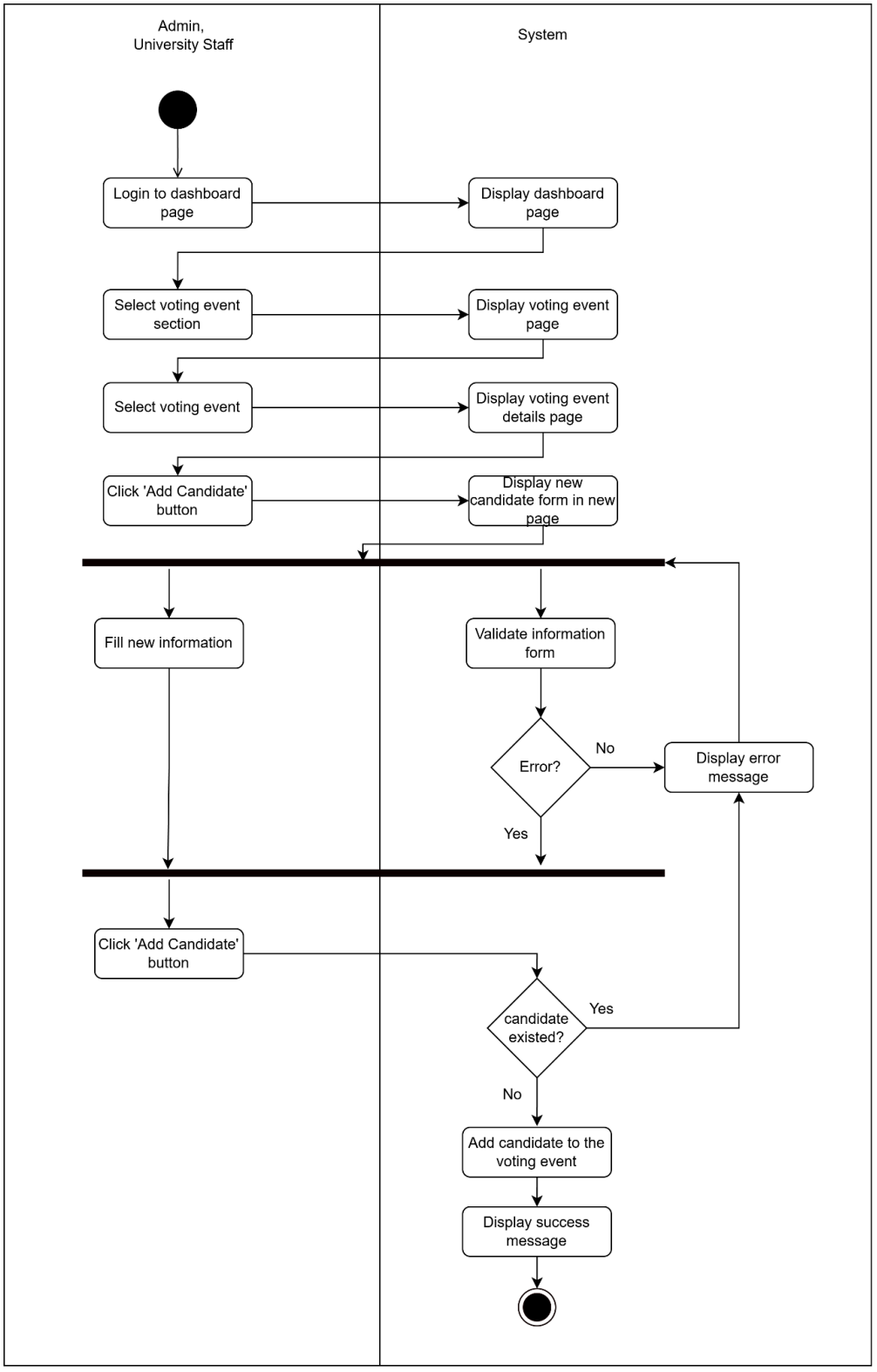


Figure 4.25. Add candidate

Edit candidate

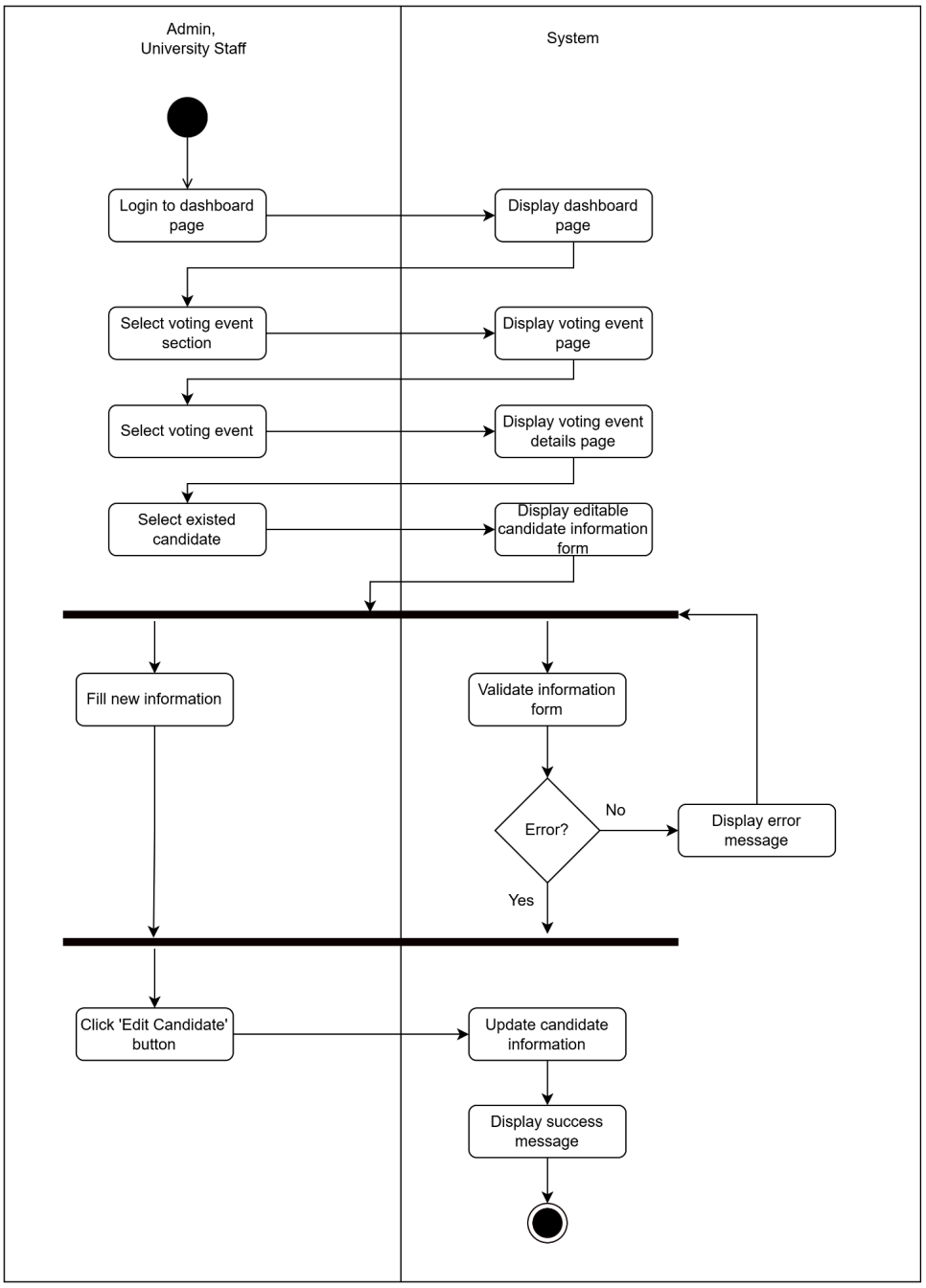


Figure 4.26. Edit candidate

Remove candidate

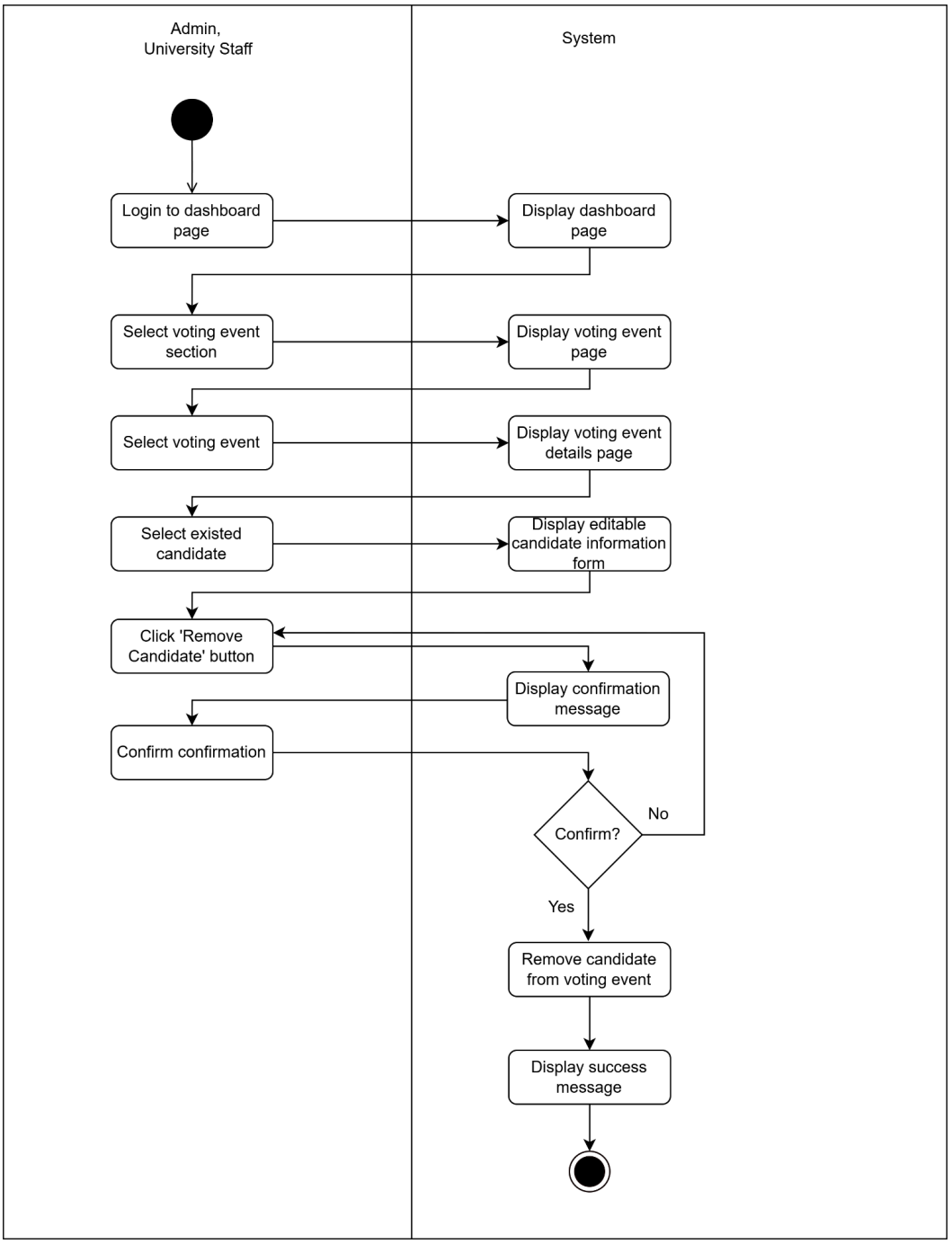


Figure 4.27. Remove candidate

4. Blockchain Integration Module

Trigger blockchain transaction for create voting event

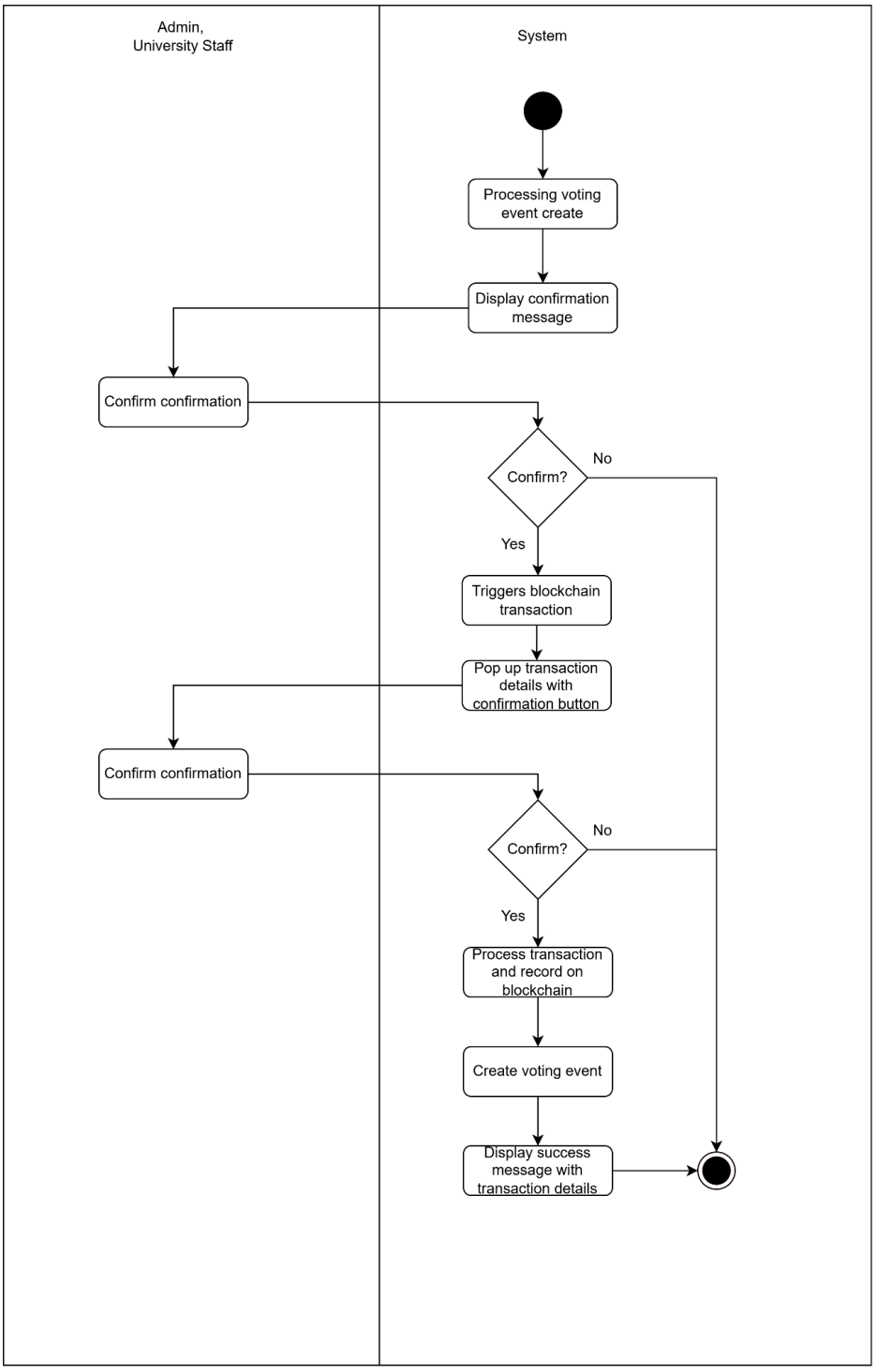


Figure 4.28. Trigger blockchain transaction for create voting event

Trigger blockchain transaction for vote submission

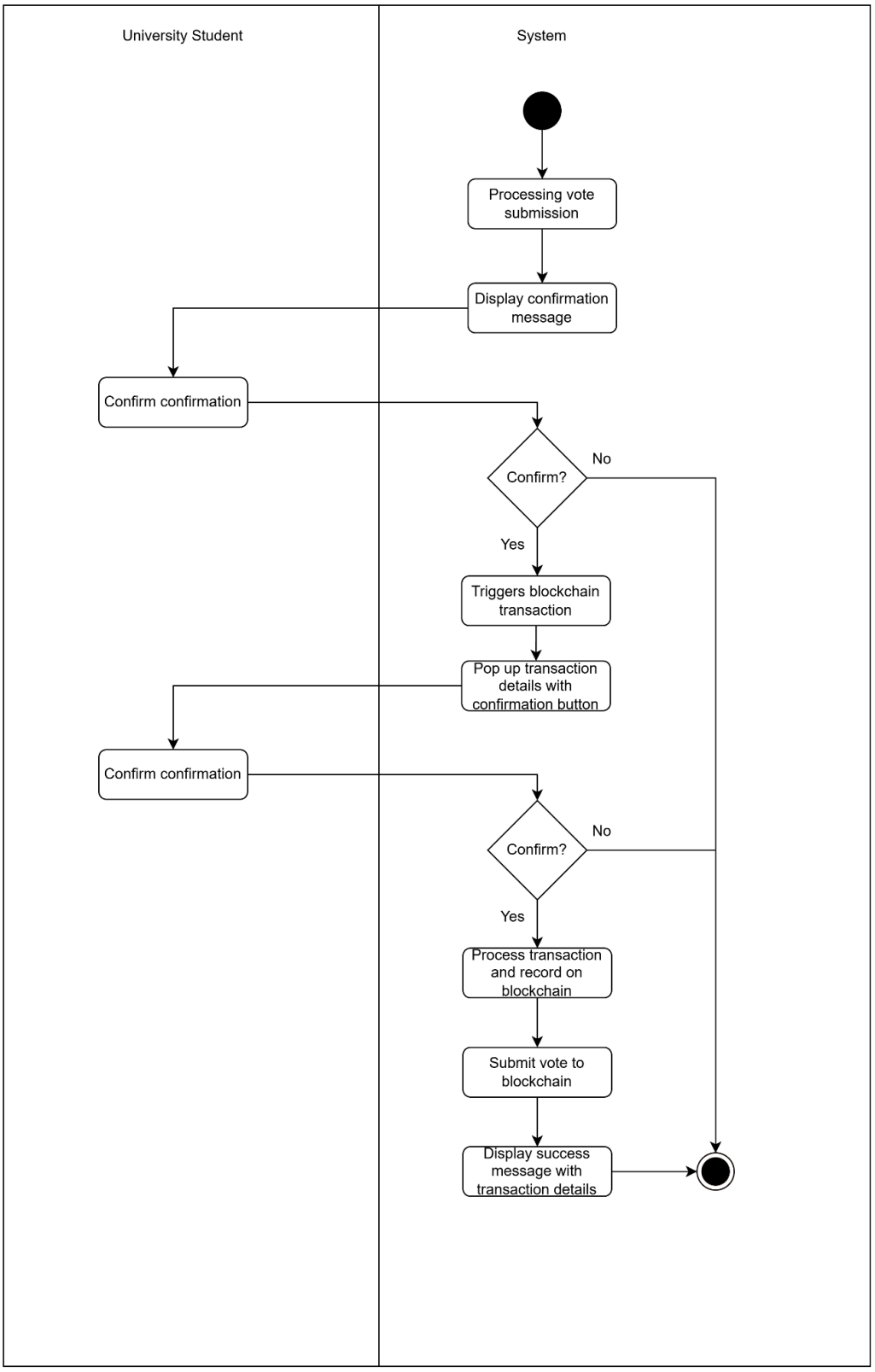


Figure 4.29. Trigger blockchain transaction for vote submission

5. *Voting Inteface Module*

Vote and Confirmation

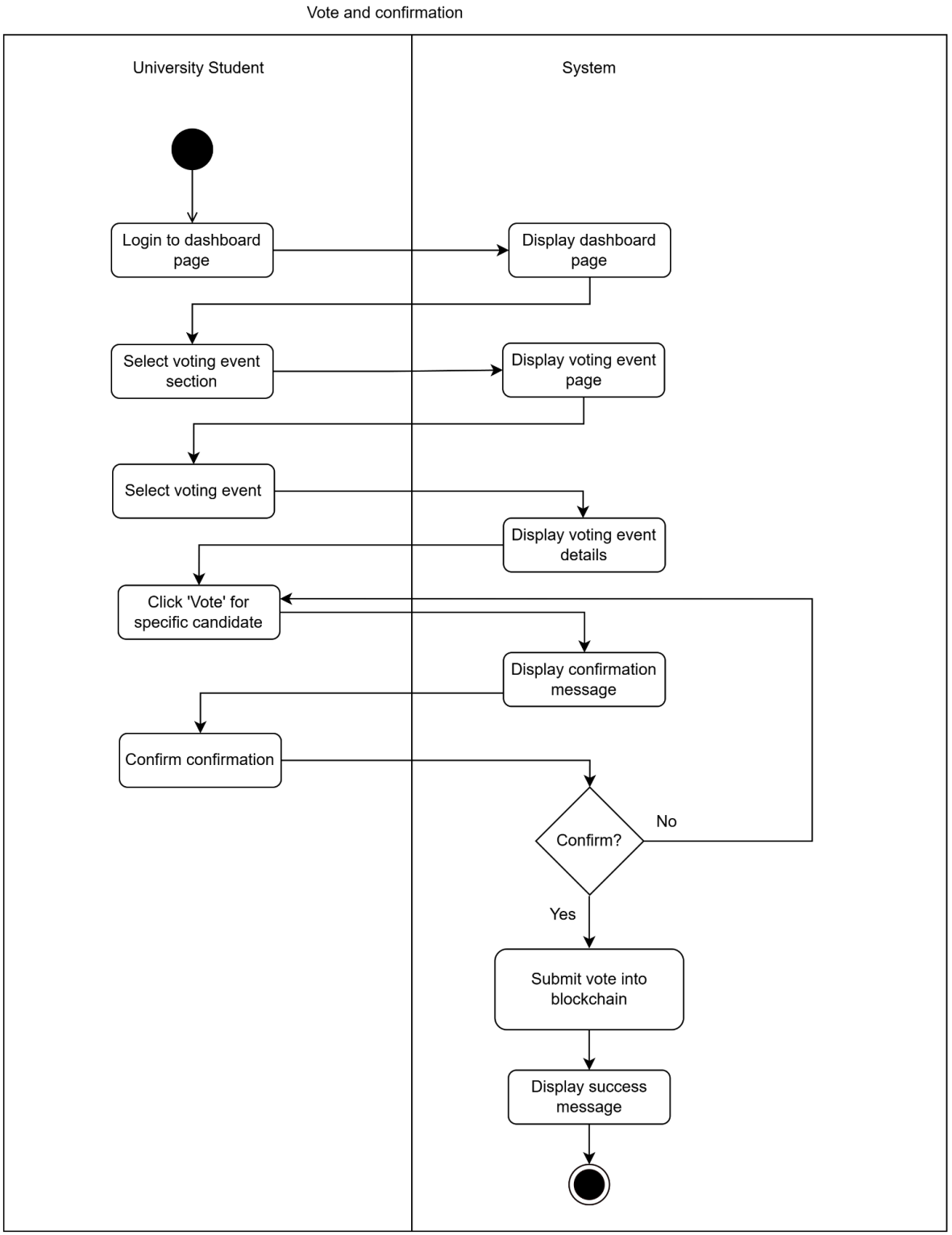


Figure 4.30. Vote and Confirmation

6. Voting Results Module

Export voting results in report

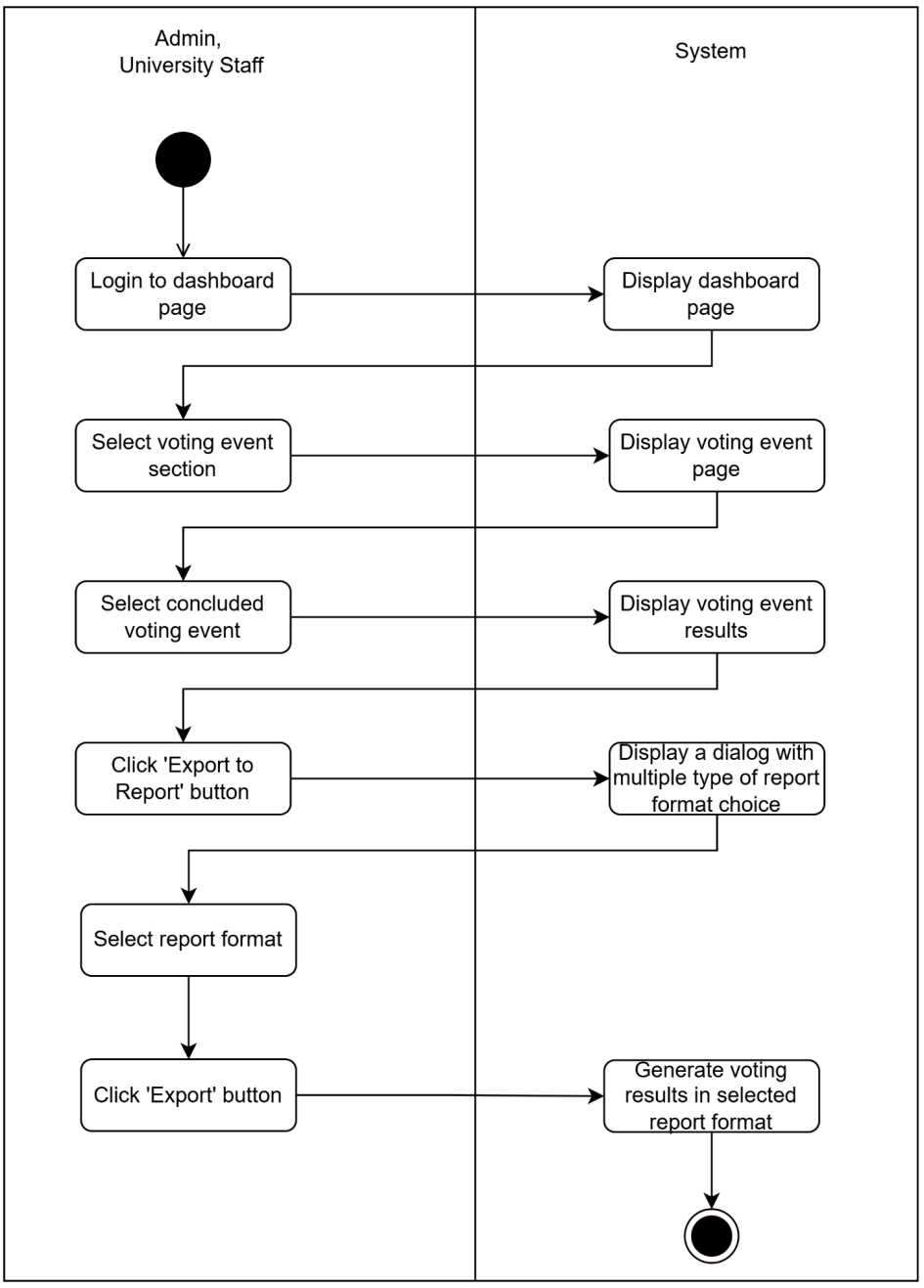


Figure 4.31. Export voting results in report

7. Reporting Module

Generate report

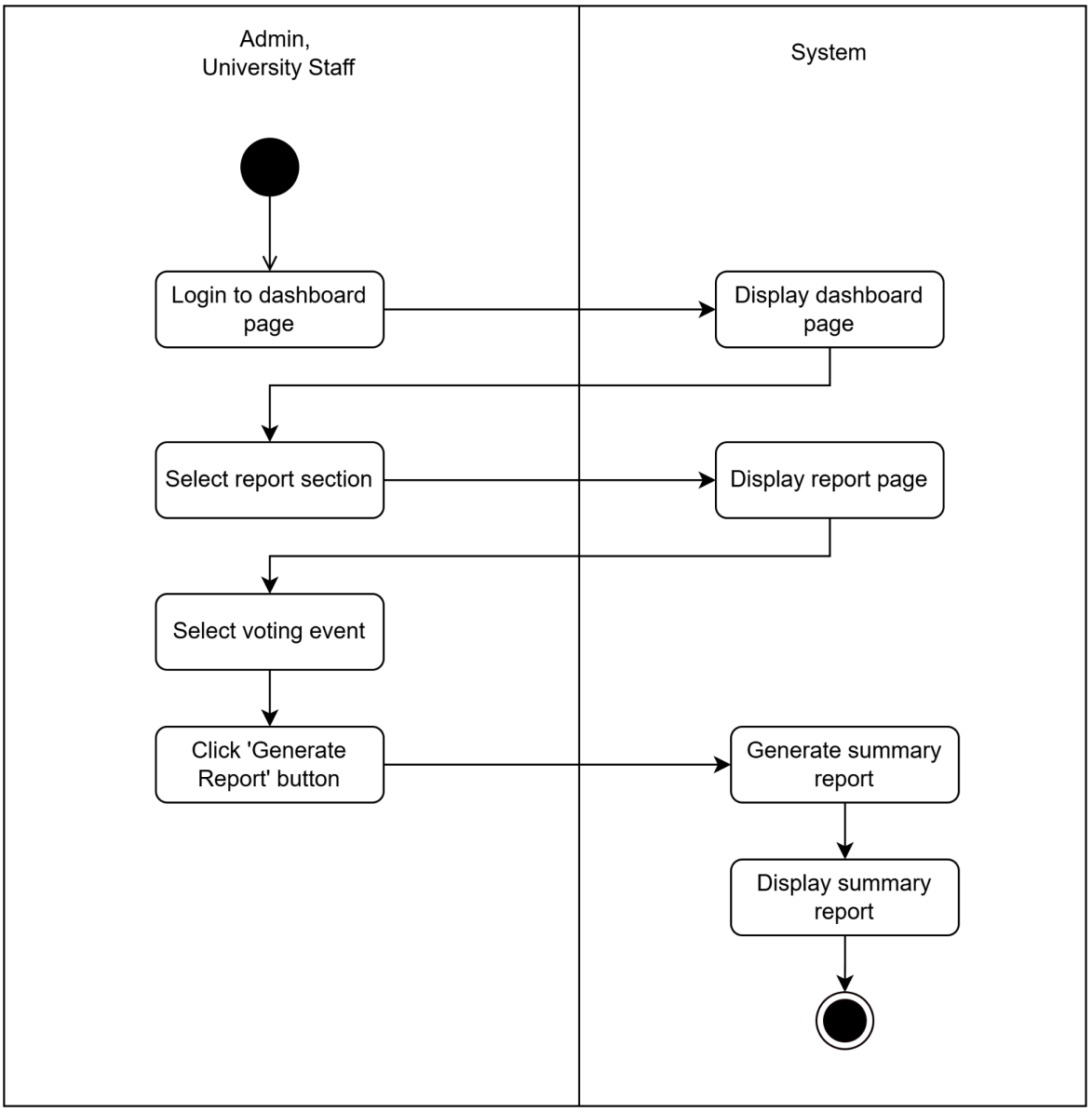


Figure 4.32. Generate report

8. Notifications Module

Send notification

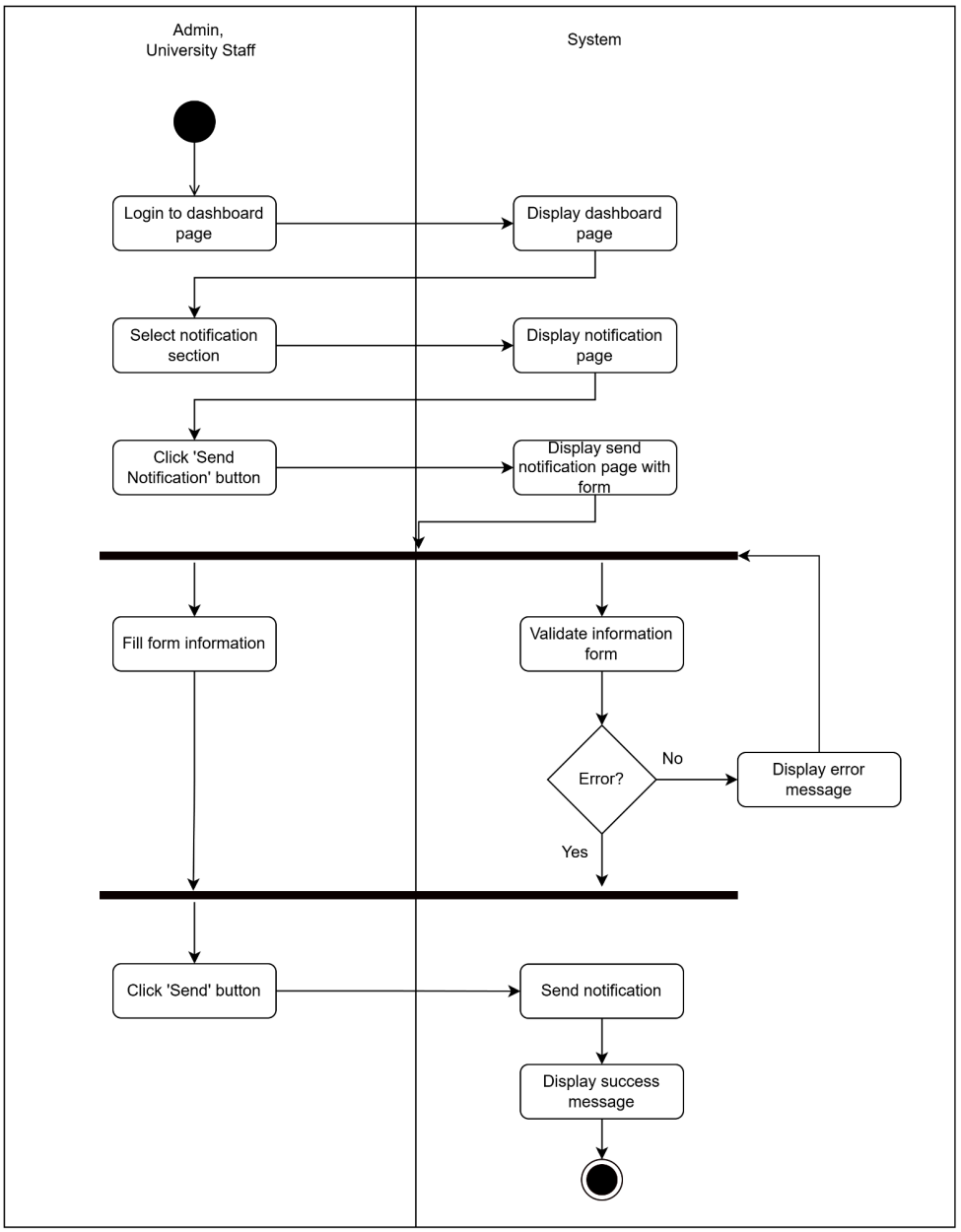


Figure 4.33. Send notification

Set automation reminder notification

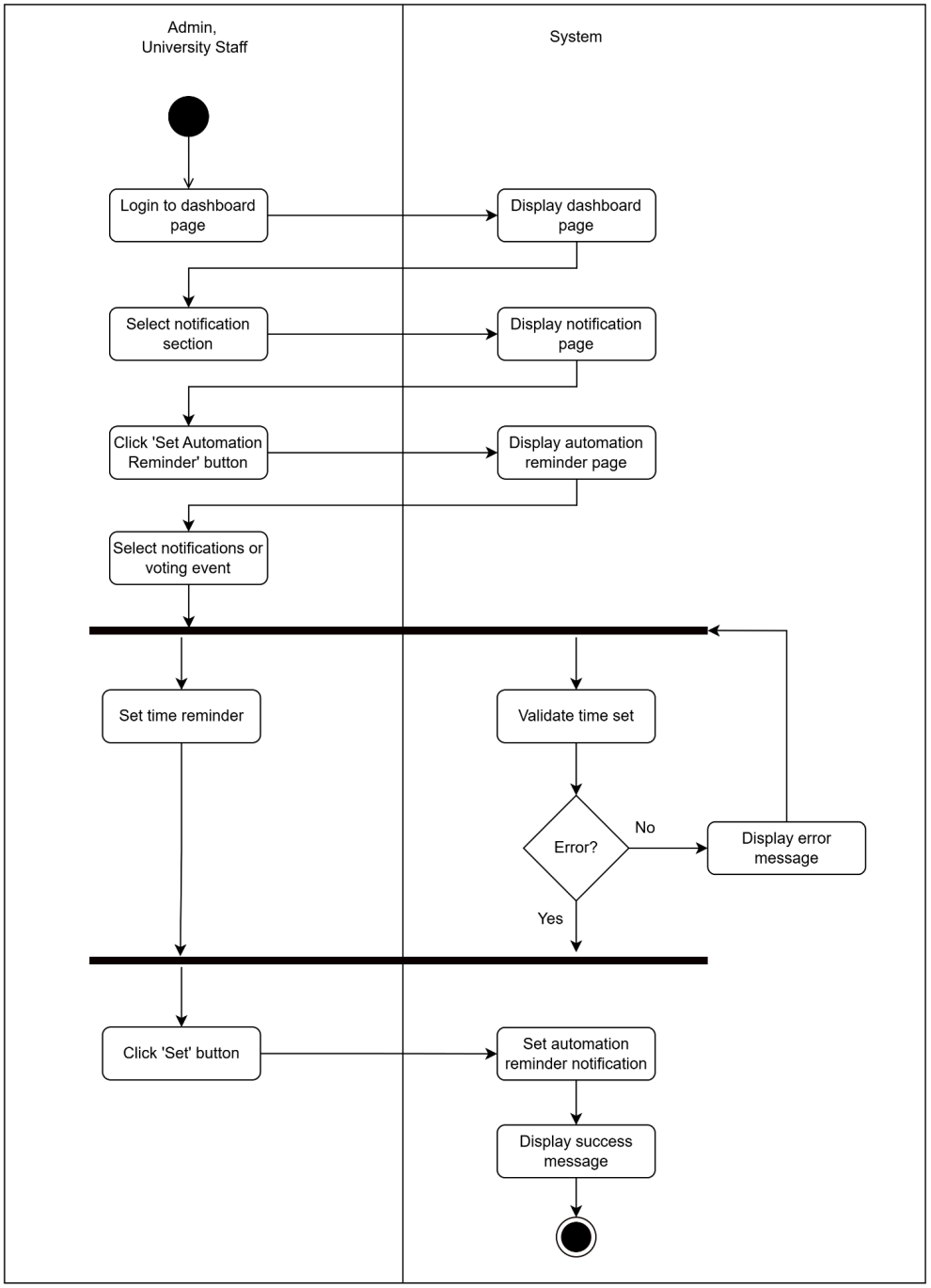


Figure 4.34. Set automation reminder notification

Enable notification

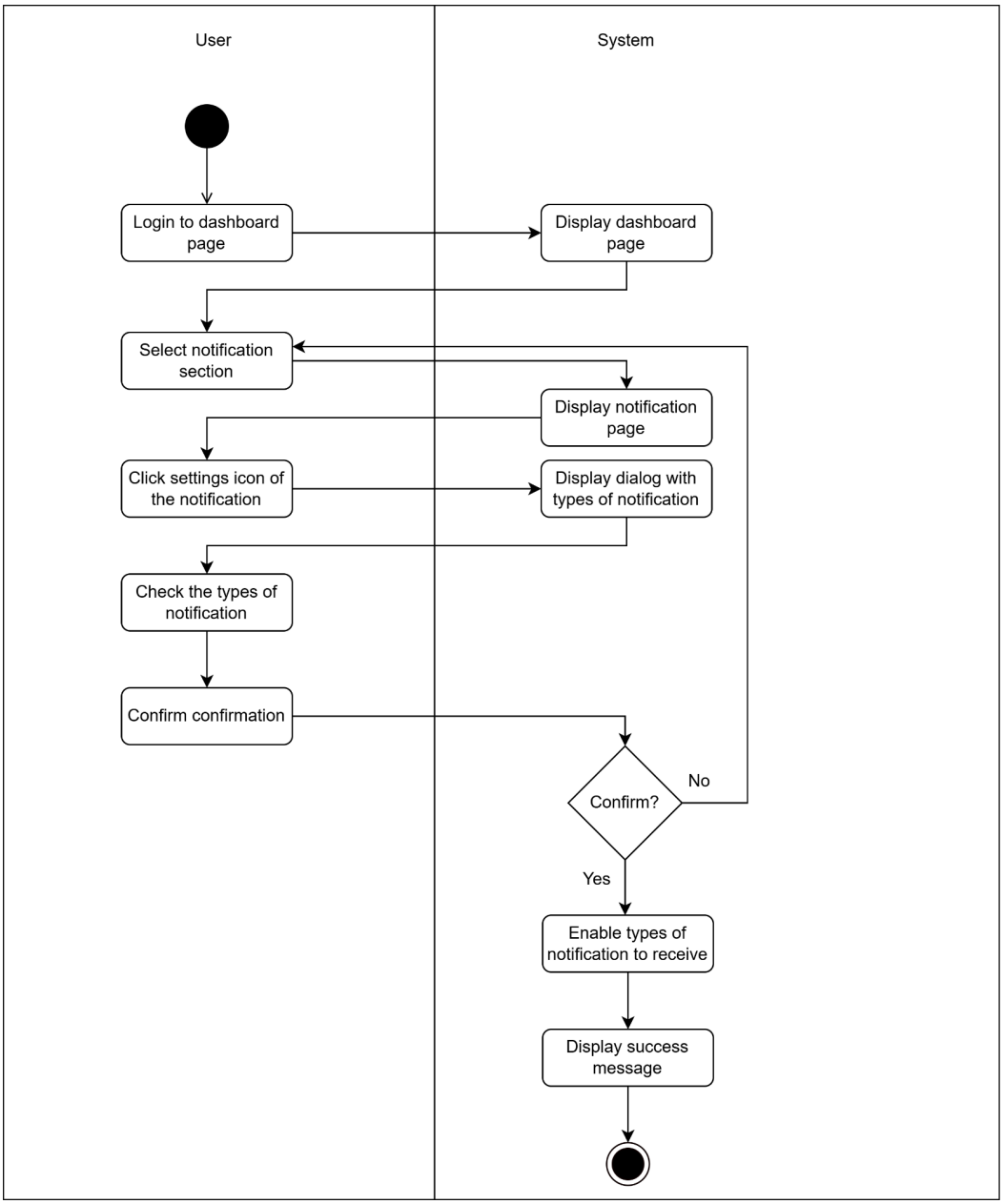


Figure 4.35. Enable notification

Disable notification

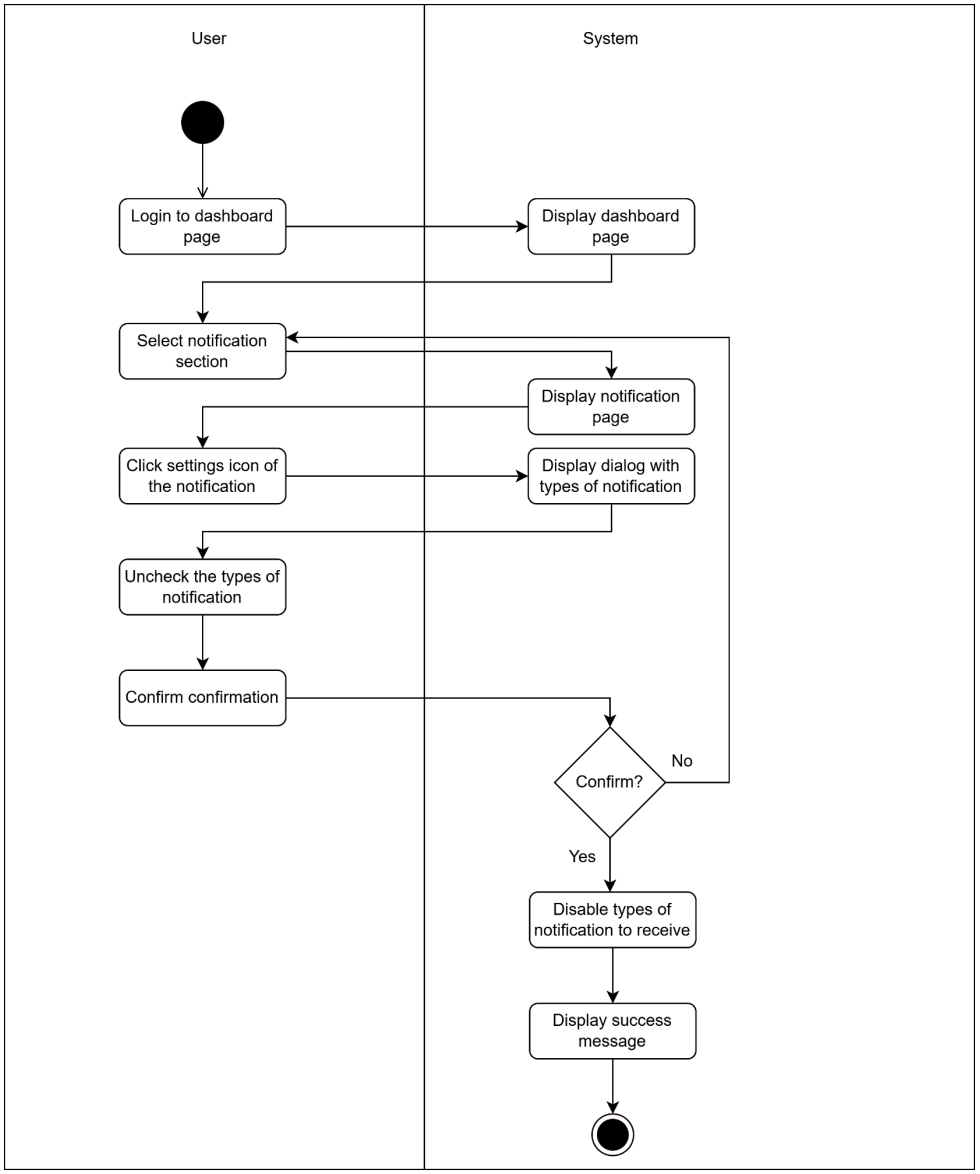


Figure 4.36. Disable notification

4.3 Data Design

4.3.1 Entity Relationship Diagram (ERD)

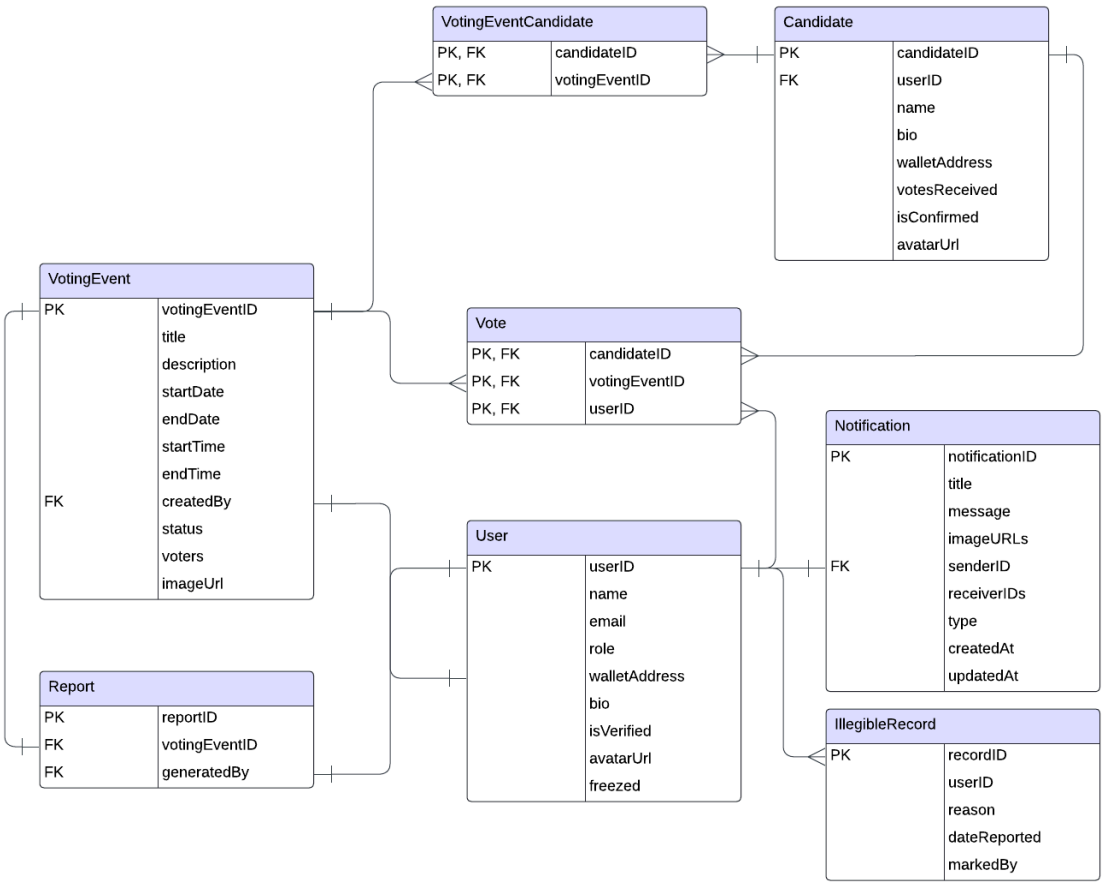


Figure 4.37. Entity Relationship Diagram

4.3.2 Class Diagram

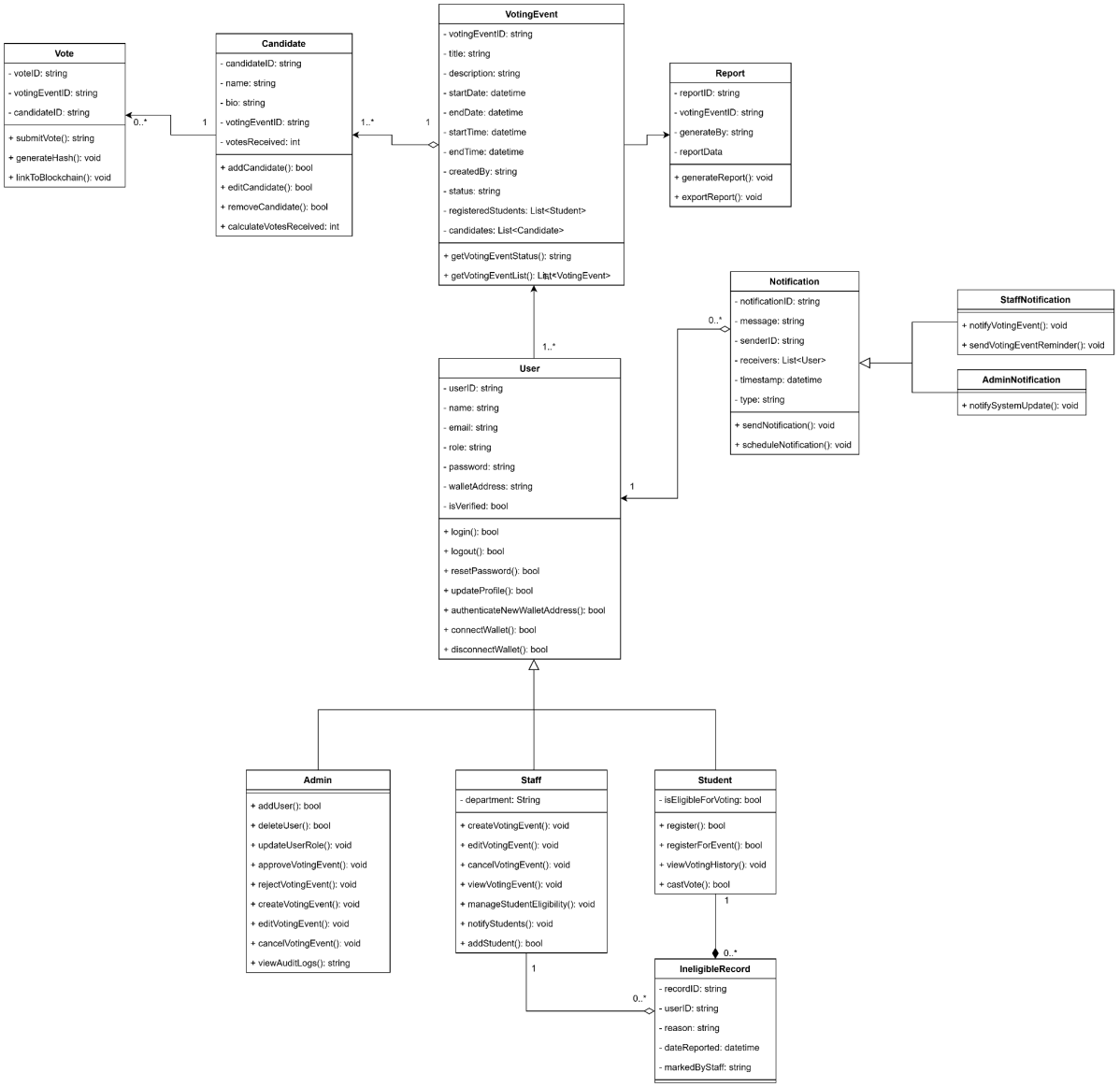
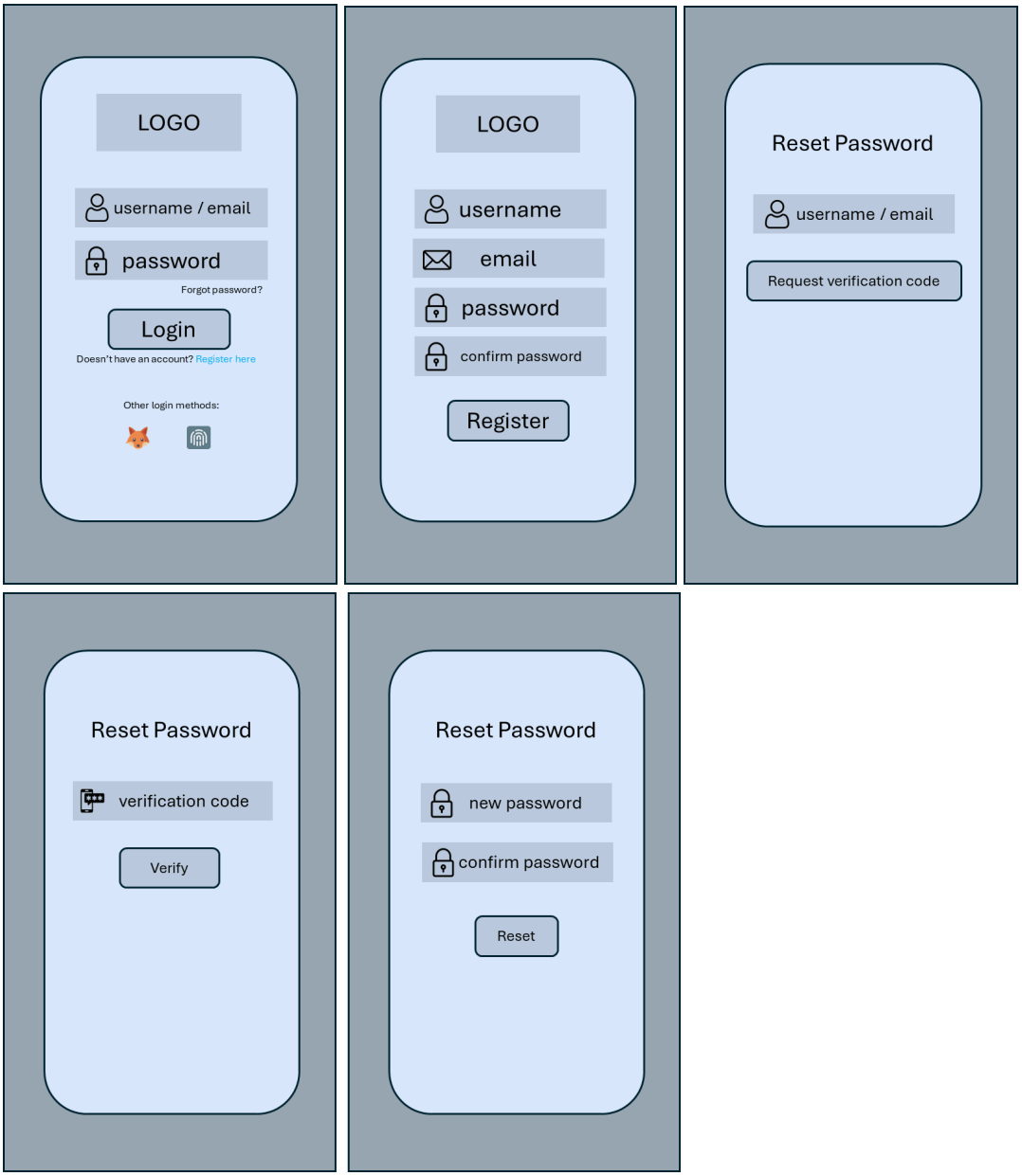
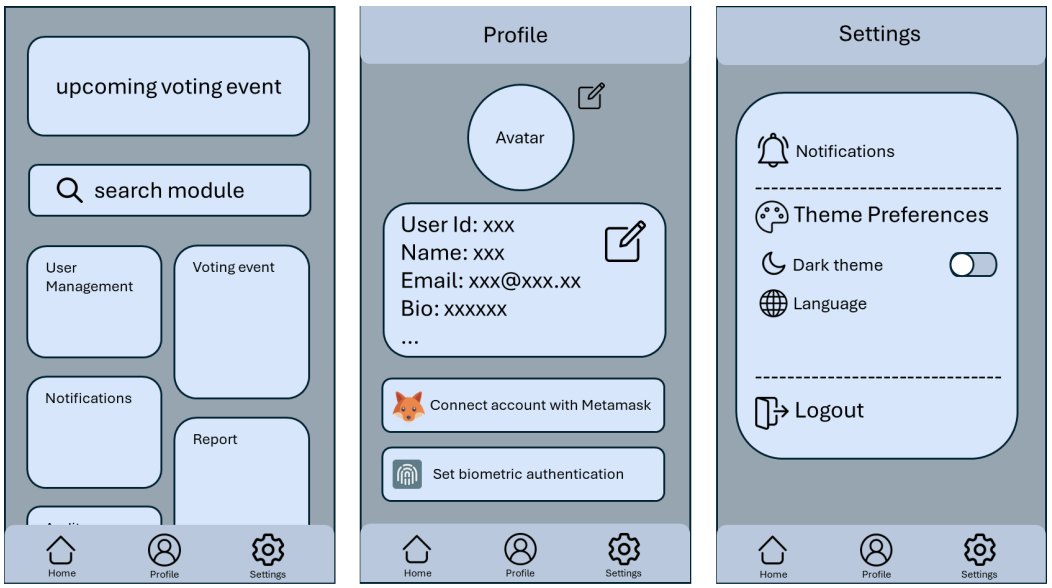


Figure 4.38. Class Diagram

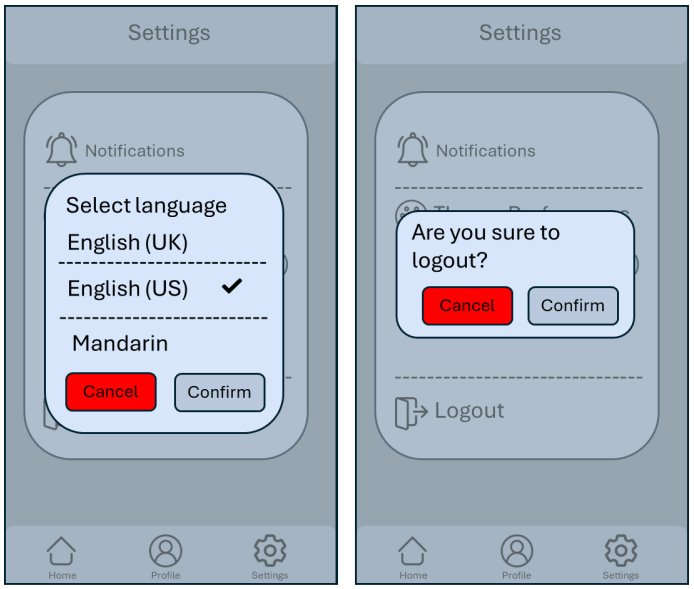
4.4 UI Design



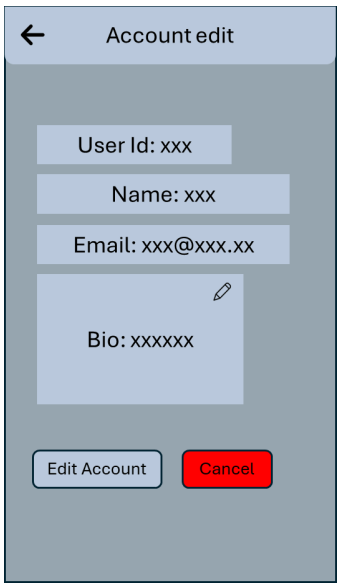
These pages are for the users to login their account. Users can choose one of the method to login. If using credentials, users need to type in their account's username / email and password, then click the login button to proceed, system will process and verify the username / email and password then redirect to home page if matched. User can also choose Metamask or Biometric Authentication to login too. Both function will pop up a dialog from bottom, for Metamask is to connect with Metamask account then login, for Biometric will request user to input the fingerprint to verify and login. Other than that, if user didn't have an account, user can click the 'Register here' blue text to navigate to register page and register account. If user forgot password, user can click the 'Forgot password' text and be navigated to reset password page to reset it.



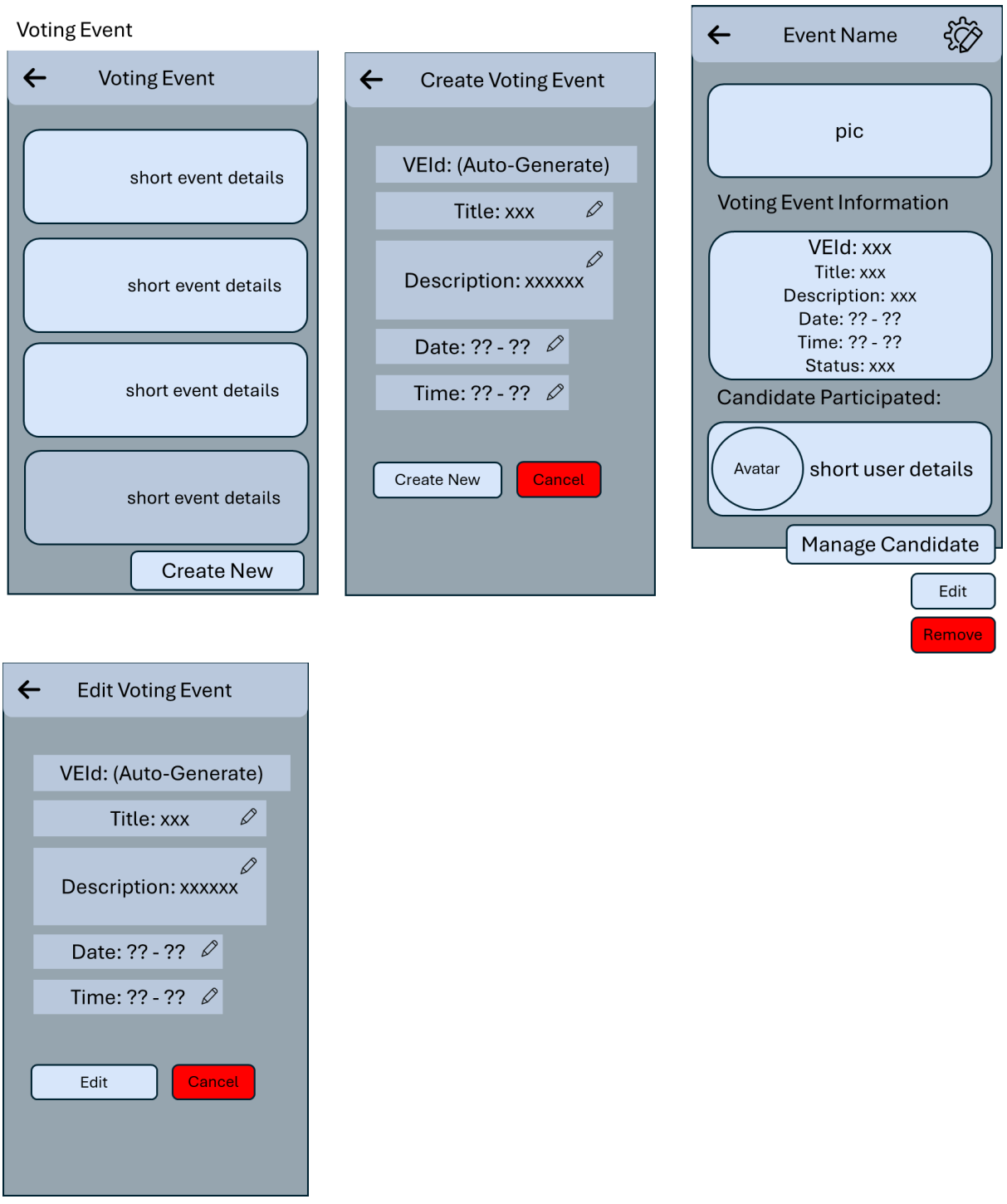
These pages show the dashboard, profile and settings layouts and common functions.



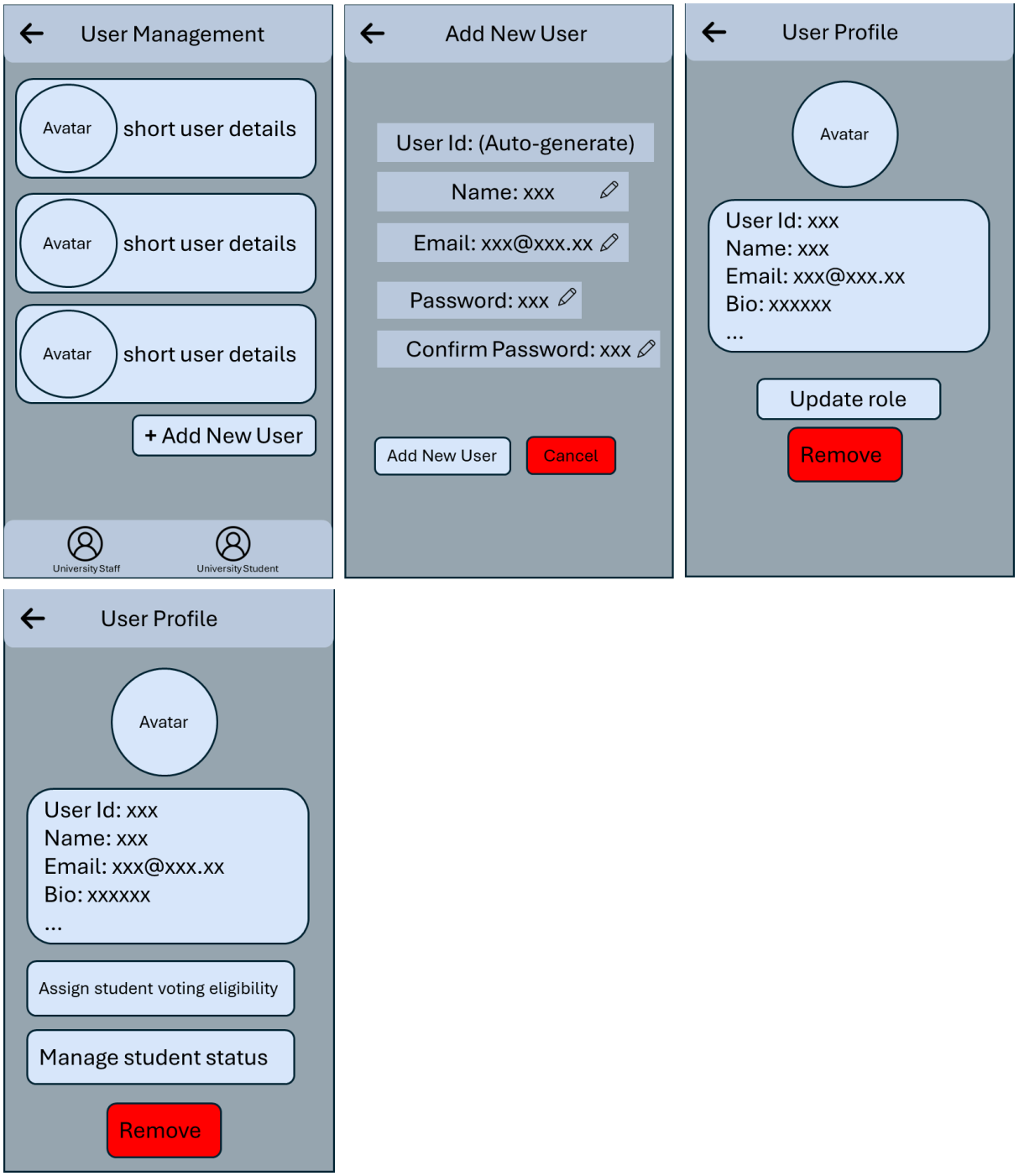
User can click the ‘Language’ tab to open a dialog with multiple language options to select. After selecting the language, user need to confirm the confirmation, then the application will restart and change the language to the selected language. If user clicked the logout button, it will prompt a dialog and request user to confirm it.



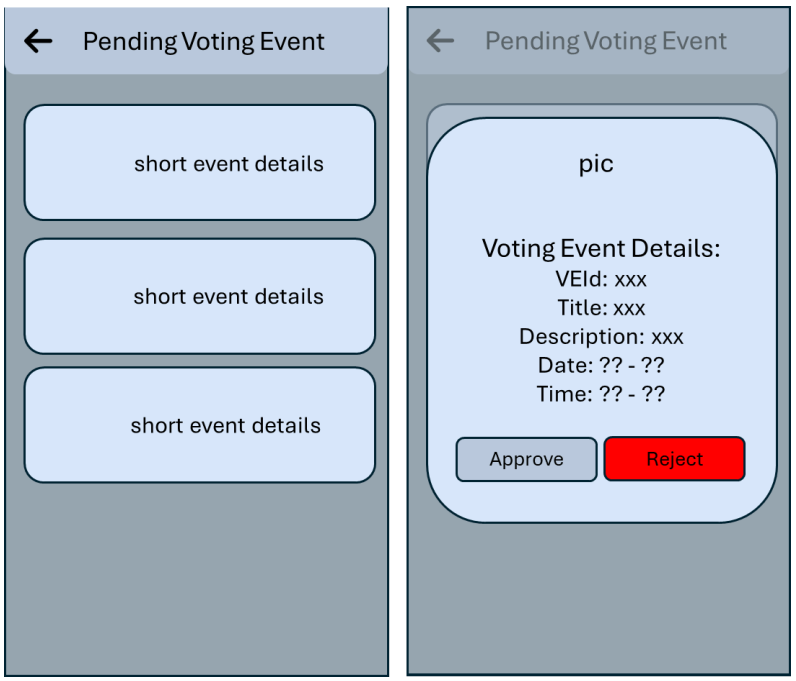
This page shows the account edit stuffs. User can their bio only. User Id is auto generated when account is created and the name and email cannot be changed. (The boxes with the editing icon can be edited) After edited, user need to press ‘Edit Account’ button to confirm changes and update it.



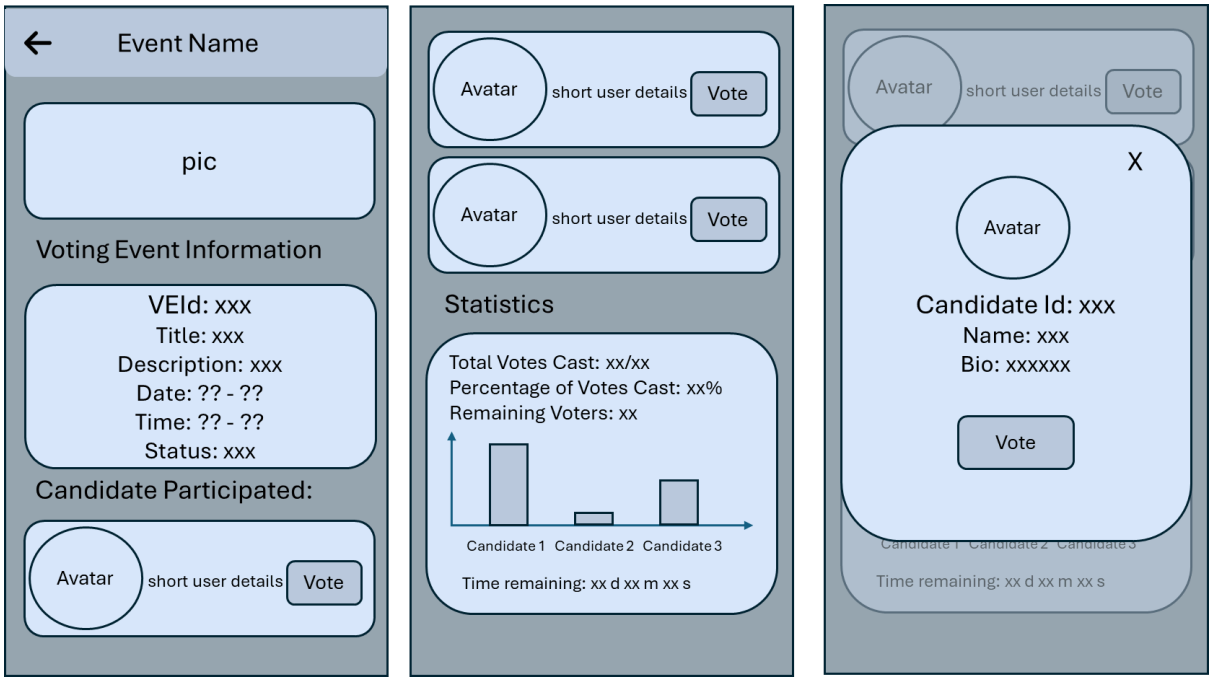
The voting event list layout shows from admin and university staff view (University student does not has the 'Create New' button). The voting event with darker colour background represents the voting event status (pending). Admin and university staff can create the voting event by clicking the 'Create New' button, system will navigate them to the Create Voting Event page, they can type in the voting event details and set the date and time. After it, they need to press the 'Create New' button to create the voting event. The voting event will be created with pending status. All users can click the voting event tab to view the voting event details. In the voting event details page, university student does not have the interact button or icon like manage, edit , remove and customize icon.



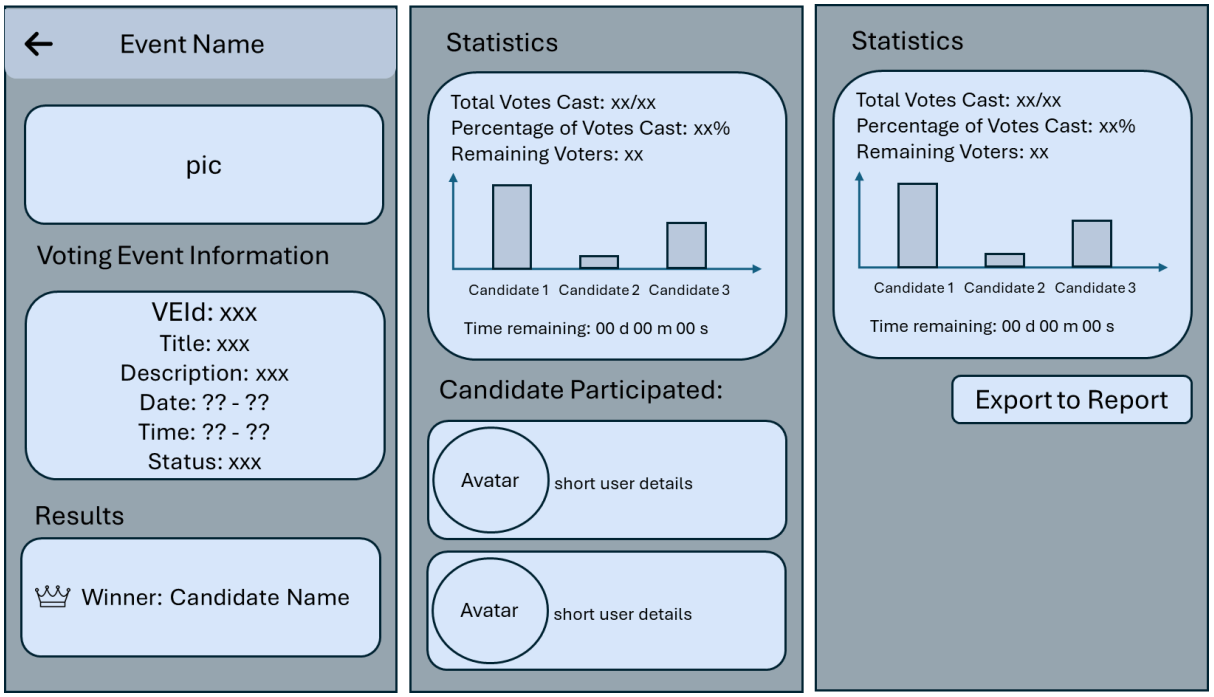
In the user management page only admin and university staff can access. Admin and university staff will have the add new user function button. Admin can add both university staff and student, but university staff only can add student. Admin and university staff can click the '+ Add New User' button to the add new user page to fill in the new user information form and click 'Add New User' button to create new user. Admin and university staff can open the user details (profile) by clicking the user tab in user management page. For admin, admin can update the user account role (student to staff). For university staff, they can assign the eligibility of voting to student and manage the student status (active / deactive / suspend). Both admin and university staff can remove the user account based on their role-based control.



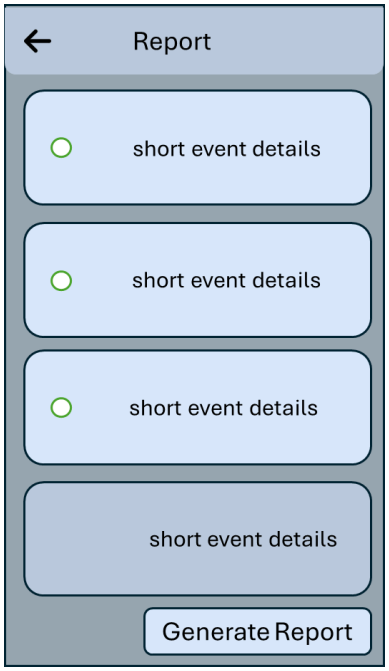
The pending voting event section allow admin to access only. Admin can click on the voting event, it will display a dialog for admin to approve or reject the voting event.



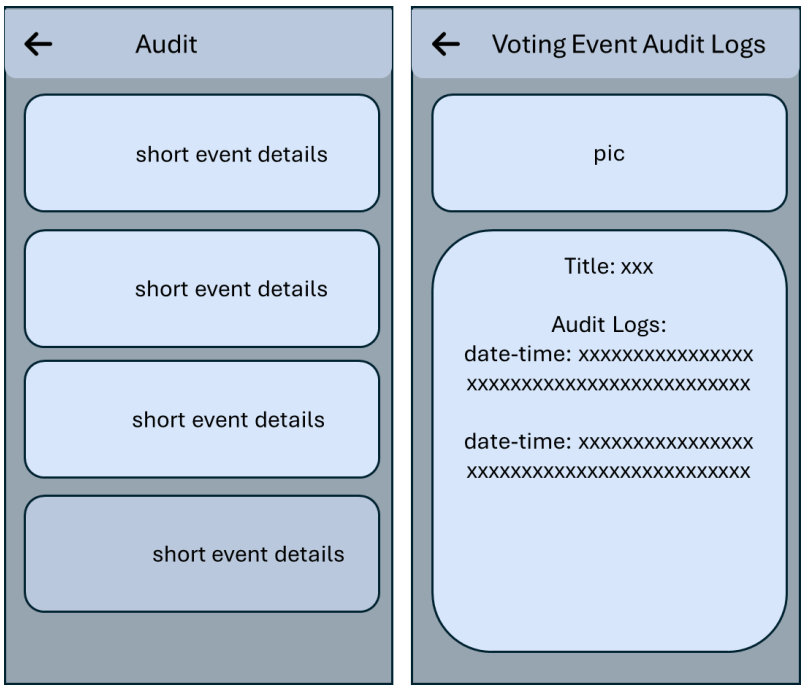
These pages are from university student view. University student can view the voting event details and vote to the candidate they like (when the voting event is progressing).



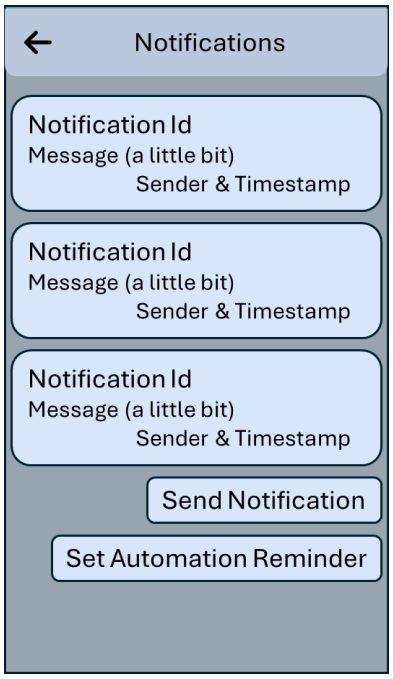
These pages show the voting results after the event had concluded and the winner show there. For admin and university staff, they have the export to report button to export the results to a report in the format they want.



Report page allow admin and university staff to access only. They can select the available voting event, then click the generate report button to generate relevant information in the report format they want.



Audit pages allow admin and university staff only to access. They can click the voting event to be navigated to the related voting event audit logs. The audit logs will show the date, time, action that relevant to the voting event.



All users can access to the notifications page, but university student will not be able to send notification and set automation reminder.

4.5 Software Architecture Design

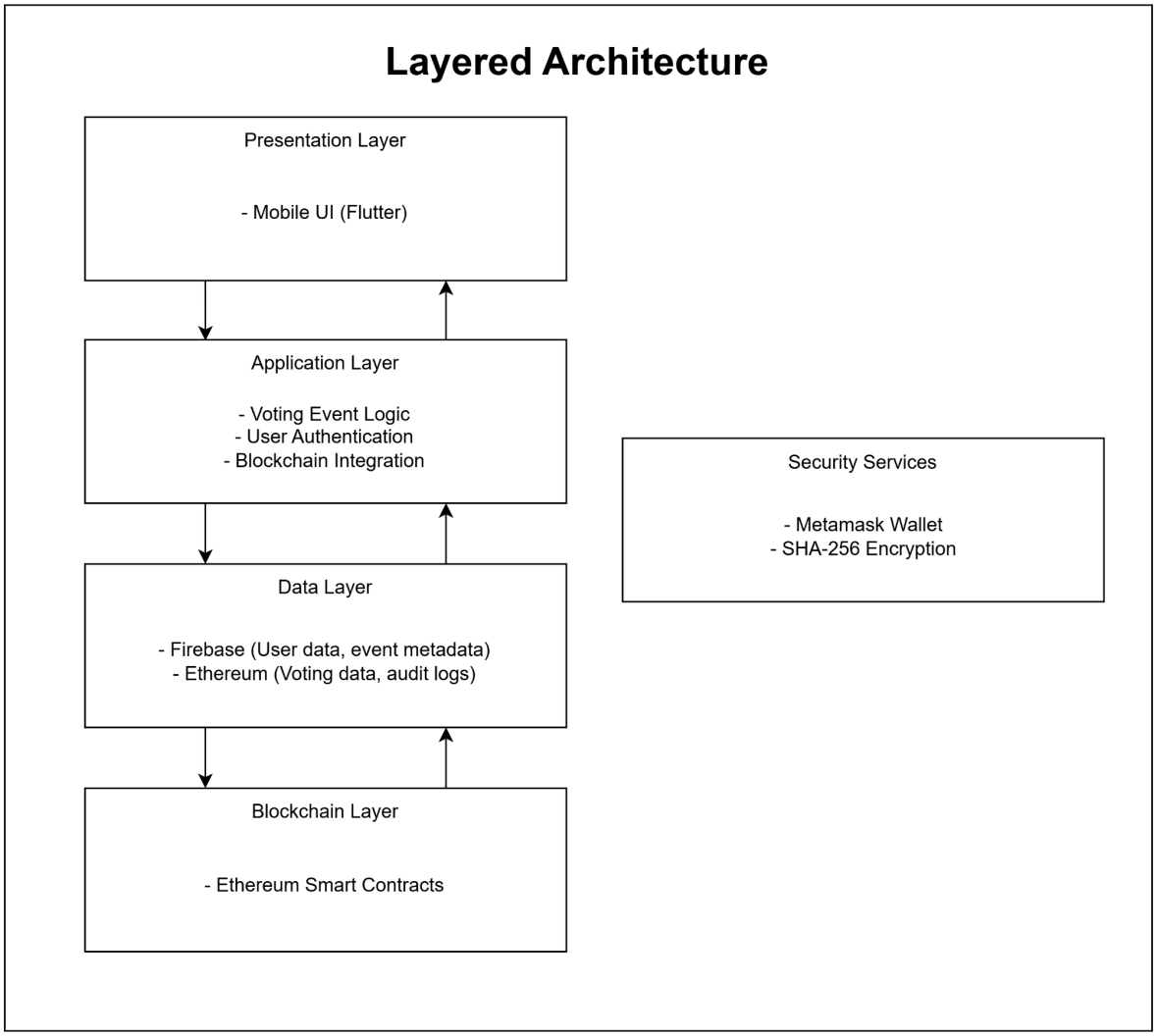


Figure 4.44. Layered Architecture Diagram

4.6 Chapter Summary and Evaluation

This chapter presents the detailed design framework of the system, including authentication, user management, voting event management, blockchain integration, and reporting modules. It combines UML diagrams, database structure, UI design, and software architecture to ensure a comprehensive understanding of the system's workflow and functionality.

The design leverages advanced technologies such as blockchain and biometric authentication to ensure usability, scalability, and security. The evaluation highlighted its effectiveness in meeting the system requirements, but further testing and user feedback are essential for improvement and optimization.

This page is intentionally left blank to indicate the back cover. Ensure that the back cover is black in color.

Chapter 5

Implementation and Testing

2 Implementation and Testing

A short introduction that describes what will be included in this chapter.

IMPORTANT NOTE TO STUDENTS: Include details about:

- **Implementation**

  A detailed description of how you actually carried out the implementation (e.g., coding, etc.) of your system/prototype.

  - Include code snippets and descriptions to show how the requirements of the application/prototype have been met –
    - For Smart Campus projects, code snippets of the MQTT protocol for both client side and server side has to be included with explanation.
  - Include descriptions of important settings - Setting for server, network protocol, IP, database, security
  - Note: this should **not** be a chronological account of the work you carried out.

- **Testing**

  All test cases that have been carried out must be provided - To tabulate the test cases in tables

Note: Students may also opt to split Implementation and Testing into 2 separate chapters.

2.1 Implementation / Coding

Sub-sections should be used to divide the chapter into logical parts.

2.1.1 Smart Contract Integration

Contract Loading Mechanism

```
try {
      // 确保contract为null，避免重复初始化
      contract = null;

      debugPrint("Smart_Contract_Service (_loadContract): Loading ABI file");
      final abiString = await rootBundle.loadString('assets/contracts/contractAbi.json');
      debugPrint("Smart_Contract_Service (_loadContract): ABI file loaded successfully");

      final abiJson = jsonDecode(abiString);
      debugPrint("Smart_Contract_Service (_loadContract): ABI JSON decoded");

      final contractAbi = ContractAbi.fromJson(jsonEncode(abiJson), 'Voting');
      debugPrint("Smart_Contract_Service (_loadContract): ContractAbi created");

      final ethAddress = EthereumAddress.fromHex(contractAddress);
      debugPrint("Smart_Contract_Service (_loadContract): Contract address: $ethAddress");

      contract = DeployedContract(contractAbi, ethAddress);
      debugPrint("Smart_Contract_Service (_loadContract): DeployedContract instance created");

      contractLoaded = true;
      debugPrint("Smart_Contract_Service (_loadContract): Contract loaded successfully");
    } catch (e) {
      debugPrint("Smart_Contract_Service (_loadContract): Error loading contract: $e");
      contractLoaded = false;
      contract = null;
      throw Exception("Failed to load smart contract: $e");
    }
  }
```

- The `_loadContract()` method in `SmartContractService` loads contract ABIs from JSON files in the assets/contracts directory.
- `contractLoaded` boolean flag tracks whether the contract has been successfully loaded.
- The system uses the `DeployedContract` class from the ReOWN AppKit to represent the loaded contract.

Contract Function Execution

```
try {
      // check if the voting event exists
      if (!await checkIfVotingEventExists(candidate.votingEventID)) {
        print("Smart_Contract_Service (voteInBlockchain): Voting event ID '${candidate.votingEventID}' not found in blockchain.");
        return false;
      }

      // vote for the candidate in blockchain
      final bool success = await writeFunction('castVote', [
        candidate.votingEventID,
        candidate.candidateID,
      ]);

      if (!success) {
        print("Smart_Contract_Service (voteInBlockchain): Transaction was rejected or failed.");
        return false;
      }

      print("Smart_Contract_Service (voteInBlockchain): Voted successfully.");
      return true;
    } catch (e) {
      debugPrint("Smart_Contract_Service (voteInBlockchain): $e");
      return false;
    }
  }
```

- `executeFunction(String functionName, List<dynamic> params)` method handles all contract interactions.
- Contract functions including `registerCandidate()`, `vote()`, `getCandidateVotes()`, and `hasVoted()` are exposed through dedicated methods.
- Each function call is wrapped in try-catch blocks for error handling.

State Verification Mechanisms

```
void checkAvailable() {
    if (_appKitModal == null || !_appKitModal!.isConnected) {
      throw Exception("Smart_Contract_Service: AppKitModal is not initialized correctly.");
    }
    if (_appKitModal!.session == null || _appKitModal!.selectedChain == null) {
      throw Exception("Smart_Contract_Service: Session or selectedChain is null");
    }
    if (!contractLoaded || contract == null) {
      throw Exception("Smart_Contract_Service: Contract is not initialized correctly.");
    }
  }
```

```
if (!walletService.isInitialized) {
      debugPrint("Smart_Contract_Service (initialize): WalletConnectService not initialized, initializing now")
;     try {
        await walletService.initialize(context);
      } catch (e) {
        debugPrint("Smart_Contract_Service (initialize): Failed to initialize WalletConnectService: $e");
        throw Exception("Failed to initialize wallet service: $e");
      }
    }

    // 确保AppKitModal可以获取到
    try {
      _appKitModal = walletService.getAppKitModal(context);
      debugPrint("Smart_Contract_Service (initialize): Got AppKitModal successfully");
    } catch (e) {
      debugPrint("Smart_Contract_Service (initialize): Failed to get AppKitModal: $e");

      // 尝试异步获取
      try {
        debugPrint("Smart_Contract_Service (initialize): Trying async method to get AppKitModal");
        _appKitModal = await walletService.getAppKitModalAsync(context);
        debugPrint("Smart_Contract_Service (initialize): Got AppKitModal successfully with async method");
      } catch (asyncError) {
        debugPrint("Smart_Contract_Service (initialize): Failed with async method too: $asyncError");
        throw Exception("Failed to get wallet connection: $e");
      }
    }
```

- `checkAvailable()` method verifies if the contract is loaded and ready for interaction.
- `initialize()` method ensures proper initialization with null check for `_appKitModal`.

Nullability Handling

```
// 重置方法，用于应用重启时清理实例
  static void reset() {
    if (_instance != null) {
      _instance!._reset();
      _instance = null;
    }
  }

  void _reset() {
    try {
      // 清理现有资源
      contractLoaded = false;
      // 清理_appKitModal
      _appKitModal = null;
      debugPrint("SmartContractService: Reset completed, _appKitModal cleared");
    } catch (e) {
      debugPrint("SmartContractService: Error during reset: $e");
    }
  }
```

- The `contract` variable is defined as `DeployedContract?` (nullable) to prevent errors during reinitialization.
- `_reset()` method properly clears resources before reinitialization.
- Debug prints are strategically placed to track initialization and error states.

2.1.2 Blockchain Integration

ReOWN AppKit Integration

```
Future<ReownAppKitModal> _buildAppKitModal(BuildContext context, String projectId) async {
    try {
      return ReownAppKitModal(
        context: rootNavigatorKey.currentContext!,
        projectId: projectId,
        metadata: const PairingMetadata(
          name: 'University Blockchain Voting',
          description: 'App description',
          url: 'https://reown.com',
          icons: ['https://reown.com/logo.png'],
          redirect: Redirect(
            native: 'exampleapp://',
            universal: 'https://reown.com/exampleapp',
            linkMode: true,
          ),
        ),
      );
    } catch (e) {
      debugPrint("WalletConnectService: Error creating ReownAppKitModal: $e");
      throw Exception("Failed to create ReownAppKitModal: $e");
    }
  }

  Future<ReownAppKitModal> getAppKitModalAsync(BuildContext context) async {
    if (!isInitialized || _appkitModal == null) {
      debugPrint("WalletConnectService: AppKitModal not initialized, initializing now...");
      await initialize(rootNavigatorKey.currentContext!);
    }

    if (_appkitModal == null) {
      throw Exception('Failed to initialize WalletConnectService');
    }

    return _appkitModal!;
  }
```

- The system initializes AppKit via `AppKitModal.init()` with custom configuration parameters.
- `BlockchainService` class oversees blockchain connectivity through the `_appKitModal` instance.
- `getAppKit()` method provides access to the AppKit instance with proper null checking.

Event Listening

```dart
Future<void> subscribeToEvents(BuildContext context) async {
    if (_appkitModal == null) {
      debugPrint("WalletConnectService: Cannot subscribe to events, AppKitModal is null");
      return;
    }

    debugPrint("WalletConnectService: Subscribing to events");

    _appkitModal!.onModalConnect.subscribe((ModalConnect? event) async {
      final userProvider = Provider.of<UserProvider>(rootNavigatorKey.currentContext!, listen: false);
      final user = userProvider.user;

      // userviewmodel got user, means user is logged in
      if (user != null) {
        // if user's wallet address is not the same as the event's address
        if (user.walletAddress == "" || user.walletAddress.isEmpty) {
          // update wallet address in provider and firestore
          print("WalletConnectService: User's wallet address is empty, updating wallet address");
          await userProvider.updateUser(userProvider.user!.copyWith(walletAddress: getWalletAddress(context)));
          updateWalletAddress(context);
          SnackbarUtil.showSnackBar(context, AppLocale.walletConnected.getString(context));
          return;
        } else if (user.walletAddress != "" && user.walletAddress.isNotEmpty && user.walletAddress != getWalletAddress(context)) {
          // prompt error and disconnect wallet
          print("WalletConnectService: User's wallet address is not the same as the event's address");
          handleDisconnect(context, false);
          SnackbarUtil.showSnackBar(context, "Wallet address is not the same as the event's address");
          return;
        } else {
          print("WalletConnectService: User's wallet address is the same as the event's address");
          updateWalletAddress(context);
          SnackbarUtil.showSnackBar(context, AppLocale.walletConnected.getString(context));
          return;
        }
      }
      print("WalletConnectService: User is not logged in, logging in with Metamask");
      updateWalletAddress(context);
      authService.loginWithMetamask(context);
    });

    _appkitModal!.onModalDisconnect.subscribe((ModalDisconnect? event) {
      if (_appkitModal!.isConnected) {
        handleDisconnect(context, true);
      }
    });

    _appkitModal!.onModalError.subscribe((ModalError? event) {
      if (event?.message.contains('expired') ?? false) {
        handleDisconnect(context, true);
      }
    });

    _appkitModal!.onModalNetworkChange.subscribe((ModalNetworkChange? event) {});
    _appkitModal!.onModalUpdate.subscribe((ModalConnect? event) {});
  }
```

- Checks user login status and wallet address validity.
- Updates or disconnects wallet based on address comparison.
- Monitors wallet disconnection events.
- Calls handleDisconnect to manage disconnection state.
- Subscribes to error events for expired sessions.

2.1.3 Firebase Integration

User Authentication

```dart
Future<void> loginWithCredentials(context, String emailOrUsername, String password) async {
    try {
      // check the input is email or username
      final bool isEmail = emailOrUsername.contains('@');

      // 1. verify with firebase authentication
      if (isEmail) {
        auth_user.UserCredential userCredential = await _auth.signInWithEmailAndPassword(
          email: emailOrUsername,
          password: password,
        );
        if (userCredential.user != null) {
          await _verifyUserInFirestore(context, userCredential.user!.uid);
        }
      } else {
        // 2. login with username
        final String? foundEmail = await _getEmailByUsername(emailOrUsername);
        if (foundEmail == null) {
          // username not found in any role
          SnackbarUtil.showSnackBar(context, AppLocale.userNotFound.getString(context));
          return;
        }

        // If found, sign in using the discovered email
        auth_user.UserCredential userCredential = await _auth.signInWithEmailAndPassword(
          email: foundEmail,
          password: password,
        );

        if (userCredential.user != null) {
          await _verifyUserInFirestore(context, userCredential.user!.uid);
        }
      }
    } on auth_user.FirebaseAuthException catch (e) {
      SnackbarUtil.showSnackBar(context, 'Login failed: ${e.message}');
    } catch (e) {
      // Add error handling to show user feedback
      SnackbarUtil.showSnackBar(context, 'Login failed: $e');
    }
  }
```

```dart
Future<void> _verifyUserInFirestore(context, String userId) async {
    bool userFound = false;
    model_user.User? user;

    List<model_user.UserRole> roles = model_user.UserRoleExtension.getAllUserRoles();

    for (model_user.UserRole role in roles) {
      DocumentSnapshot userDoc = await FirebasePathUtil.getUserCollection(role).doc(userId).get();

      if (userDoc.exists) {
        var userData = userDoc.data() as Map<String, dynamic>;

        user = model_user.User(
          userID: userData['userID'],
          name: userData['username'],
          email: userData['email'],
          role: role,
          walletAddress: userData['walletAddress'],
          bio: userData['bio'],
          isVerified: userData['isVerified'],
          avatarUrl: userData['avatarUrl'],
          freezed: userData['freezed'],
        );

        final UserProvider userProvider = Provider.of<UserProvider>(context, listen: false);

        if (role == model_user.UserRole.staff) {
            print("user is staff");
            userProvider.setDepartment(userData['department']);
          } else if (role == model_user.UserRole.student) {
            print("user is student");
            print("isEligibleForVoting: ${userData['isEligibleForVoting']}");
            userProvider.setIsEligibleForVoting(userData['isEligibleForVoting']);
          } else {
            print("user is admin");
          }

        setUserAndLoginAndNavigate(context, user);
        userFound = true;
        break;
      }
    }

    if (!userFound) {
      SnackbarUtil.showSnackBar(context, AppLocale.userNotFound.getString(context));
    }
  }
```

```dart
Future<void> registerWithCredentials(
    BuildContext context,
    String username,
    String email,
    String password, [
    String walletAddress = '',
    model_user.UserRole role = model_user.UserRole.student,
    String department = 'General',
  ]) async {
    try {
      bool registerSuccess = false;
      // check if username or email is already taken
      final bool usernameExists = await _isUsernameTaken(username);
      if (usernameExists) {
        SnackbarUtil.showSnackBar(context, 'Username is already taken');
        return;
      }

      final bool emailExists = await _isEmailTaken(email);
      if (emailExists) {
        SnackbarUtil.showSnackBar(context, 'Email is already taken');
        return;
      }

      // create user in firestore + firebase authentication
      if (role == model_user.UserRole.staff) {
        print('Creating staff user with department: $department');
        registerSuccess = await userRepo.createUser(
          Staff(
            userID: '', // will be assigned internally in createUser method
            name: username,
            email: email,
            role: role,
            walletAddress: walletAddress,
            department: department,
            freezed: false,
            avatarUrl: '',
          ),
          password,
        );
        print('Staff user created successfully');
      } else {
        registerSuccess = await userRepo.createUser(
          Student(
            userID: '', // will be assigned internally in createUser method
            name: username,
            email: email,
            role: role,
            walletAddress: walletAddress,
            isEligibleForVoting: true,
            freezed: false,
            avatarUrl: '',
          ),
          password,
        );
        print('Student user created successfully');
      }

      if (!registerSuccess) {
        return;
      }

      // success message
      SnackbarUtil.showSnackBar(context, AppLocale.registrationSuccess.getString(context));

      // if user does not have a wallet address, just navigate to login page
      if (walletAddress.isEmpty) {
        NavigationHelper.navigateToLoginPage(context);
      } else {
        // if the user has a wallet address, you might directly log them in with Metamask
        await loginWithMetamask(context);
        SnackbarUtil.showSnackBar(context, '${AppLocale.walletConnectionSuccessful.getString(context)}!');
      }
    } on auth_user.FirebaseAuthException catch (e) {
      print('Firebase Auth Error: ${e.code} - ${e.message}');
      SnackbarUtil.showSnackBar(context, 'Registration failed: ${e.message}');
    } catch (e) {
      print('Registration Error: $e');
      SnackbarUtil.showSnackBar(context, 'Registration failed: $e');
    }
  }
```

- The AuthService class implements a comprehensive authentication system that handles both traditional and blockchain-based login methods through

loginWithCredentials(email, password) for standard authentication and loginWithMetamask(address, signature) for wallet-based authentication, with robust error handling and session management.

- User registration is managed through registerWithCredentials(email, password, name, role) which supports multiple user roles (student/staff), includes department selection for staff members, and optionally integrates MetaMask wallet connection during the registration process.
- The _verifyUserInFirestore(User user) method performs thorough user verification by checking user existence across different role collections, retrieving additional profile data, and validating user permissions and eligibility status for voting operations.

FCM Integration

```dart
static Future<void> sendNotification(String token, String title, String body) async {
    const String functionUrl = 'https://us-central1-blockchain-voting-system-f1dfe.cloudfunctions.net/sendNotification';

    try {
      final response = await http.post(
        Uri.parse(functionUrl),
        headers: {
          'Content-Type': 'application/json',
        },
        body: json.encode({
          'token': token,
          'title': title,
          'body': body,
        }),
      );

      print("Notification sent to token: $token, Response: ${response.statusCode}");
      if (response.statusCode != 200) {
        print("Error response: ${response.body}");
      }
    } catch (e) {
      print("Error sending notification: $e");
    }
  }

  static Future<void> sendTopicNotification(String topic, String title, String body) async {
    const String functionUrl = 'https://us-central1-blockchain-voting-system-f1dfe.cloudfunctions.net/sendTopicNotification';

    try {
      final response = await http.post(
        Uri.parse(functionUrl),
        headers: {
          'Content-Type': 'application/json',
        },
        body: json.encode({
          'topic': topic,
          'title': title,
          'body': body,
        }),
      );

      print("Notification sent to topic: $topic, Response: ${response.statusCode}");
      if (response.statusCode != 200) {
        print("Error response: ${response.body}");
      }
    } catch (e) {
      print("Error sending topic notification: $e");
    }
  }
```

- `NotificationService` handles push notification setup and reception.
- `requestNotificationPermission()` ensures proper permissions are granted.
- Topics like `voting_events` and `results` allow targeted notifications based on user interests.

Firebase Storage Integration

```dart
// upload image to Firebase Storage and return the download URL
  Future<String> uploadVotingEventImage(File image, String votingEventId) async {
    try {
      print('VotingEventRepository: Uploading image for voting event: $votingEventId');

      // create a unique filename using timestamp + event ID
      final String fileName = '${DateTime.now().millisecondsSinceEpoch}_$votingEventId.jpg';
      final String path = 'voting_events/$votingEventId/$fileName';

      // get the storage reference
      final Reference storageRef = FirebaseStorage.instance.ref().child(path);

      // upload the image
      final UploadTask uploadTask = storageRef.putFile(image);

      // await the completion of the upload and get the download URL
      final TaskSnapshot taskSnapshot = await uploadTask;
      final String downloadUrl = await taskSnapshot.ref.getDownloadURL();

      print('VotingEventRepository: Image uploaded. URL: $downloadUrl');
      return downloadUrl;
    } catch (e) {
      print('VotingEventRepository (uploadVotingEventImage): Error uploading image: $e');
      return '';
    }
  }

  // delete image from Firebase Storage
  Future<bool> deleteVotingEventImage(String imageUrl) async {
    try {
      if (imageUrl.isEmpty) {
        return true; // nothing to delete
      }

      print('VotingEventRepository: Deleting image at URL: $imageUrl');

      // extract the reference from the URL
      final Reference storageRef = FirebaseStorage.instance.refFromURL(imageUrl);

      // delete the file
      await storageRef.delete();

      print('VotingEventRepository: Image deleted successfully');
      return true;
    } catch (e) {
      print('VotingEventRepository (deleteVotingEventImage): Error deleting image: $e');
      return false;
    }
  }
```

- The uploadVotingEventImage function handles secure image uploads to Firebase Storage with unique filename generation and returns downloadable URLs for voting event images.
- The deleteVotingEventImage function manages the cleanup of voting event images by safely removing files from Firebase Storage using their URL references.
- Both functions implement comprehensive error handling and logging to ensure reliable storage operations and maintain organized file structure in the voting_events directory.

2.1.4 Voting Implementation

Candidate Registration Function

```dart
Future<bool> addCandidatesToVotingEvent(VotingEvent votingEvent) async {
    print("Smart_Contract_Service (addCandidatesToVotingEvent): Adding candidates to voting event in blockchain.");

    try {
      if (!await checkIfVotingEventExists(votingEvent.votingEventID)) {
        print("Smart_Contract_Service (addCandidatesToVotingEvent): Voting event ID '${votingEvent.votingEventID}' not found in blockchain.");
        return false;
      }

      // add candidates to voting event in blockchain
      final List<String> candidateIDs = votingEvent.candidates.map((candidate) => candidate.candidateID).toList();
      final List<EthereumAddress> candidateWalletAddresses = votingEvent.candidates.map((candidate) {
        try {
          // ensure the address format is correct (starts with 0x)
          String address = candidate.walletAddress;
          if (!address.startsWith('0x')) {
            address = '0x$address';
          }
          return EthereumAddress.fromHex(address);
        } catch (e) {
          print("Error converting address ${candidate.walletAddress}: $e");
          // if the conversion fails, return a zero address or throw an error
          throw Exception("Invalid wallet address format: ${candidate.walletAddress}");
        }
      }).toList();

      // for loop to print the candidate ids and wallet addresses
      for (var i = 0; i < candidateIDs.length; i++) {
        print("Candidate ID: ${candidateIDs[i]}");
        print("Candidate Wallet Address: ${candidateWalletAddresses[i]}");
      }

      final bool success = await writeFunction('addCandidates', [
        votingEvent.votingEventID,
        candidateIDs,
        candidateWalletAddresses,
      ]);

      // check if transaction was successful
      if (!success) {
        print("Smart_Contract_Service (addCandidatesToVotingEvent): Transaction was rejected or failed.");
        return false;
      }

      print("Smart_Contract_Service (addCandidatesToVotingEvent): Candidates added successfully.");
      return true;
    } catch (e) {
      debugPrint("Smart_Contract_Service (addCandidatesToVotingEvent): $e");
      return false;
    }
  }
```

- Handles bulk candidate registration with proper wallet address validation and formatting
- Maintains synchronization between blockchain and Firebase databases for candidate records
- Implements comprehensive error handling for transaction failures and invalid addresses

```
// add candidates to a voting event
    function addCandidates(
        string memory _eventIDStr,
        string[] memory _candidateIDStrs,
        address[] memory _walletAddresses
    ) public {
        bytes32 _eventID = stringToBytes32(_eventIDStr);
        VotingEvent storage votingEvent = votingEvents[_eventID];

        require(votingEvent.eventID != bytes32(0), "Voting event does not exist");
        require(msg.sender == votingEvent.createdBy, "Only event creator can add candidate");
        require(block.timestamp < votingEvent.startDate, "Event already started, cannot add candidate");
        require(_candidateIDStrs.length == _walletAddresses.length, "Candidate IDs and wallet addresses count mismatch");
        require(_candidateIDStrs.length > 0, "No candidates provided");

        for (uint256 i = 0; i < _candidateIDStrs.length; i++) {
            bytes32 _candidateID = stringToBytes32(_candidateIDStrs[i]);
            address walletAddress = _walletAddresses[i];

            require(walletAddress != address(0), "Invalid wallet address");
            require(_candidateID != bytes32(0), "Invalid candidate ID");
            require(candidatesByEvent[_eventID][_candidateID].walletAddress == address(0), "Candidate already exists");

            Candidate memory newCandidate = Candidate({
                candidateID: _candidateID,
                walletAddress: walletAddress,
                votesReceived: 0
            });

            candidatesByEvent[_eventID][_candidateID] = newCandidate;
            candidateIDsByEvent[_eventID].push(_candidateID);
            candidates[walletAddress] = newCandidate;
            votingEvent.candidates.push(walletAddress);

            // emit candidate assign event
            emit CandidateAssign(_eventID, walletAddress);
        }
    }
```

- Allows the event creator to add multiple candidates before the event starts
- Validates candidate information and ensures no duplicate candidates
- Updates the event's candidate list and emits CandidateAssign events for each new candidate

Voting Event Create Function

```dart
Future<bool> createVotingEvent(
    String title,
    String description,
    DateTime? startDate,
    DateTime? endDate,
    TimeOfDay? startTime,
    TimeOfDay? endTime,
    String walletAddress,
      String userID,
      {File? imageFile}) async {
    print("Voting_Event_Provider: Creating VotingEvent object.");
    String imageUrl = '';

    VotingEvent newVotingEvent = VotingEvent(
      votingEventID: "VE-${_votingEventList.length + 1}",
      title: title,
      description: description,
      startDate: startDate,
      endDate: endDate,
      startTime: startTime,
      endTime: endTime,
      createdBy: walletAddress,
      imageUrl: imageUrl,
    );

    // if we have an image, upload it first
    if (imageFile != null) {
      imageUrl = await _votingEventRepository.uploadVotingEventImage(imageFile, newVotingEvent.votingEventID);
      if (imageUrl.isNotEmpty) {
        // update the voting event with the image URL
        newVotingEvent = newVotingEvent.copyWith(imageUrl: imageUrl);
      }
    }

    bool success = await _votingEventRepository.insertNewVotingEvent(userID, newVotingEvent);
    if (success) {
      _votingEventList.add(newVotingEvent);
      // notify listeners that the list has been updated
      _forceRefreshVotingEvents();
      notifyListeners();
    } else {
      // if insert failed but we uploaded an image, delete it
      if (imageUrl.isNotEmpty) {
        await _votingEventRepository.deleteVotingEventImage(imageUrl);
      }
    }

    return success;
  }
```

```dart
Future<bool> insertNewVotingEvent(String userID, VotingEvent votingEvent) async {
    print("Voting_Event_Repository: Inserting voting event to blockchain and firebase.");

    try {
      // blockchain insertion
      await _smartContractService.createVotingEventToBlockchain(votingEvent);


// firebase insertion (will not insert if transaction is failed)
      bool success = await insertVotingEventToFirebase(votingEvent);

      if (!success) {
        return false;
      }

      // send notification to all users
      await FirebaseService.sendVotingEventCreatedNotification(userID, votingEvent);

      return true;
    } catch (e) {
      debugPrint("Voting_Event_Repository (insertNewVotingEvent): $e");
      return false;
    }
  }
```

```dart
// insert the important details only into blockchain, (id, title, start&end date, createBy, candidates, voters)
  Future<void> createVotingEventToBlockchain(VotingEvent votingEvent) async {
    print("Smart_Contract_Service (createVotingEventToBlockchain): Inserting voting event to blockchain.");

    // if the voting event's id is null, throw an error
    if (votingEvent.votingEventID.isEmpty) {
      throw Exception("Smart_Contract_Service (createVotingEventToBlockchain): Voting event id is null.");
    }

    try {
      // check is the voting event id already exists in blockchain
      final votingEventIDs = await readFunction('getVotingEventIDs');
      if (votingEventIDs.contains(votingEvent.votingEventID)) {
        throw Exception("Smart_Contract_Service (createVotingEventToBlockchain): Voting event id already exists in blockchain.");
      }

      final BigInt startDate = ConverterUtil.dateTimeToBigInt(votingEvent.startDate!);
      final BigInt endDate = ConverterUtil.dateTimeToBigInt(votingEvent.endDate!);

      // if the voting event id does not exist, insert the voting event to blockchain
      final bool success = await writeFunction('createVotingEvent', [
        votingEvent.votingEventID,
        votingEvent.title,
        startDate,
        endDate,
      ]);

      // check if transaction was successful
      if (!success) {
        throw Exception("Smart_Contract_Service (createVotingEventToBlockchain): Transaction was rejected or failed.");
      }

      print("Smart_Contract_Service (createVotingEventToBlockchain): Voting event created successfully.");
    } catch (e) {
      debugPrint("Smart_Contract_Service (createVotingEventToBlockchain): $e");
      // re-throw the exception to let the caller know the operation failed
      throw Exception("Failed to create voting event: $e");
    }
  }
```

- Creates new voting events with unique IDs and proper date/time formatting for Malaysia timezone

- Implements validation checks to prevent duplicate event IDs and ensures proper event parameters
- Manages transaction success verification and provides detailed error logging for debugging

```
// create a new voting event with empty candidates and voters list
    function createVotingEvent(
        string memory _eventIDStr,
        string memory _title,
        uint256 _startDate,
        uint256 _endDate
    ) public {
        bytes32 _eventID = stringToBytes32(_eventIDStr);

        // ensure the event doesn't already exist (using empty eventID as indicator)
        require(votingEvents[_eventID].eventID == bytes32(0), "Voting Event already existed.");

        VotingEvent memory newEvent = VotingEvent({
            eventID: _eventID,
            title: _title,
            startDate: _startDate,
            endDate: _endDate,
            createdBy: msg.sender,
            status: 0, // available by default
            candidates: new address[](0),
            voters: new address[](0)
        });

        // store the event in the mapping
        votingEvents[_eventID] = newEvent;
        // store event id into array
        votingEventIDsList.push(_eventID);

        // emit creation event
        emit VotingEventCreate(_eventID);
    }
```

- Creates a new voting event with a unique event ID, title, start date, and end date
- Stores the event in the blockchain with an empty candidates and voters list
- Emits a VotingEventCreate event to notify the system of the new event creation

Vote Casting Function

```dart
Future<bool> vote(Candidate candidate, User user, VotingEvent votingEvent) async {
    bool success1 = await _smartContractService.voteInBlockchain(candidate);
    bool success2 = false;

    if (success1) {
      votingEvent.voters.add(
        Student(
          userID: user.userID,
          name: user.name,
          email: user.email,
          walletAddress: user.walletAddress,
          role: user.role,
          isVerified: user.isVerified,
          isEligibleForVoting: true, // no need second verification
          freezed: user.freezed,
        ),
      );
      success2 = await updateVotingEventInFirebase(votingEvent);
    }

    return success1 && success2;
  }
```

```dart
Future<bool> voteInBlockchain(Candidate candidate) async {
    print("Smart_Contract_Service (voteInBlockchain): Voting for candidate in blockchain.");

    try {
      // check if the voting event exists
      if (!await checkIfVotingEventExists(candidate.votingEventID)) {
        print("Smart_Contract_Service (voteInBlockchain): Voting event ID '${candidate.votingEventID}' not found in blockchain.");
        return false;
      }

      // vote for the candidate in blockchain
      final bool success = await writeFunction('castVote', [
        candidate.votingEventID,
        candidate.candidateID,
      ]);

      if (!success) {
        print("Smart_Contract_Service (voteInBlockchain): Transaction was rejected or failed.");
        return false;
      }

      print("Smart_Contract_Service (voteInBlockchain): Voted successfully.");
      return true;
    } catch (e) {
      debugPrint("Smart_Contract_Service (voteInBlockchain): $e");
      return false;
    }
  }
```

- The vote function in VotingEventRepository implements a dual-layer voting system that first records votes on the blockchain and then updates Firebase, ensuring data consistency across both platforms while maintaining voter records.
- The voteInBlockchain function in SmartContractService handles the blockchain-specific voting operations by verifying the voting event's existence and

executing the smart contract's castVote function with proper error handling and transaction verification.

- Both functions work together to create a secure voting process where the blockchain transaction must succeed first before updating the Firebase database, preventing any inconsistencies in the voting records.

```
// cast a vote for a candidate in a specific voting event
    function castVote(
        string memory _eventIDStr,
        string memory _candidateIDStr
    ) public {
        bytes32 _eventID = stringToBytes32(_eventIDStr);
        bytes32 _candidateID = stringToBytes32(_candidateIDStr);
        VotingEvent storage votingEvent = votingEvents[_eventID];

        require(votingEvent.eventID != bytes32(0), "Voting event does not exist");
        require(!hasVoted[_eventID][msg.sender], "Already voted in this event");
        require(block.timestamp >= votingEvent.startDate && block.timestamp <= votingEvent.endDate, "Voting not active");

        // Ensure msg.sender is in the voters list
        bool isRegisteredVoter = false;
        for (uint256 i = 0; i < votingEvent.voters.length; i++) {
            if (votingEvent.voters[i] == msg.sender) {
                isRegisteredVoter = true;
                break;
            }
        }
        require(isRegisteredVoter, "You are not registered to vote in this event");

        // Check if candidate exists
        require(candidatesByEvent[_eventID][_candidateID].walletAddress != address(0), "Candidate does not exist");

        // Record vote
        Vote memory newVote = Vote({
            votes: msg.sender,
            candidateID: _candidateID,
            timestamp: block.timestamp
        });

        // Update vote counts and records
        voteCountsByCandidate[_eventID][_candidateID]++;
        candidatesByEvent[_eventID][_candidateID].votesReceived++;
        hasVoted[_eventID][msg.sender] = true;
        votesHistory.push(newVote);
        voteByEvent[_eventID].push(newVote);

        emit VoteCast(_eventID, msg.sender, _candidateID);
    }
```

- Validates the voter's eligibility and the voting event's status before allowing a vote
- Records the vote by updating vote counts and marking the voter as having voted
- Emits a VoteCast event to track the voting transaction on the blockchain

Voting Event Result Calculation

```
// get vote results for an event
    function getVoteResults(string memory _eventIDStr) public view returns (bytes32[] memory candidateIDs, uint256[] memory voteCounts) {
        bytes32 _eventID = stringToBytes32(_eventIDStr);
        bytes32[] memory _candidates = candidateIDsByEvent[_eventID];
        uint256[] memory votes = new uint256[](_candidates.length);

        for (uint256 i = 0; i < _candidates.length; i++) {
            votes[i] = voteCountsByCandidate[_eventID][_candidates[i]];
        }

        return (_candidates, votes);
    }
```

getVoteResults function in voting.sol:

- Returns two arrays: candidate IDs and their corresponding vote counts
- This is the primary function that stores and retrieves the raw voting data from the blockchain

```
Future<List<dynamic>> getVoteResultsFromBlockchain(String votingEventID) async {
    try {
      final voteResults = await readFunction('getVoteResults', [votingEventID]);
      return voteResults; // it should return list of candidate ids and their vote counts
    } catch (e) {
      print("Failed to get vote results for event ID: $votingEventID - $e");
    }
    return [];
  }
```

getVoteResultsFromBlockchain in smart_contract_service.dart:

- Calls the blockchain's getVoteResults function
- Returns the list of candidate IDs and their vote counts
- Acts as a bridge between the blockchain and the application

```
// Calculate the votes and determine the winner (if event has ended)
    if (isEnded && candidateList.isNotEmpty) {
      winner = _votingEvent.candidates.reduce((a, b) => a.votesReceived > b.votesReceived ? a : b);
    }
```

The VotingEventPage widget in voting_event_page.dart displays the results:

- Calculate the vote counts to find out the winner candidate
- Shows the winner with their vote count
- Displays a bar chart of all candidates' votes
- Calculates and shows the winning percentage

2.1.5 Main Architecture

Provider Pattern

```
runApp(
    MultiProvider(
      providers: [
        ChangeNotifierProvider(
          create: (_) => ThemeProvider(),
        ),
        ChangeNotifierProvider(
          create: (_) => WalletProvider(),
        ),
        ChangeNotifierProvider(
          create: (_) => UserProvider(),
        ),
        ChangeNotifierProvider(
          create: (_) => VotingEventProvider()
        ),
        ChangeNotifierProvider(
          create: (_) => StudentProvider()
        ),
        Provider<WalletConnectService>(
          create: (_) {
            // 每次都重置并创建新实例
            WalletConnectService.reset();
            return WalletConnectService();
          },
        ),
        Provider<SmartContractService>(
          create: (_) {
            // 每次都重置并创建新实例
            SmartContractService.reset();
            return SmartContractService();
          },
        ),
        ChangeNotifierProvider(
          create: (_) => NotificationProvider(),
        ),
        ChangeNotifierProvider(
          create: (_) => UserManagementProvider(),
        ),
        ChangeNotifierProvider(
          create: (_) => CandidateProvider(),
        ),
      ],
      child: MainApp(user: user),
    ),
  );
```

- The code uses MultiProvider to set up multiple providers simultaneously, allowing different parts of the application to access shared state and services. This is done through a list of providers, each wrapped in either ChangeNotifierProvider or Provider, creating a centralized state management system.
- For critical services like WalletConnectService and SmartContractService, the code implements a reset-and-create pattern. Before creating new instances, it calls a reset() method to ensure a clean state, preventing any potential state conflicts or memory leaks from previous instances.
- ChangeNotifierProvider for reactive state that needs to notify listeners of changes (used for UI-related state like themes, user data, and voting events)
- Provider for services that don't need to notify listeners of changes (used for core services like wallet connection and smart contract interaction)

Theme Shared-Preferences

```dart
class ThemeSharedPreferences {
  static const String _themeKey = 'theme_mode';

  // Save theme mode to shared preferences
  static Future<void> saveThemeMode(bool isDarkMode) async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    await prefs.setBool(_themeKey, isDarkMode);
  }

  // Load theme mode from shared preferences
  static Future<bool> loadThemeMode() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    return prefs.getBool(_themeKey) ?? false; // Default to light mode
  }
}
```

- ThemeSharedPreferences class in theme_shared_preferences.dart - Handles theme preferences
- saveThemeMode() and loadThemeMode() methods for persistence

Scrollable Responsive Widget

```dart
class ScrollableWidget extends StatelessWidget {
  final Widget child;
  final bool hasBottomNavigationBar;

  const ScrollableWidget({
    super.key,
    required this.child,
    this.hasBottomNavigationBar = false,
  });

  @override
  Widget build(BuildContext context) {
    return SingleChildScrollView(
      child: ConstrainedBox(
        constraints: BoxConstraints(
          minHeight: hasBottomNavigationBar ?
            (MediaQuery.of(context).size.height - kBottomNavigationBarHeight - 30)
            : MediaQuery.of(context).size.height - 90, // ensures minimum height matches screen size
        ),
        child: Padding(
          padding: const EdgeInsets.all(20),
          child: child,
        ),
        // child: IntrinsicHeight( // makes column height dynamic based on content
        //   child: child,
        // ),
      ),
    );
  }
}
```

```dart
class ResponseWidget extends StatelessWidget {
  final Widget phone;
  final Widget tablet;

  const ResponseWidget({
    super.key,
    required this.phone,
    required this.tablet,
  });

  @override
  Widget build(BuildContext context) {
    bool isTablet = MediaQuery.of(context).size.shortestSide > 800;

    return isTablet ? tablet : phone;
  }
}

class ScrollableResponsiveWidget extends StatelessWidget {
  final Widget phone;
  final Widget tablet;
  final bool hasBottomNavigationBar;

  const ScrollableResponsiveWidget ({
    super.key,
    required this.phone,
    required this.tablet,
    this.hasBottomNavigationBar = false,
  });

  @override
  Widget build(BuildContext context) {
    return ScrollableWidget(
      hasBottomNavigationBar: hasBottomNavigationBar,
      child: ResponseWidget(
        phone: phone,
        tablet: tablet,
      ),
    );
  }
}
```

- The ResponseWidget implements a simple but effective responsive design system that switches between phone and tablet layouts based on screen width. It uses MediaQuery.of(context).size.shortestSide > 800 as a breakpoint, automatically rendering the appropriate layout (phone or tablet) without requiring manual screen size checks throughout the app.
- The ScrollableWidget provides a consistent scrolling container with dynamic height constraints. It automatically adjusts its minimum height based on whether there's a bottom navigation bar (subtracting kBottomNavigationBarHeight + 30 pixels) or not (subtracting 90 pixels), ensuring content fills the screen appropriately while maintaining scrollability.
- The ScrollableResponsiveWidget combines both responsive and scrollable functionality by wrapping the ResponseWidget with ScrollableWidget. This creates a unified component that handles both screen size adaptation and proper scrolling behavior, with an optional hasBottomNavigationBar parameter to adjust the layout accordingly.

Flutter Localization

```dart
final FlutterLocalization _localization = FlutterLocalization.instance;

  @override
  void initState() {
    super.initState();
    _localization.init(
      mapLocales: [
        MapLocale('en',
          AppLocale.en,
          countryCode: 'US',
          fontFamily: 'Font EN',
        ),
        MapLocale('ms',
          AppLocale.ms,
          countryCode: 'MS',
          fontFamily: 'Font MS',
        ),
        MapLocale('zh',
          AppLocale.zh,
          countryCode: 'ZH',
          fontFamily: 'Font ZH',
        ),
      ],
      initLanguageCode: 'zh',
    );
    _localization.onTranslatedLanguage = _onTranslatedLanguage;
  }

  void _onTranslatedLanguage(Locale? locale) {
    setState(() {});
  }
```

- The code initializes a multi-language system supporting English, Malay, and Chinese, with each language having its own country code and font family.
- The system is configured to start with Chinese as the default language through the initLanguageCode: 'zh' parameter.
- The _onTranslatedLanguage callback function is implemented to refresh the UI whenever the language is changed, ensuring all text elements update to the new language.

.env Settings

```
GMAIL_MAIL = "tiewjj-wp21@student.tarc.edu.my"
GMAIL_PASSWORD = "vads klzp onbg tdhl"

PRIVATE_KEY = "76f92ce2fd78acfebd7eaffcb518662033d52e1bdcaf235294daa679042d9d73"
CONTRACT_ADDRESS = "0xF4bC6fb85780748c16972BEef2c5ce669D1822fC"
```

Blockchain Configuration:

- CONTRACT_ADDRESS: Stores the Ethereum smart contract address for the voting system
- Multiple RPC provider URLs for blockchain connectivity

Email Configuration:

- GMAIL_MAIL: Email address used for sending notifications
- GMAIL_PASSWORD: Password for the email account

Security and Access:

- The .env file is intentionally excluded from version control (listed in .gitignore)
- Environment variables are loaded using the flutter_dotenv package
- These variables are accessed throughout the application using dotenv.env['VARIABLE_NAME']

2.2 Testing Strategies

intro.

. Smart Contract Testing

- Include any tests for your smart contract functions:
  - Tests for creating voting events
  - Tests for casting votes
  - Tests for retrieving results

2. Application Testing

- Include tests for your Flutter application:
  - Authentication tests
  - Voting process tests
  - Results display tests

3. Integration Testing

- Show how blockchain and application integration was tested:
  - Tests for wallet connection
  - Tests for transaction handling
  - Tests for data consistency between blockchain and application

4. Security Testing

- Include security testing aspects:
  - Authentication and authorization tests
  - Vote validation tests
  - Blockchain transaction security testing

5. User Acceptance Testing

- Include any user testing you conducted:
  - User scenarios and results

Feedback and improvements made

2.2.1 Sub-subsection Heading

Sub-section numbering should be limited to a maximum of 3 levels (e.g. 5.3.1) in order to avoid confusion.

2.3 Test Plan

2.4 Test Data

2.5 Test Cases

Sub-sections should be used to divide the chapter into logical parts.

2.5.1 Sub-subsection Heading

Sub-section numbering should be limited to a maximum of 3 levels (e.g. 5.3.1) in order to avoid confusion.

2.6 Chapter Summary and Evaluation

At the end of each chapter, evaluate the contents stated or discussed in the relevant sub-sectio

Chapter 7

Discussions and Conclusion

3 Discussions and Conclusion

Each student is required to make an *evaluation of the project* he/she has embarked on. The project evaluation may include the following sections.

IMPORTANT NOTE TO STUDENTS: In this chapter, for problems related to code, hardware, internet connection, etc (where applicable):

- List the technical problems faced and state how they were resolved
- List the unsolved technical problems for future enhancement
- List the achieved objectives/modules
- List the incomplete parts for future enhancement - this is regardless the parts listed in the pre-determined scope. Suggestions for future improvement

Summary

Summarize the project including the problem and proposed solutions, justification of the choice of tools, techniques and methodologies used in this project.

Achievements

Students are required to evaluate the project's achievement against project objectives, completion of the project, students' view of the strengths and weaknesses of the work done.

Contributions

Discuss the creativity, innovativeness, contribution of the proposed system. Explain why the proposed system is necessary. Describe the marketability of the system.

Limitations and Future Improvements

Identify the limitations of the research or project. Provide suggestions for improvement or further development of the system or research in the future.

Issues and Solutions

Students are required to describe the various problems faced by students during the project development and explain what has been done to solve the problems. The valuable experiences gained or lessons learnt through the project as a whole. The issues may include technical issues, project management issues, team dynamics problems, and other difficulties

encountered and lessons learnt. How the issues are solved or can be solved to ensure the project can be completed on time or to be improved in the future.

References

The list of references should contain all the items that have been explicitly referred to in the report. Students may have referred to some manuals or conducted research on certain knowledge areas, for example Oracle, RFID scanners etc.

If the students choose to write a bibliography, then they should list down all materials that are useful either directly or indirectly to the project concerned and which the students have read. An example of material useful could be an article or an Oracle user manual etc.

References should be cited in the main text using Author-Year system. The full references should be given as below (essentially APA Referencing format), in the alphabetical order in 10 pt. Times New Roman, with a 6pt spacing between each. References should be arranged first alphabetically and then further sorted chronologically if necessary. More than one reference from the same author(s) in the same year must be identified by the letters "a", "b", "c", etc., placed after the year of publication.

Please ensure that every reference cited in the text is also present in the reference list (and vice versa). Any references cited in the abstract must be given in full.

Web references used as a minimum, the full URL should be given and the date when the reference was last accessed. Any further information, if known (DOI, author names, dates, reference to a source publication, etc.), should also be given. Web references can be listed separately (e.g., after the reference list) under a different heading if desired, or can be included in the reference list.

Appendices

In order to enhance better understanding of the project, students should as far as possible include all directly relevant materials, figures or diagrams in the **main body** rather than in the Appendix. The appendix is reserved only for items which may not directly be relevant or essential to enhance a reader's understanding of the project, or which may interrupt the smooth reading of the project document (for example being too voluminous).

Appendices should only include supportive materials **directly referred** to in the writing and should be kept to a **minimum**, e.g. selected pages of an annual report, not the entire document. Examples of items included in Appendices are:

- Company's report and documentation, such as sample invoice, purchase order form, etc.
- Project meeting documentation e.g. minutes of meetings, tracking documents, memos etc.
- Questionnaires and results, interview questions and results, pilot test and results, observation sheet and results, experiment test plan and results, etc.
- Analysis/design diagrams (only those not incorporated in the main body of the report).

If there is more than one appendix, they should be identified as A, B, etc (e.g. Appendix A). Formulae and equations in appendices should be given separate numbering: Eq. (A.1), Eq. (A.2), etc.; in a subsequent appendix, Eq. (B.1) and so on. Similarly for tables and figures: Table A.1; Fig. A.1, etc.

IMPORTANT NOTE TO STUDENTS

APPENDIX *n* User Guide

- List the username and password of multiple roles (if applicable)
- Provide clear screen shots of each page, explain the functions of each button

As a rough guide, the user guide should include the following sections:

System Document

In this section, students should provide the following pieces of information:

- **System (hardware and software requirements)**. Students should describe the minimum hardware and software requirements to install the software application which has been developed by the students, for example, DBMS, OS, program development tools etc.
- **Installation**. Under this section, students should create an 'installation' CD and provide a brief step-by-step guide on how a new user can install the software on a computer system. Students should indicate any special setup information, such as the specific location of placing the database files, the SQL statements to add tables, etc. Software source code should also be included in this CD. Students are not required to print out the software code.

Operation Document

Under this section, students are required to provide a brief step-by-step guide on how to use the installed software. The guides should teach the user how to run the software and use its major functions and features. For example, steps show guide the users on how to run the system, e.g. to use an executable file or to use the IDE. The login information such as username and password for each of the different users (or roles) must be provided. Some screen interfaces would be useful.

APPENDIX *n+1* Developer Guide*

*To be included for ***Smart Campus Projects*** and ***Real-Life Projects***. This section should include the following:

- List the necessary software, installer, API, library that must be installed
- List the authentication details, such as username and password for all security

Other Appendices:

- Supporting documents
- Non-disclosure agreement (if applicable)
- Other detailed documentation, such as important references, interview results, survey results, etc.

This page is intentionally left blank to indicate the back cover. Ensure that the back cover is black in color.