

**Project Name**

***UR-students saving system***

**system**

**UNIVERSITY OF RWANDA COLLEGE OF BUSINESS AND ECONOMICS**

**SCHOOL OF BUSINESS**

**BIT DEPARTMENT**

**LEVEL 2 GROUP 1**

**COURSE*: SYSTEMS ENGINEERING, DATABASE AND JAVA PROGRAMMING***

**PROJECT PROPOSAL**

**ON**

**UR-STUDENTS-SAVING-SYSTEM**

**Names: NIYONZIMA gilbert**

**Registration number: 222010015**

**Submitted to: Dr BUGINGO Emmanuel (Lecturer)**

**Submission date: 09-mar-2024**

**PART 1.** SYSTEM ENGINEERING

# Table of contents

Contents

[Table of contents 2](#_Toc137559274)

[List of Figures 4](#_Toc137559275)

[List of tables 5](#_Toc137559276)

[1. Introduction 6](#_Toc137559277)

[2. Statement of the problem 7](#_Toc137559278)

[2.1 Description of the existing system 7](#_Toc137559279)

[2.2 Problems of the current system 7](#_Toc137559280)

[2.3 How the proposed system will work 7](#_Toc137559281)

[2.4 Describe other alternatives 7](#_Toc137559282)

[3. Software requirement 7](#_Toc137559283)

[3.1 User Requirements 7](#_Toc137559284)

[3.1.1 Functional requirement 7](#_Toc137559285)

[3.1.2 Non-function requirement 7](#_Toc137559286)

[Portability requirement 7](#_Toc137559287)

[Reliability requirement 7](#_Toc137559288)

[Usability requirement 7](#_Toc137559289)

[Space requirement 7](#_Toc137559290)

[Organization requirement 7](#_Toc137559291)

[Implementation requirement 7](#_Toc137559292)

[External environment requirement 7](#_Toc137559293)

[Privacy requirement 7](#_Toc137559294)

[Safety requirement 7](#_Toc137559295)

[3.2 System requirements 7](#_Toc137559296)

[3.2.1 Minimum End-user Hardware Requirements 7](#_Toc137559297)

[3.2.2 Minimum End-user Software Requirements 7](#_Toc137559298)

[3.3 Software specifications 7](#_Toc137559299)

[**4.** DATA FLOW DIAGRAM (LEVEL 0, LEVEL 1**)** 8](#_Toc137559300)

[4.1 DFD LEVEL 0 8](#_Toc137559301)

[4.2 DFD LEVEL 1 9](#_Toc137559302)

[5. Feasibility study 9](#_Toc137559303)

[5.1 Technical Feasibility 9](#_Toc137559304)

[5.2 Financial Feasibility 9](#_Toc137559305)

[5.3 Market Feasibility 10](#_Toc137559306)

[5.4 Economic Feasibility 10](#_Toc137559307)

[5.5 Legal and Regulatory Feasibility 10](#_Toc137559308)

[5.6 Operational Feasibility 10](#_Toc137559309)

[6. Description of the selected Process Model 10](#_Toc137559310)

[6.1 Cause of your selection 10](#_Toc137559311)

[7. Conclusion 10](#_Toc137559312)

[8. References 10](#_Toc137559313)

# List of Figures

# List of tables

1. Introduction

**Saving:** is keep money aside for future use

The UR-students saving system is a program that helps students at the University of Rwanda save money for their education. The system offers a variety of features to help students reach their savings goals, including matching contributions, low fees, and easy access to funds. The system also provides financial education resources to help students learn how to manage their money wisely.

Here are some of the benefits of the UR-students saving system:

**Helps students save money for their education:** This can reduce the amount of debt that students have to take on when they graduate.

**Teaches students about financial responsibility:** This can help students develop good financial habits that they can use throughout their lives.

**Increases access to education:** By making it easier for students to save money, the system can help more students attend college.

The UR-students saving system is easy to use. Students can create an account on the UR-students saving system website or app. Once they have created an account, they can set savings goals and track their progress. They can also make deposits into their savings account through a variety of methods, such as bank transfers, credit card payments, or cash deposits. Students will be able to access their savings account funds at any time.

If you are a student at the University of Rwanda, I encourage you to create an account and start saving today! The UR-students saving system is a valuable tool that can help you reach your financial goals.

A student savings system project is a program that helps students save money for their education issues

Sometimes in UR colleges students face things that require money, often we don't have any, so this system will help us to solve these problems using the money we have saved

1. Statement of the problem

2.1 Description of the existing system

System that exists, UR student does not use online system to save their money. They form groups and then they vote a leader of those groups that collect and control their money

2.2 Problems of the current system

Saving money using an offline system, such as keeping cash or using traditional methods can have several problems:

System doesn’t use online saving

Difficult to believe a leader that have responsibility of collect and control money

**Vulnerability to theft:** Storing money offline, especially in physical form, makes it susceptible to theft. If someone gains access to your cash or savings jars, there is a high risk of losing all your savings without any means of recovery.

**Lack of interest or growth:** Unlike online or digital savings accounts, offline systems do not offer any interest or investment opportunities. This means your savings remain stagnant and do not grow over time. Inflation can erode the value of your money, leading to a loss of purchasing power.

**Limited tracking and management:** Offline systems make it difficult to track and manage your savings effectively. Without proper records or online tools, it can be challenging to keep track of your progress, set goals, or monitor your spending habits.

**Risk of damage or loss:** Physical forms of savings, such as cash or coins, can be easily damaged or lost due to various factors such as fire, accidents, or misplacement. If such an event occurs, there is no way to recover the lost money.

**Lack of convenience and accessibility:** Offline savings systems may lack the convenience and accessibility provided by digital methods. Accessing your savings or making transactions requires physical interaction and may be limited to specific locations or circumstances.

To address these problems, it is advisable to consider using online or digital savings options. Digital savings accounts or investment platforms offer greater security, growth potential, and convenience. They provide features like interest earnings, online tracking, automated transfers, and easy access to funds when needed. Additionally, maintaining an emergency fund in a secure offline location can provide a backup in case of unexpected situations.

2.3 How the proposed system will work

UR-students saving system will work as follows:

The UR-Students Saving System project is designed to help students save money and manage their finances effectively. Here is a general description of how the system might work:

User Registration: Students would begin by registering on the UR-Students Saving System platform. They would provide their basic information, such as their name, email address, and student ID.

**Account Creation:** Once registered, students would create their personal savings account within the system. This account would serve as a secure digital wallet where they can deposit and manage their savings.

**Deposit and Withdrawal:** Students would have the option to deposit money into their savings account using various methods, such as bank transfers, online payment platforms, or through integration with their student bank accounts. Withdrawals could be made when needed, allowing students to access their savings.

**Saving Goals:** The system would allow students to set specific savings goals. They could define the purpose of their savings, such as tuition fees, textbooks, travel, or any other personal financial objectives.

**Tracking Progress:** The platform would provide students with tools to track their savings progress towards their goals. They could see how much they have saved, their remaining target, and the time frame they have set to achieve their goals.

**Budgeting and Expense Tracking**: The UR-Students Saving System could also offer budgeting and expense tracking features. Students could categorize their expenses, set spending limits, and receive notifications when they approach or exceed their budget.

**Financial Education Resources:** The platform might provide educational resources on personal finance topics. These resources could include articles, videos, interactive tools, and quizzes to help students learn about budgeting, saving strategies, investing, and other relevant financial concepts.

**Incentives and Rewards**: To motivate students to save, the system could offer incentives and rewards. For example, students might earn points or receive discounts at partner stores for reaching specific savings milestones or demonstrating consistent saving habits.

**Alerts and Reminders**: The system could send regular alerts and reminders to students to encourage them to save, check their progress, or provide updates on their financial goals. These notifications could be delivered via email, SMS, or within the platform itself.

**Security and Privacy:** The UR-Students Saving System would prioritize the security and privacy of students' personal and financial information. Robust encryption protocols, secure servers, and strict data privacy measures would be implemented to ensure the confidentiality and integrity of users' data.

It's important to note that the actual functionality and features of the UR-Students Saving System project may vary based on the specific design and implementation choices made by the development team.

Here are some of the benefits of the UR-students saving system:

**Helps students save money for their education:** This can reduce the amount of debt that students have to take on when they graduate.

**Teaches students about financial responsibility**: This can help students develop good financial habits that they can use throughout their lives.

**Increases access to education:** By making it easier for students to save money, the system can help more students attend college.

The UR-students saving system is a valuable tool for helping students save money for their education. The system can offer a number of benefits, and it is easy to use. If you are a student at the University of Rwanda, I encourage you to create an account and start saving today!

* 1. Describe other alternatives

The UR-Students Saving System project aims to provide a platform for students to save money and manage their finances effectively. While the specifics of the project are not provided, here are some alternative approaches or features that could be considered as part of an alternative system:

**Virtual Piggy Bank:** Create a virtual piggy bank where students can deposit money, track their savings, and set savings goals. The system could have a gamified interface with rewards and incentives for achieving savings targets.

**Financial Education Portal:** Develop an online portal that offers financial education resources tailored to students. This could include articles, videos, interactive tools, and quizzes to help students learn about budgeting, investing, and managing their money wisely.

**Expense Tracking and Budgeting Tool:** Build a platform that allows students to track their expenses, categorize their spending, and create budgets. The system could generate reports and provide visualizations to help students understand their spending habits and make informed financial decisions.

**Peer-to-Peer Saving Groups:** Facilitate the formation of saving groups among students, where members contribute a fixed amount regularly, and the collective savings can be used for specific purposes or emergencies. The system could provide tools to manage group contributions, track progress, and distribute funds when needed.

**Integration with Student Bank Accounts**: Collaborate with banks to integrate the saving system directly with student bank accounts. This would allow for seamless transfer of funds, real-time balance updates, and personalized financial recommendations based on students' spending patterns.

**Financial Goal Setting:** Enable students to set specific financial goals, such as saving for tuition fees, textbooks, or a study abroad program. The system could provide progress tracking, reminders, and suggestions on how to achieve those goals efficiently.

Incentivized Saving: Introduce incentives to encourage students to save money. For example, the system could offer cashback rewards, discounts, or access to exclusive deals for students who meet certain savings milestones.

**Collaborative Saving:** Enable students to invite friends or family members to contribute to their savings goals. This feature would allow for collaborative efforts to achieve larger financial objectives and foster a sense of community support.

**Automatic Round-up Savings:** Implement a feature that automatically rounds up students' transactions to the nearest dollar or a predefined amount, with the difference being deposited into their savings account. This approach encourages incremental saving without requiring conscious effort.

**Investment Options**: Provide students with investment opportunities to grow their savings. This could involve offering a range of investment products with varying risk levels, such as low-cost index funds or socially responsible investment options.

1. Software requirement

3.1 User Requirements

3.1.1 Functional requirement

The functional requirements for the UR-students saving system are as follows:

**Create an account:** Students must be able to create an account on the UR-students saving system website or app.

Set savings goals: Students must be able to set savings goals and track their progress.

**Make deposits:** Students must be able to make deposits into their savings account through a variety of methods, such as bank transfers, credit card payments, or cash deposits.

**Withdraw funds:** Students must be able to withdraw funds from their savings account at any time.

**Matching contributions:** The University of Rwanda may match a portion of the money that students save.

**Low fees:** The UR-students saving system must have low fees, which means that more of the money that students save goes towards their education.

**Easy access:** Students must be able to access their savings account funds at any time.

**Financial education resources:** The UR-students saving system must provide financial education resources to help students learn how to manage their money wisely.

These are just some of the functional requirements for the UR-students saving system. The specific requirements may vary depending on the specific needs of the University of Rwanda and its students.

Here are some additional functional requirements that could be considered for the UR-students saving system:

**The ability to transfer funds between savings accounts:** This would allow students to move money between their UR-students saving account and other accounts, such as a checking account or a credit card.

**The ability to set up automatic transfers:** This would allow students to set up recurring transfers from their checking account to their UR-students saving account. This would help students save money on a regular basis.

**The ability to view account statements:** Students should be able to view their account statements online or through the UR-students saving system app. This would allow them to track their spending and savings habits.

**The ability to contact customer support:** If students have any questions or problems with the UR-students saving system, they should be able to contact customer support.

3.1.2 non-function requirement

These are just some of the non-functional requirements for the UR-students saving system. The specific requirements may vary depending on the specific needs of the University of Rwanda and its students.

**Portability:** The system must be portable. This means that it must be able to be easily moved from one environment to another. For example, the system must be able to be deployed on-premises or in the cloud. This is important because it allows the university to easily scale the system as needed. It also allows the university to move the system to a different environment if necessary.

**Reliability:** The system must be reliable. This means that it must be able to consistently perform its functions without errors. The system must also be able to recover from failures quickly and without data loss. This is important because it ensures that students can always access their accounts and make transactions. It also ensures that the university's financial data is always secure.

**Usability:** The system must be easy to use. This means that the system must be intuitive and easy to navigate. The system must also provide clear and concise error messages. This is important because it ensures that students can easily use the system to manage their accounts. It also helps to reduce the number of errors that students make.

**Space requirement:** The system must be able to store a large amount of data, including student records, account information, and transaction history. This means that the system must have a large storage capacity. The system must also be able to efficiently access and process this data.

**Organizational requirement:** The system must be aligned with the university's organizational structure and policies. This means that the system must be able to integrate with the university's existing systems and processes. The system must also comply with the university's privacy and security policies.

**Implementation requirement**: The system must be implemented using secure and reliable technologies. This means that the system must be developed using secure coding practices and must be hosted on secure servers. The system must also be regularly updated with security patches.

**External environment requirement:** The system must be able to operate in a variety of external environments, including on-premises and cloud-based environments. This means that the system must be able to be deployed in a variety of ways and must be able to adapt to changes in the external environment.

**Safety requirement**: The system must be safe and secure. Student data must be protected from unauthorized access. This means that the system must have strong authentication and authorization mechanisms in place. The system must also be regularly audited for security vulnerabilities.

**Privacy requirement:** The system must comply with all applicable privacy laws and regulations. This means that the system must only collect and use student data in ways that are consistent with the law. The system must also provide students with the ability to access and control their personal data.

These are just some of the non-functional requirements that could be considered for the UR-students saving system. The specific requirements that are adopted will depend on the needs of the university and its students.

3.2 System requirements

3.2.1 Minimum End-user Hardware Requirements

The minimum end-user hardware requirements for the UR-students saving system are as follows:

Operating system: Windows 7 or later, macOS 10.11 or later, or Linux

Processor: Intel Core i3 or equivalent

Storage: 1 GB free disk space

Memory: 4 GB RAM

Internet connection: Broadband internet connection

These are the minimum requirements for the UR-students saving system to function properly. However, students may experience a better user experience if they have a computer with higher specifications.

Here are some additional hardware requirements that could be considered for the UR-students saving system:

Graphics card: A graphics card with at least 2 GB of VRAM is recommended for a better user experience.

Display: A display with a resolution of 1920x1080 or higher is recommended for a better user experience.

Web browser: The UR-students saving system is compatible with the latest versions of Chrome, Firefox, and Edge.

3.2.2 Minimum End-user Software Requirements

The minimum end-user software requirements for the UR-students saving system are as follows:

Web browser: The UR-students saving system is compatible with the latest versions of Chrome, Firefox, and Edge.

Operating system: Windows 7 or later, macOS 10.11 or later, or Linux

Browser plugins: The UR-students saving system requires the following browser plugins:

JavaScript: JavaScript is required for the UR-students saving system to function properly.

Cookies: Cookies are required for the UR-students saving system to remember your login information and other settings.

Ad blockers: Ad blockers may interfere with the functionality of the UR-students saving system. It is recommended to disable ad blockers when using the system.

These are the minimum requirements for the UR-students saving system to function properly. However, students may experience a better user experience if they have a web browser with higher specifications.

Here are some additional software requirements that could be considered for the UR-students saving system:

Security software: It is recommended to install security software on your computer to protect your data from unauthorized access.

Virus scanner: A virus scanner is recommended to scan your computer for viruses and other malware.

Firewall: A firewall is recommended to protect your computer from unauthorized access from the internet.

3.3 Software specifications

The UR-students saving system is a web-based application that will allow students to manage their savings accounts. The system will allow students to view their account balances, make deposits and withdrawals, and transfer money between accounts. The system will also provide students with reports on their spending habits.

Functional Requirements

The UR-students saving system must meet the following functional requirements:

**Account management:** The system must allow students to create and manage their savings accounts. This includes the ability to view account balances, make deposits and withdrawals, and transfer money between accounts.

**Reports:** The system must provide students with reports on their spending habits. This includes the ability to view monthly spending reports, category-based spending reports, and merchant-based spending reports.

**Security:** The system must be secure and protect students' personal and financial information. This includes the use of strong authentication and authorization mechanisms, as well as regular security audits.

Non-Functional Requirements

The UR-students saving system must meet the following non-functional requirements:

Availability: The system must be available 24/7. Students should be able to access their accounts and make transactions at any time, day or night.

Scalability: The system must be able to handle a high volume of transactions. The number of students enrolled at UR is expected to grow in the future, so the system must be able to scale to meet the demand.

Security: The system must be secure and protect students' personal and financial information. This includes the use of strong authentication and authorization mechanisms, as well as regular security audits.

Reliability: The system must be reliable. Students must be able to rely on the system to be available and to work properly.

Usability: The system must be easy to use. Students must be able to easily navigate the system and complete their transactions.

Implementation

The UR-students saving system will be implemented using the following technologies:

Web application: The system will be a web application that can be accessed by students from any device with an internet connection.

Database: The system will use a database to store student data, account information, and transaction history.

Web services: The system will use web services to interact with other systems, such as the university's student information system.

Testing

The UR-students saving system will be tested using the following methods:

Unit testing: Unit tests will be used to test the individual components of the system.

Integration testing: Integration tests will be used to test the interactions between the different components of the system.

System testing: System tests will be used to test the entire system as a whole.

Deployment

The UR-students saving system will be deployed to a cloud-based environment. This will allow the system to be easily scaled to meet the demand.

Maintenance

The UR-students saving system will be maintained by a team of qualified professionals. The team will be responsible for fixing bugs, adding new features, and keeping the system up-to-date with the latest security patches.

Additional Details

The system will be developed using the following programming languages: Java, JavaScript, and SQL.

The system will be hosted on the Amazon Web Services (AWS) cloud platform.

The system will use the following open source technologies: Spring Boot, Hibernate, and JUnit.

4. DATA FLOW DIAGRAM (LEVEL 0, LEVEL 1)

4.1 DFD LEVEL 0

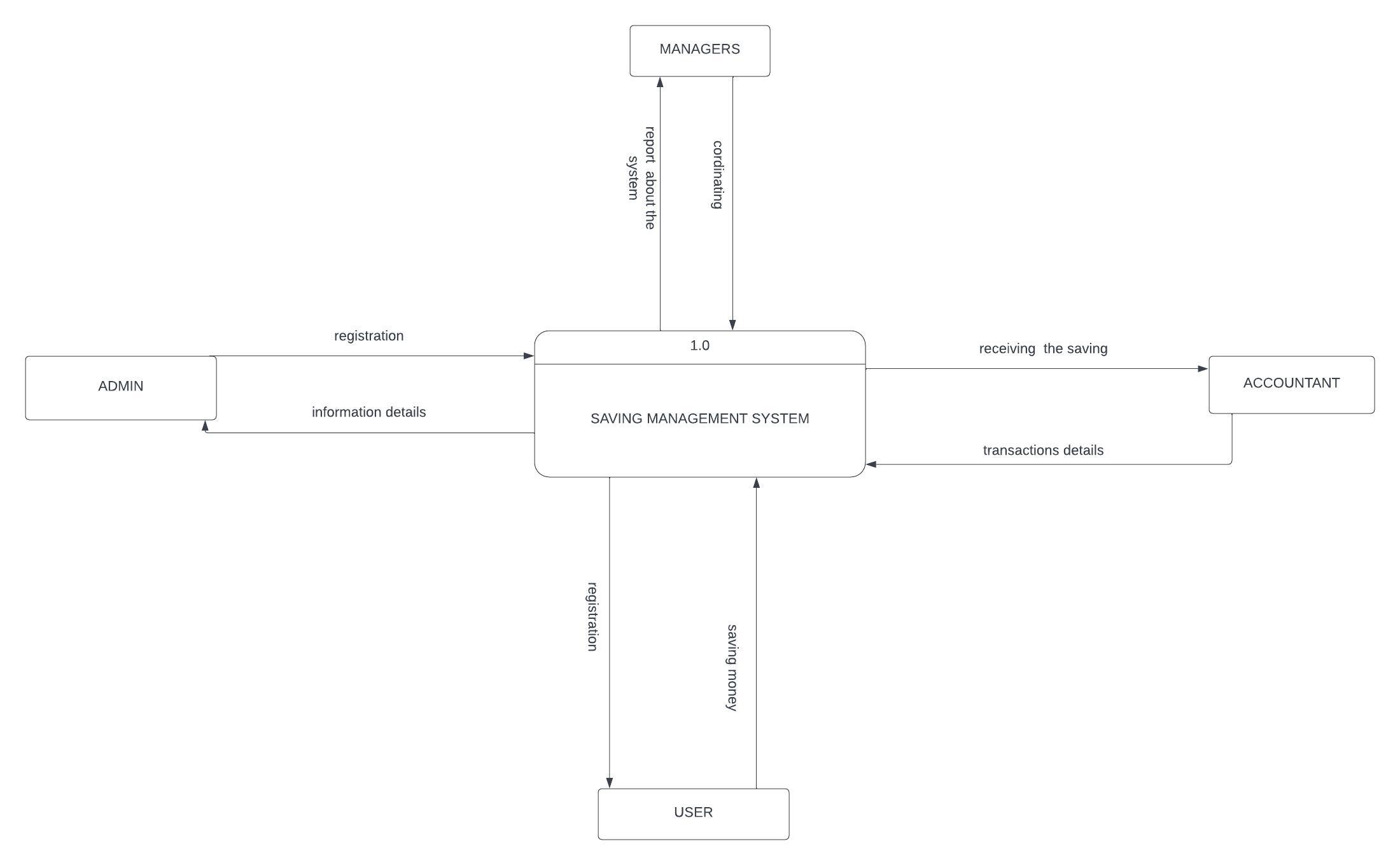


Figure1: data flow diagram level 0

This figure defines the boundary between the system, or part of a system, and its environment, showing the entities that interact with it, this diagram is a high-level view of a system.

4.2 DFD LEVEL 1

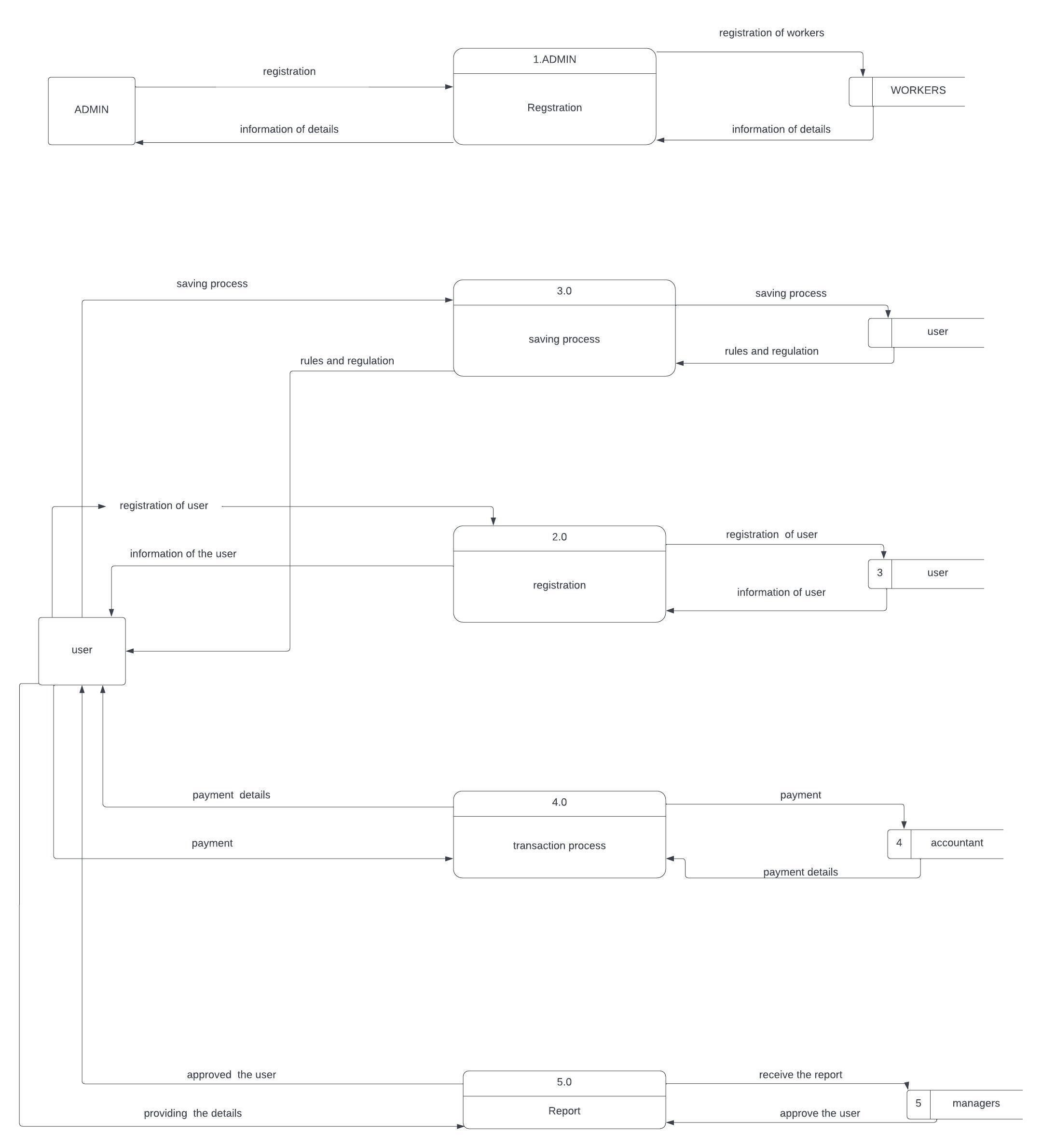


Figure2: data flow diagram level 1

1system is represented as a single process. A level 1 DFD notates each of the main sub-processes that together form the complete system.

5. Feasibility study

5.1 Technical Feasibility

The project must be technically feasible. This means that it must be possible to develop and deploy the system using the available technology. The project's technical feasibility will depend on the size and complexity of the system, as well as the skills and experience of the development team.

5.2 Financial Feasibility

 The project must be financially feasible. This means that it must generate enough revenue to cover its costs. The project's costs will include the cost of development, deployment, and maintenance. The project's revenue will come from fees charged to students for using the system.

5.3 Market Feasibility

 The project must be market feasible. This means that there must be a demand for the system. The project's market feasibility will depend on the number of students at UR, as well as their willingness to use the system.

5.4 Economic Feasibility

The project must be economically feasible. This means that it must generate enough revenue to cover its costs and provide a return on investment. The project's economic feasibility will depend on the number of students who use the system, as well as the fees that are charged.

5.5 Legal and Regulatory Feasibility

 The project must be legal and regulatory feasible. This means that it must comply with all applicable laws and regulations. The project's legal and regulatory feasibility will depend on the laws and regulations in the jurisdiction where the system will be deployed.

5.6 Operational Feasibility

 The project must be operationally feasible. This means that it must be possible to operate and maintain the system effectively. The project's operational feasibility will depend on the availability of resources, such as staff and equipment.

6. Description of the selected Process Model

I decide to select agile model

Agile is a process model that is well-suited for projects with changing requirements and a dynamic environment. The UR-students saving system project is a good candidate for an agile process model because the requirements for the system are likely to change as the project progresses. The agile process model allows the project team to adapt to changes in the requirements and to deliver the system on time and within budget.

The agile process model is based on the following principles:

**Iterative and incremental development:** The project is broken down into small, manageable tasks. These tasks are then completed in short sprints, and the project is continuously updated as new requirements are identified.

**Collaboration and communication:**The project team works closely together to ensure that the project is successful. The team communicates regularly with the stakeholders to ensure that the project meets their needs.

**Adaptability:** The project team is able to adapt to changes in the requirements and the environment. This allows the project to stay on track and to deliver the system on time and within budget.

The agile process model is a good choice for the UR-students saving system project because it allows the project team to adapt to changes in the requirements and to deliver the system on time and within budget. The agile process model is also a good choice because it promotes collaboration and communication between the project team and the stakeholders.

6.1 Cause of your selection

There are several reasons why agile could be a good process model for the UR-students saving system project.

The requirements for the system are likely to change as the project progresses. The agile process model is well-suited for projects with changing requirements because it allows the project team to adapt to changes as they happen. This is important for the UR-students saving system project because the requirements for the system are likely to change as the project progresses. For example, the university may decide to add new features to the system or change the way that the system is used. The agile process model allows the project team to adapt to these changes without having to start the project over from scratch.

The project team is likely to be made up of people with different skills and experience. The agile process model is well-suited for projects with teams that have different skills and experience because it allows the team to work together in a collaborative way. This is important for the UR-students saving system project because the project team is likely to be made up of people with different skills and experience. For example, the team may include people with experience in software development, user experience design, and project management. The agile process model allows these people to work together to deliver the system on time and within budget.

The project is likely to be delivered in phases. The agile process model is well-suited for projects that are delivered in phases because it allows the project team to deliver the system incrementally. This is important for the UR-students saving system project because the project is likely to be delivered in phases. For example, the project team may first deliver a basic version of the system and then add new features in subsequent phases. The agile process model allows the project team to deliver the system in a way that meets the needs of the stakeholders.

so, there are several reasons why agile could be a good process model for the UR-students saving system project. The agile process model is well-suited for projects with changing requirements, teams with different skills and experience, and projects that are delivered in phases.

7. Conclusion

The UR-students saving system project is a complex undertaking that has the potential to be a valuable resource for students at UR. The project team will need to carefully plan and execute the project in order to ensure its success.

# References

<https://store.ramseysolutions.com/money/books/the-total-money-makeover-by-dave-ramsey/>

<https://www.goodreads.com/en/book/show/621>

<https://yourmoneyoryourlife.com/book-summary/>

<https://www.goodreads.com/en/book/show/998>

these some books:

"A Random Walk Down Wall Street" by Burton G. Malkiel

"The Behavior Gap" by Carl Richards

"The Richest Man in Babylon" by George S. Clason

"Smart Women Finish Rich" by David Bach

-my colleagues

-software engineering books

**PART 2. DATABASE DESIGN**

**DATA BASE OF THE SYSTEM**

**2.1INTRODUCTION**:

The database for the University of Rwanda Student Savings System plays a crucial role in storing, managing, and safeguarding financial data related to student accounts and transactions. With its robust architecture, data integrity measures, and scalability features, the database ensures the reliability, security, and efficiency of the savings system, thereby supporting the financial needs of students effectively.

A database for your student saving system involves creating a structured storage system to efficiently store and manage student information. It includes designing tables to represent different aspects of student data such as personal details, enrollment information, grades, etc. Additionally, implementing database management software like MySQL, PostgreSQL, or MongoDB to handle data manipulation, retrieval, and security. This transition enables streamlined access to student records, facilitates data analysis, and ensures data integrity.

Top of Form

**SECTION I**

**1. Describe all the entities and their corresponding attributes that are in your database.**

**STUDENT**

**student-id, first-name, last-name, email, phone-number, date of birth, major, GPA, year in school**

**ACCOUNTANT**

**Accountant-id, first-name, last-name, email, phone-number, date of birth, years of experience, specialization**

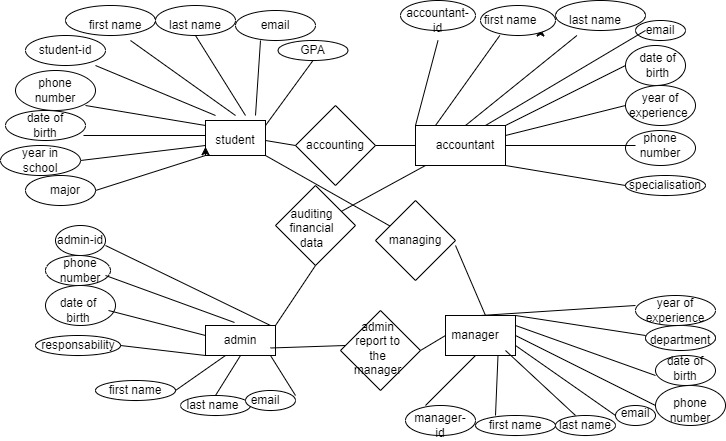
**MANAGER**

**Manager-id, first-name, last-name, email, email, phone-number, date of birth, department, years of experience**

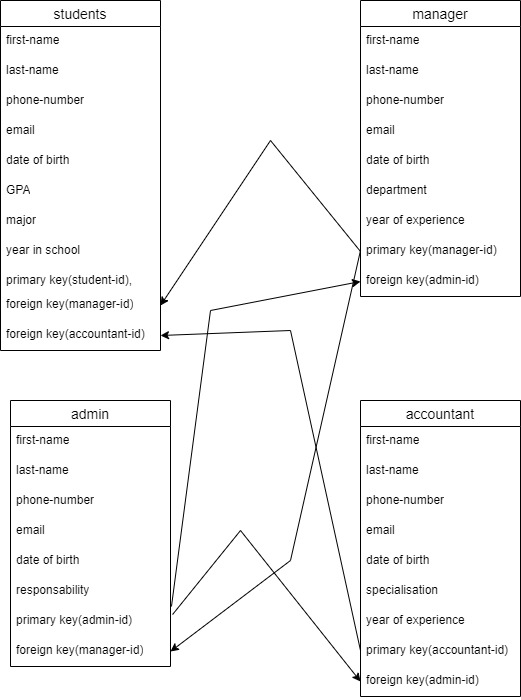
**ADMIN**

**Admin-id, first-name, last-name, email, phone-number, date of birth, responsibilities**

**2. create an ERD**

****

**3.create an LMD of your entities**

****

**SECTION II**

1. **Create the database of your system.**

**CREATE DATABASE UR STUDENT SAVING SYSTEM;**

**2. Write queries to create all the tables and relationships of your system.**

**CREATE TABLE students (**

**student\_id INT NOT NULL AUTO\_INCREMENT,**

**first\_name VARCHAR(255) NOT NULL,**

**last\_name VARCHAR(255) NOT NULL,**

**email VARCHAR(255) NOT NULL,**

**phone\_number VARCHAR(255) NOT NULL,**

**date\_of\_birth DATE NOT NULL,**

**major VARCHAR(255) NOT NULL,**

**GPA FLOAT NOT NULL,**

**year\_in\_school INT NOT NULL,**

**PRIMARY KEY (student\_id),**

**FOREIGN KEY (manager\_id),**

**FOREIGN KEY (accountant\_id)**

**);**

**CREATE TABLE accountants (**

**accountant\_id INT NOT NULL AUTO\_INCREMENT,**

**first\_name VARCHAR(255) NOT NULL,**

**last\_name VARCHAR(255) NOT NULL,**

**email VARCHAR(255) NOT NULL,**

**phone\_number VARCHAR(255) NOT NULL,**

**date\_of\_birth DATE NOT NULL,**

**years\_of\_experience INT NOT NULL,**

**specialization VARCHAR(255) NOT NULL,**

**PRIMARY KEY (accountant\_id),**

**FOREIGN KEY (admin\_id)**

**);**

**CREATE TABLE managers (**

**manager\_id INT NOT NULL AUTO\_INCREMENT,**

**first\_name VARCHAR(255) NOT NULL,**

**last\_name VARCHAR(255) NOT NULL,**

**email VARCHAR(255) NOT NULL,**

**phone\_number VARCHAR(255) NOT NULL,**

**date\_of\_birth DATE NOT NULL,**

**department VARCHAR(255) NOT NULL,**

**years\_of\_experience INT NOT NULL,**

**PRIMARY KEY (manager\_id),**

**FOREIGN KEY (admin\_id),**

**);**

**CREATE TABLE admins (**

**admin\_id INT NOT NULL AUTO\_INCREMENT,**

**first\_name VARCHAR(255) NOT NULL,**

**last\_name VARCHAR(255) NOT NULL,**

**email VARCHAR(255) NOT NULL,**

**phone\_number VARCHAR(255) NOT NULL,**

**date\_of\_birth DATE NOT NULL,**

**responsibilities VARCHAR(255) NOT NULL,**

**PRIMARY KEY (admin\_id),**

**FOREIGN KEY (manager\_id),**

**);**

**3. write queries to insert data into your tables.**

**-- Insert data into the student table**

**INSERT INTO student (name, email, student\_id, major)**

**VALUES ('John Doe', 'john.doe@example.com', 123456789, 'Computer Science');**

**-- Insert data into the accountant table**

**INSERT INTO accountant (name, email, employee\_id, department)**

**VALUES ('Jane Doe', 'jane.doe@example.com', 987654321, 'Accounting');**

**-- Insert data into the manager table**

**INSERT INTO manager (name, email, employee\_id, department)**

**VALUES ('Peter Smith', 'peter.smith@example.com', 234567890, 'Sales');**

**-- Insert data into the admin table**

**INSERT INTO admin (name, email, employee\_id)**

**VALUES ('Susan Jones', 'susan.jones@example.com', 345678901);**

**4. Write queries to display all the information in your tables.**

**-- Display all the information in the student table**

**SELECT \* FROM student;**

**-- Display all the information in the accountant table**

**SELECT \* FROM accountant;**

**-- Display all the information in the manager table**

**SELECT \* FROM manager;**

**-- Display all the information in the admin table**

**SELECT \* FROM admin;**

**5. Write a query to update information in any of the two tables of your system.**

**-UPDATE student**

**SET name = 'John Doe', email = 'john.doe@example.com'**

**WHERE id = 123456789;**

**-UPDATE student**

**SET name = 'John Doe'**

**WHERE major = 'Computer Science';**

**SECTION III**

**1. Create a view to insert data into your tables.**

**CREATE VIEW insert\_view AS**

**SELECT**

**student\_id,**

**first\_name,**

**last\_name,**

**email,**

**phone\_number,**

**date\_of\_birth,**

**major,**

**GPA,**

**year\_in\_school**

**FROM**

**students;**

**INSERT INTO students**

**SELECT**

**student\_id,**

**first\_name,**

**last\_name,**

**email,**

**phone\_number,**

**date\_of\_birth,**

**major,**

**GPA,**

**year\_in\_school**

**FROM**

**insert\_view;**

**CREATE VIEW insert\_accountant\_view AS**

**SELECT**

**first\_name,**

**last\_name,**

**email,**

**phone\_number,**

**date\_of\_birth,**

**years\_of\_experience,**

**specialization**

**FROM**

**accountants;**

**INSERT INTO accountants**

**SELECT**

**first\_name,**

**last\_name,**

**email,**

**phone\_number,**

**date\_of\_birth,**

**years\_of\_experience,**

**specialization**

**FROM**

**insert\_accountant\_view;**

**CREATE VIEW insert\_manager\_view AS**

**SELECT**

**manager\_id,**

**first\_name,**

**last\_name,**

**email,**

**phone\_number,**

**date\_of\_birth,**

**department,**

**years\_of\_experience**

**FROM**

**managers;**

**INSERT INTO managers**

**SELECT**

**manager\_id,**

**first\_name,**

**last\_name,**

**email,**

**phone\_number,**

**date\_of\_birth,**

**department,**

**years\_of\_experience**

**FROM**

**insert\_manager\_view;**

**CREATE VIEW insert\_admin\_view AS**

**SELECT**

**admin\_id,**

**first\_name,**

**last\_name,**

**email,**

**phone\_number,**

**date\_of\_birth,**

**responsibilities**

**FROM**

**admins;**

**INSERT INTO admins**

**SELECT**

**admin\_id,**

**first\_name,**

**last\_name,**

**email,**

**phone\_number,**

**date\_of\_birth,**

**responsibilities**

**FROM**

**insert\_admin\_view;**

**2. Create a view to display all the information in your tables.**

**CREATE VIEW all\_users AS**

**SELECT \***

**FROM manager**

**UNION ALL**

**SELECT \***

**FROM admin**

**UNION ALL**

**SELECT \***

**FROM accountant**

**UNION ALL**

**SELECT \***

**FROM student;**

**3. Create a view to update information in any of the two tables of your system.**

**-CREATE VIEW manager\_student\_update AS**

**SELECT manager.id, manager.name, student.department**

**FROM manager**

**JOIN student ON manager.id = student.manager\_id;**

**UPDATE manager\_student\_update**

**SET department = 'Engineering'**

**WHERE manager.name = 'Alice';**

**-CREATE VIEW student\_name\_update AS**

**SELECT name**

**FROM student;**

**UPDATE student\_name\_update**

**SET name = 'Alice';**

**4. Create a view to delete data in any two of your tables according to any simple condition of your choice.**

**-CREATE VIEW manager\_student\_delete AS**

**SELECT manager.id, manager.name, student.id, student.name, student.department**

**FROM manager**

**JOIN student ON manager.id = student.manager\_id**

**WHERE student.department = 'Engineering';**

**-CREATE VIEW manager\_student\_delete AS**

**SELECT manager.id, manager.name, student.id, student.name, student.department**

**FROM manager**

**JOIN student ON manager.id = student.manager\_id**

**WHERE student.department = 'Engineering' AND student.year = 4;**

**5. In your database, create one view of your choice that considers sub-query.**

**CREATE VIEW top\_performing\_students AS**

**SELECT student.id, student.name, student.gpa, student.department**

**FROM student**

**WHERE student.gpa > (**

**SELECT AVG(student.gpa)**

**FROM student**

**);**

**Section IV**

**1. Create a stored procedure to insert data into your tables.**

**CREATE PROCEDURE insert\_manager (**

**@id INT,**

**@name VARCHAR(255)**

**)**

**AS**

**BEGIN**

**INSERT INTO manager (id, name)**

**VALUES (@id, @name);**

**END;**

**CREATE PROCEDURE insert\_admin (**

**@id INT,**

**@name VARCHAR(255)**

**)**

**AS**

**BEGIN**

**INSERT INTO admin (id, name)**

**VALUES (@id, @name);**

**END;**

**CREATE PROCEDURE insert\_student (**

**@id INT,**

**@name VARCHAR(255),**

**@department VARCHAR(255),**

**@year INT**

**)**

**AS**

**BEGIN**

**INSERT INTO student (id, name, department, year)**

**VALUES (@id, @name, @department, @year);**

**END;**

**CREATE PROCEDURE insert\_accountant (**

**@id INT,**

**@name VARCHAR(255),**

**@department VARCHAR(255)**

**)**

**AS**

**BEGIN**

**INSERT INTO accountant (id, name, department)**

**VALUES (@id, @name, @department);**

**END;**

**2. Create a stored procedure to display all the information in your tables.**

**CREATE PROCEDURE display\_all\_data**

**AS**

**BEGIN**

**SELECT \***

**FROM manager**

**UNION ALL**

**SELECT \***

**FROM admin**

**UNION ALL**

**SELECT \***

**FROM student**

**UNION ALL**

**SELECT \***

**FROM accountant;**

**END;**

**3. Create a stored procedure to update information in any of the two tables of your system.**

**CREATE PROCEDURE update\_manager\_student (**

**@manager\_id INT,**

**@manager\_name VARCHAR(255),**

**@student\_id INT,**

**@student\_name VARCHAR(255)**

**)**

**AS**

**BEGIN**

**UPDATE manager**

**SET name = @manager\_name**

**WHERE id = @manager\_id;**

**UPDATE student**

**SET name = @student\_name**

**WHERE manager\_id = @manager\_id;**

**END;**

**-** **CREATE PROCEDURE update\_student\_department (**

**@manager\_name VARCHAR(255),**

**@department VARCHAR(255)**

**)**

**AS**

**BEGIN**

**UPDATE student**

**SET department = @department**

**WHERE manager\_id = (SELECT id FROM manager WHERE name = @manager\_name);**

**END;**

**4. Create a stored procedure to delete data in any two of your tables according to any simple condition of your choice.**

**-** **CREATE PROCEDURE delete\_engineering\_students (**

**@gpa FLOAT**

**)**

**AS**

**BEGIN**

**DELETE FROM student**

**WHERE department = 'Engineering' AND gpa < @gpa;**

**END;**

**OR**

**CREATE PROCEDURE delete\_engineering\_students\_and\_managers (**

**@gpa FLOAT**

**)**

**AS**

**BEGIN**

**DELETE FROM student**

**WHERE department = 'Engineering' AND gpa < @gpa;**

**DELETE FROM manager**

**WHERE id IN (SELECT manager\_id FROM student WHERE department = 'Engineering' AND gpa < @gpa);**

**END;**

**5. In your database, stored the procedure view of your choice that considers subquery.**

**-** **CREATE PROCEDURE view\_top\_performing\_students**

**AS**

**BEGIN**

**SELECT \***

**FROM (**

**SELECT student.id, student.name, student.gpa, student.department**

**FROM student**

**WHERE student.gpa > (SELECT AVG(student.gpa) FROM student)**

**) AS top\_performing\_students;**

**END;**

**-** **CREATE PROCEDURE view\_engineering\_students\_above\_3\_5\_gpa**

**AS**

**BEGIN**

**SELECT \***

**FROM (**

**SELECT student.id, student.name, student.gpa, student.department**

**FROM student**

**WHERE department = 'Engineering' AND gpa > 3.5**

**) AS engineering\_students\_above\_3\_5\_gpa;**

**END;**

**Section V**

**1. Create after inserting triggers for any two tables of your choice.**

**CREATE TRIGGER accountant\_insert\_trigger AFTER INSERT ON accountant FOR EACH ROW**

**BEGIN**

**-- Update the total number of accountants in the company.**

**UPDATE company**

**SET num\_accountants = num\_accountants + 1**

**WHERE id = 1;**

**END;**

**CREATE TRIGGER student\_insert\_trigger AFTER INSERT ON student FOR EACH ROW**

**BEGIN**

**UPDATE school**

**SET num\_students = num\_students + 1**

**WHERE id = 1;**

**UPDATE department**

**SET num\_students = num\_students + 1**

**WHERE id = NEW.department;**

**END;**

**CREATE TRIGGER manager\_insert\_trigger AFTER INSERT ON manager FOR EACH ROW**

**BEGIN**

**UPDATE company**

**SET num\_managers = num\_managers + 1**

**WHERE id = 1;**

**END;**

**2. Create after-update triggers for any two tables of your choice.**

**CREATE TRIGGER manager\_update\_trigger AFTER UPDATE ON manager FOR EACH ROW**

**BEGIN**

**IF OLD.name <> NEW.name THEN**

**UPDATE project**

**SET manager\_name = NEW.name**

**WHERE manager\_id = NEW.id;**

**END IF;**

**IF OLD.department <> NEW.department THEN**

**UPDATE project**

**SET manager\_department = NEW.department**

**WHERE manager\_id = NEW.id;**

**END IF;**

**END;**

**OR**

**-** **CREATE TRIGGER student\_update\_trigger AFTER UPDATE ON student FOR EACH ROW**

**BEGIN**

**IF OLD.department <> NEW.department THEN**

**UPDATE course**

**SET student\_department = NEW.department**

**WHERE student\_id = NEW.id;**

**UPDATE department**

**SET num\_students = num\_students - 1**

**WHERE id = OLD.department;**

**UPDATE department**

**SET num\_students = num\_students + 1**

**WHERE id = NEW.department;**

**END IF;**

**END;**

**3. Create after deleting triggers for any two tables of your choice.**

**CREATE TRIGGER manager\_delete\_trigger AFTER DELETE ON manager FOR EACH ROW**

**BEGIN**

**UPDATE company**

**SET num\_managers = num\_managers - 1**

**WHERE id = 1;**

**DELETE FROM project**

**WHERE manager\_id = OLD.id;**

**END;**

**OR**

**-CREATE TRIGGER student\_delete\_trigger AFTER DELETE ON student FOR EACH ROW**

**BEGIN**

**UPDATE school**

**SET num\_students = num\_students - 1**

**WHERE id = 1;**

**UPDATE department**

**SET num\_students = num\_students - 1**

**WHERE id = OLD.department;**

**DELETE FROM course**

**WHERE student\_id = OLD.id;**

**END;**

**Section VI**

**1. Create a user with your name as username and your student number as password and grant all privileges to the created user.**

**mysql -u root -p**

**create user 'niyonzimagilbert'@'localhost'IDENTIFIED BY '222010015';**

**GRANT ALL PRIVILEGES ON \*.\* TO 'niyonzimagilbert'@'localhost';**

**FLUSH PRIVILEGES;**

**exit;**

**2. Create a user with your "names\_semi" as username and your student number as password and give him insert, update, and delete privileges to the created user.**

**mysql -u root -p**

**CREATE USER 'niyonzima'@'localhost' IDENTIFIED BY '222010015';**

**GRANT INSERT, UPDATE, DELETE ON \*.\* TO 'niyonzima'@'localhost';**

**FLUSH PRIVILEGES;**

**exit;**

**3. Revoke insert privileges to the last user you created.**

**mysql -u root -p**

**REVOKE INSERT ON \*.\* FROM 'niyonzima'@'localhost';**

**FLUSH PRIVILEGES;**

**exit;**

**2.2. TABLES**

**STUDENT TABLE**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **REG\_NUMBER** | **NAMES** | **E-MAIL** | **ADDRESS** | **ACCOUNT\_NUMBER** | **FACULTY** | **PHONE\_NUMBER** | **ADMIN\_ID** | **MANAGER\_ID** | **ACCOUNTANT\_ID** |
| **22232** | **DAVID** | **david@gmail.com** | **Huye** | **54321-67890** | **BIT** | **0789012345** | **001** | **999** | **111** |
| **12345** | **ANGE** | **ange@gmail.com** | **HUYE** | **12345-09876** | **IT** | **0789054321** | **002** | **998** | **112** |

**ADMIN TABLE**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ADMIN\_ID** | **NAMES** | **ADDRESS** | **AGE** | **PHONE\_NUMBER** | **E-MAIL** |
| **001** | **ELSE** | **MUHANGA** | **20** | **0789076543** | **eslse@gmail.com** |
| **002** | **JOHN** | **KAMONYI** | **21** | **0789076566** | **john@gmail.com** |

**MANAGER TABLE**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **MANAGER\_ID** | **NAMES** | **ADDRESS** | **AGE** | **QUALIFICATION** | **PHONE\_NUMBER** | **ADMIN\_ID** | **E-MAIL** |
| **999** | **ERIC** | **MUHANGA** | **30** | **A0** | **0789065784** | **001** | **eric@gmail.com** |
| **998** | **CHRIS** | **KAMONYI** | **32** | **A0** | **0789065437** | **002** | **chris@gmail.com** |

**ACCOUNTANT TABLE**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ACCOUNTANT\_ID** | **NAMES** | **ADDRESS** | **QUALIFICATION** | **PHONE\_NUMBER** | **E-MAIL** | **ADMIN\_ID** |
| **111** | **JEAN** | **HUYE** | **A2** | **0789009876** | **jean@gmail.com** | **001** |
| **112** | **PETER** | **HUYE** | **A0** | **0789012346** | **peter@gmailcom** | **002** |
|  |  |  |  |  |  |  |

**2.3. REFERENCES:**

1. Database Design and Modeling
2. Relational Database Management Systems (RDBMS)
3. Database Security and Compliance
4. Data Privacy and Compliance
5. Online Courses and Tutorials
6. Community Forums and Discussion Groups

PART 3. JAVA PROGRAMING

**3.1. Introduction**

The University Student Savings System is a Java-based application designed to streamline and facilitate financial management for students within a university setting. This system provides a convenient platform for students to deposit, withdraw, and monitor their savings, empowering them to achieve their financial goals effectively. Implemented using Java programming language, this system offers a user-friendly interface and robust functionalities to cater to the diverse needs of students .

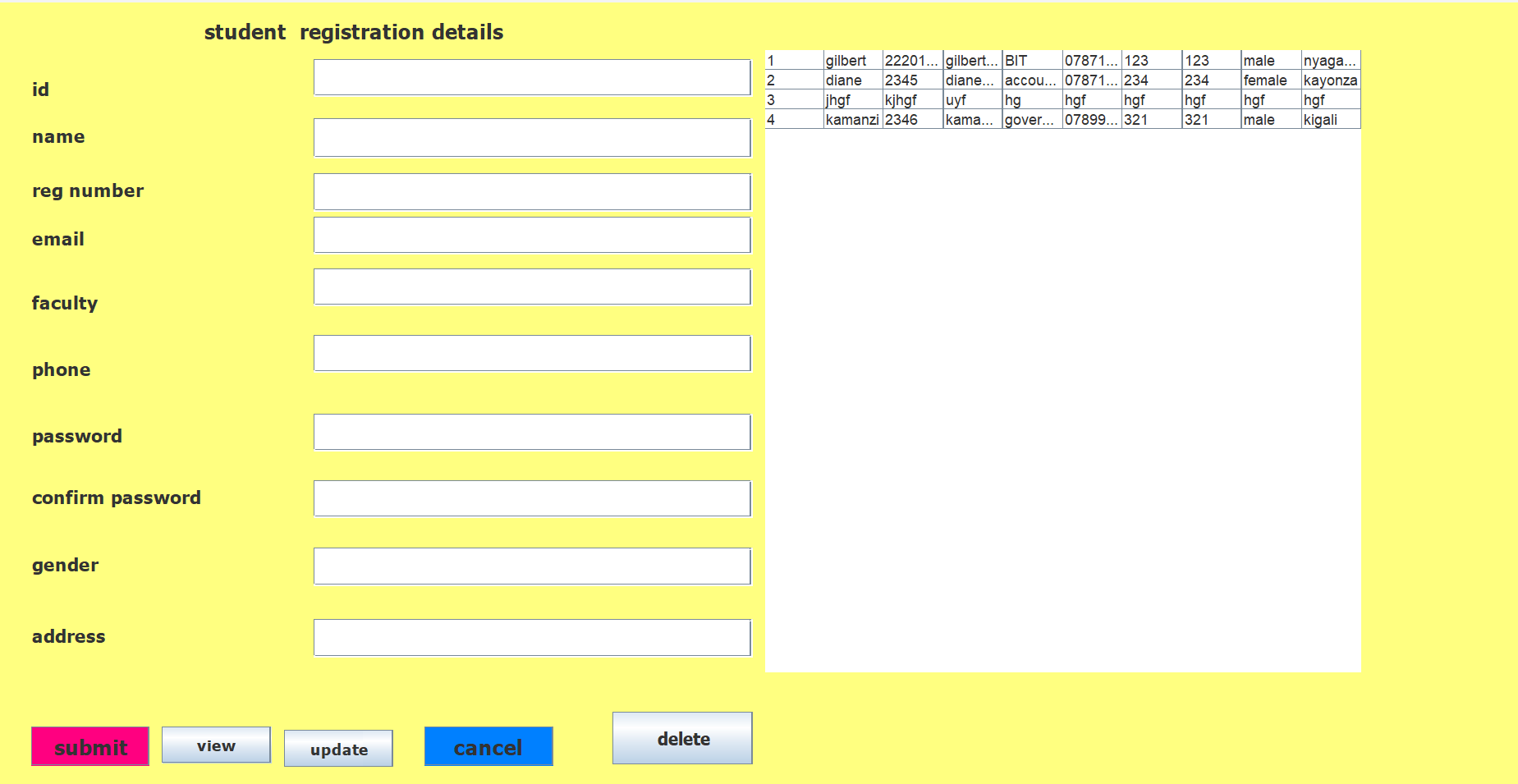
**3.2. HOW SYSTEM WORKS IN JAVA PROGRAMING STEPS BY STEP**

**3.2.1. these are how our system (university of Rwanda students saving system may work steps by step:**

Our system may work where when it start first student make registration in order to have an account in our system that allow him/her to log in and then after log in have access to dashboard of the system where dashboard contains (admin details form, manager details form, accountant details form, account details form, students details form )

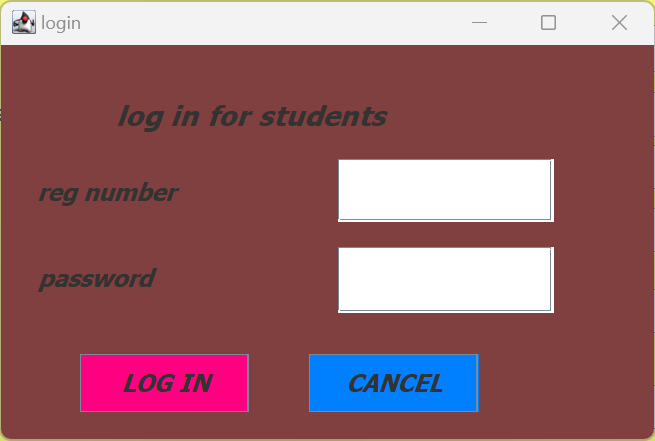
**1 step. Student Registration Form:**

* Students provide personal information such as name, student ID, and contact details.
* They create login credentials (username and password) for accessing the system.

****

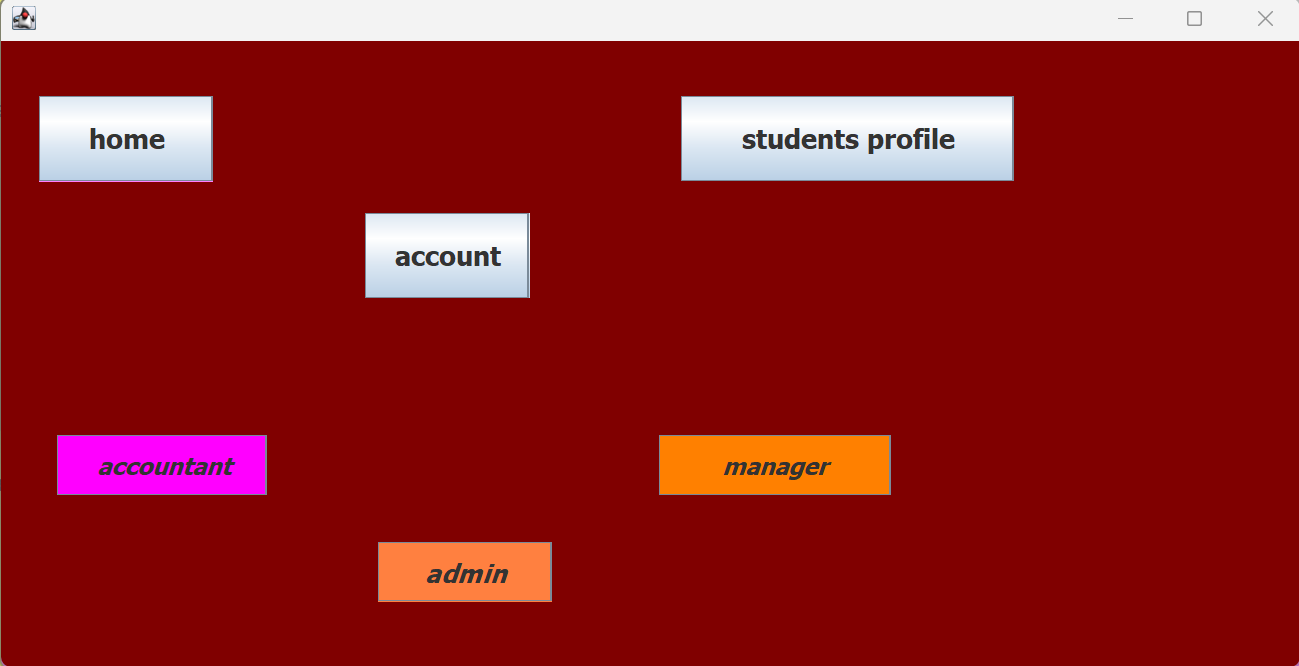
**2 step. Login Form:**

* Students, accountants, managers, and administrators log in using their respective credentials.
* Authentication mechanisms verify the credentials entered by the user.



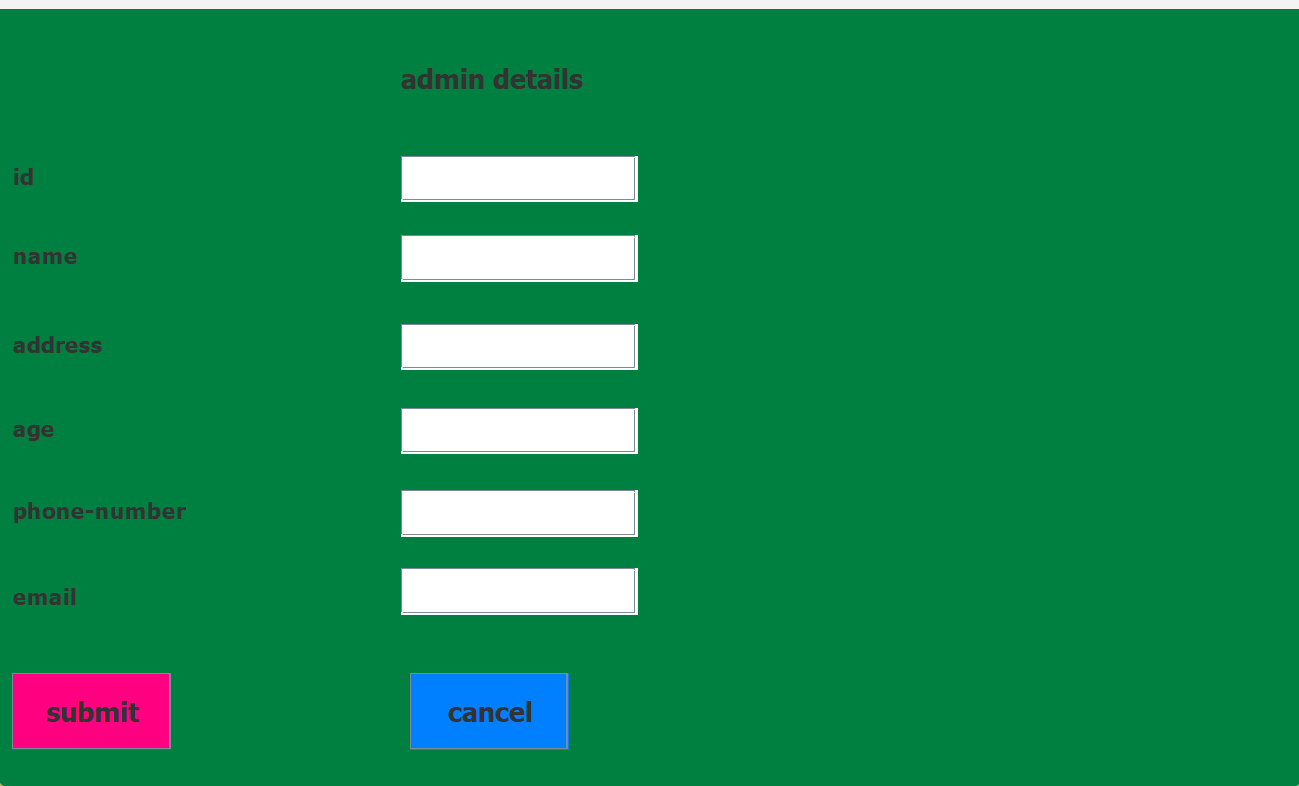
**3 step. Dashboard:**

* Upon successful login, users are directed to their respective dashboards, displaying available options based on their roles.
* The dashboard provides links/buttons to access different forms and functionalities.

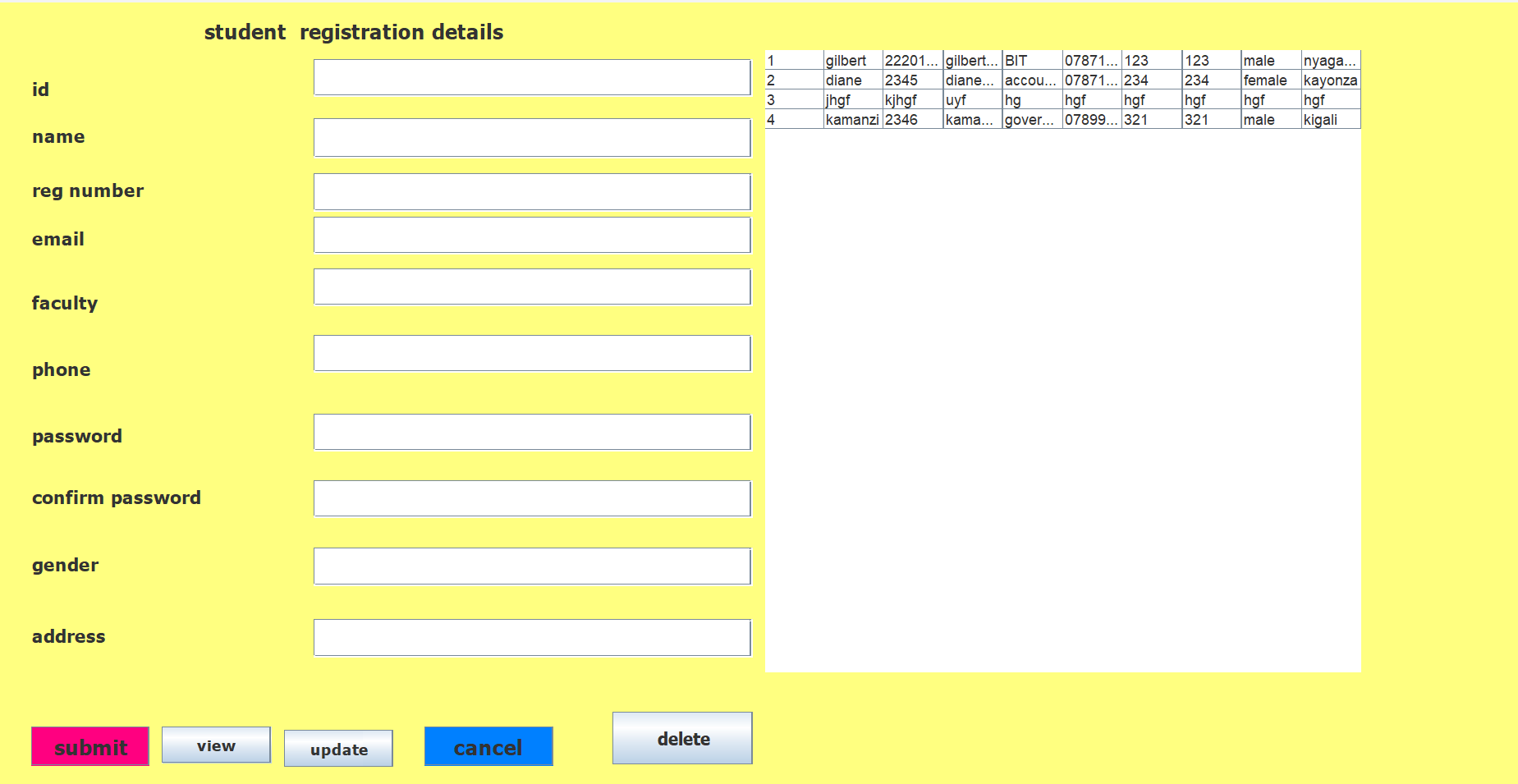


**4 step. Admin Details Form:**

* Accessible to administrators only.
* Allows administrators to manage user accounts, including adding, removing, or modifying user details.
* Provides functionalities for system configuration, such as setting permissions and managing system settings.

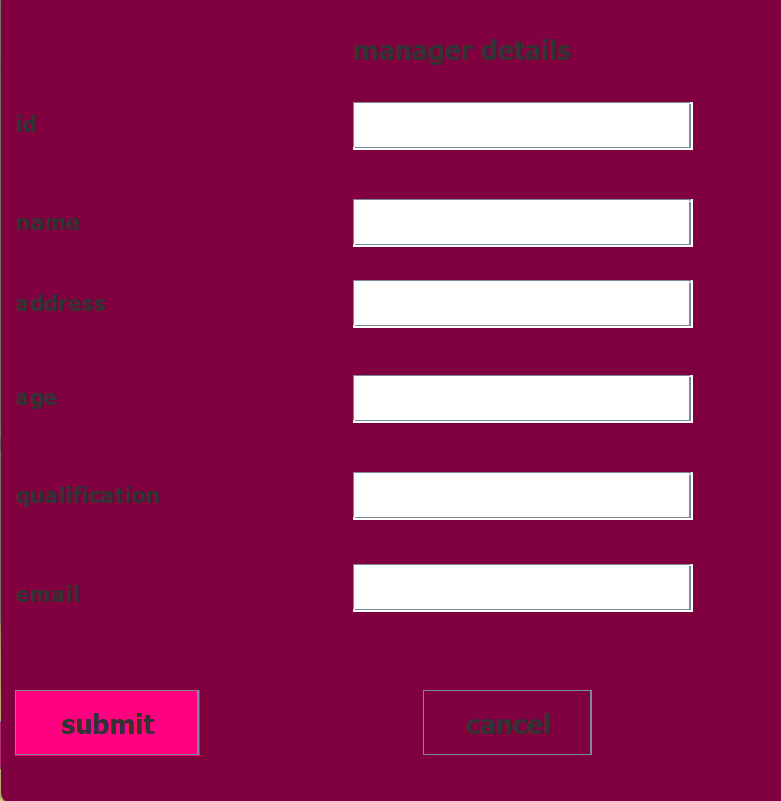


**5 step. Student Details Form:**

* Accessible to students.
* Students can view their account information, including current balance, transaction history, and personal details.
* Provides options for depositing funds, withdrawing funds, and viewing transaction details.
* 

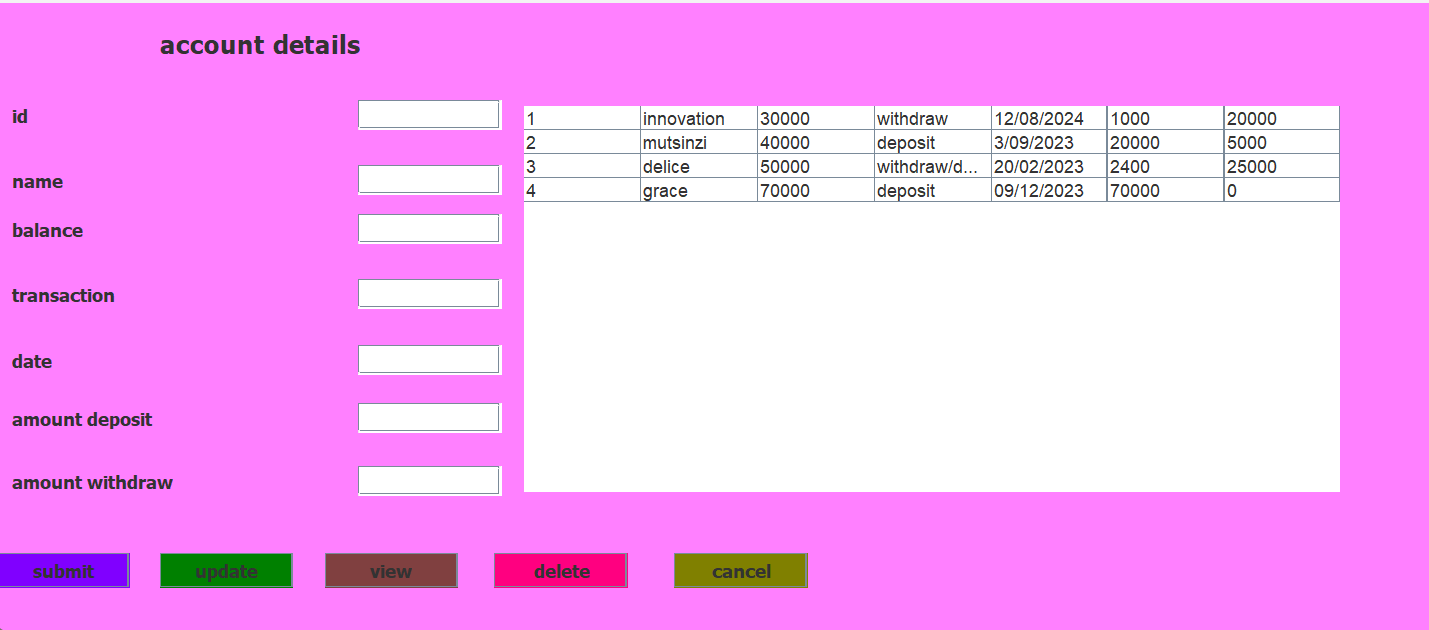
**6 step. Manager Details Form:**

* Accessible to managers.
* Provides managerial functionalities such as monitoring system performance, generating reports, and analyzing data.
* Allows managers to oversee the overall operation of the savings system and make strategic decisions.



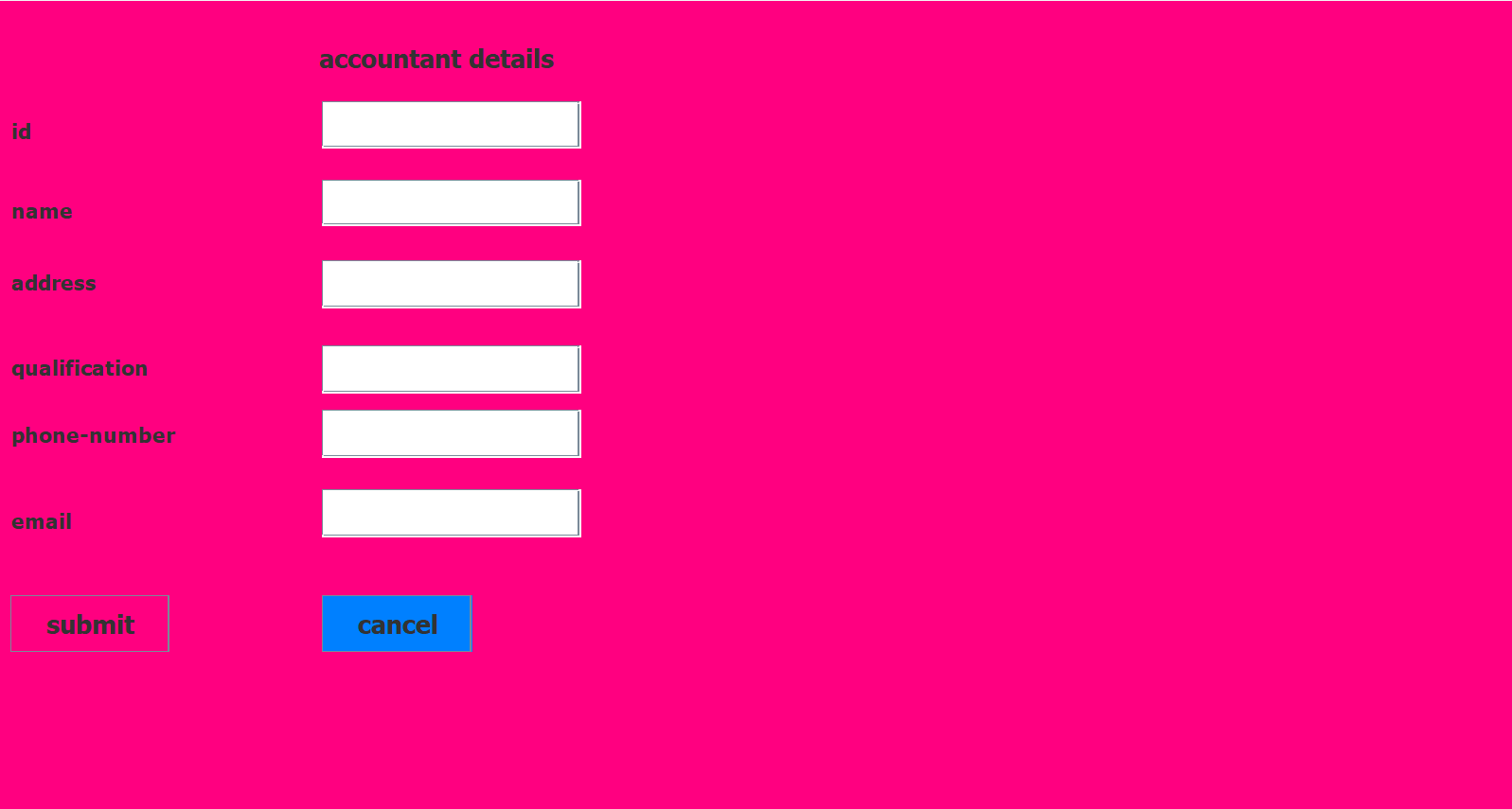
**7 step. Account Details Form:**

* Accessible to authorized users (e.g., administrators, managers, accountants).
* Allows users to view and manage student accounts, including account balances, transaction history, and account status.
* Provides functionalities for processing financial transactions such as deposits, withdrawals, and transfers.



**8 step. Accountant Details Form:**

* Accessible to accountants.
* Enables accountants to manage financial transactions, including recording deposits, withdrawals, and transfers.
* Provides options for generating financial reports, reconciling accounts, and maintaining financial records.



**Java Programming Implementation:**

* Implement the system using Java programming language, incorporating object-oriented principles for modular and maintainable code.
* Utilize Java frameworks such as JavaFX or Swing for creating graphical user interfaces (GUI) for forms and dashboards.
* Implement secure authentication and authorization mechanisms to control user access based on roles and permissions.
* Integrate with a relational database management system (e.g., MySQL, PostgreSQL) to store and manage user data, account details, and transaction records.

**Conclusion:**

* The University of Rwanda Student Savings System implemented in Java programming provides a structured and efficient platform for managing student finances.
* With role-based access control and user-friendly interfaces, the system facilitates seamless interaction for students, administrators, accountants, and managers, ensuring effective financial management within the university community.

**Functional buttons are:** submit, update, delete, cancel, view

**3.3. REFERENCES:**

1. Java Programming Resources:
2. Database Integration
3. GUI Development:
4. Security Implementation:
5. Online Courses and Tutorials
6. Open Source Projects and Github