Industrial Internship Report on

"Bank Information System"

Prepared by

HARSHITH.H.D.

Executive Summary
This report provides details of the Industrial Internship provided by upskill Campus and The IoT Academy in collaboration with Industrial Partner UniConverge Technologies Pvt Ltd (UCT). This internship was focused on a project/problem statement provided by UCT. We had to finish the project including the report in 4 weeks' time. My project was Quiz Game Application made using Python Programming language. This internship gave me a very good opportunity to get exposure to Industrial problems and design/implement solution for that. It was an overall great experience to have this internship.

TABLE OF CONTENTS

1 Preface

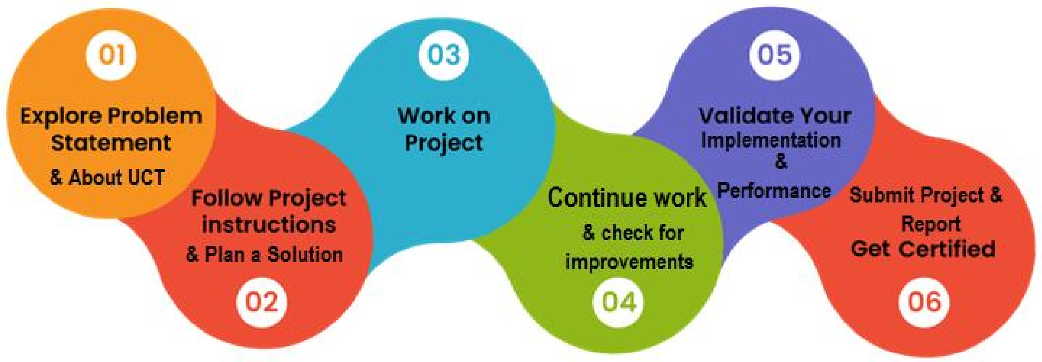
The internship of 4 weeks was helpful because it allowed me to explore more about the Python programming language.The recorded classes were very helpful which were about numpy and pandas.I learnt many unknown details about these libraries in these videos.The notes were improving our vision about the subjects.The Quiz after every week were helpful , which was used to check ourselves about the topics which we have learnt in the session/week.

I needed this kind of internship because it helps many students and graduates to improve,analyse and enhance their skills on their respective selected domain.

I choosed the project "Quiz Game".Quiz game is the game which tests the knowledge and improve their knowledge on the certain topic.I like to play quiz game.So, I choosed "Quiz Game".I have used many libraries to create a User Interface and used files which stores the questions and answers of the respective questions.

Opportunity given by USC/UCT.

The program was planned as shown below:



I have learnt many things in this internship and this is agood experience which I will take this in my future as a learning.

I Thank to all friends,teachers and family, who have helped me directly or indirectly.

I will say one thing to my juniors and peers that join this internship to upgrade the skill and knowledge on your favourite domains and programming language which gives you a good experience.

2 Introduction

2.1 About UniConverge Technologies Pvt Ltd

A company established in 2013 and working in Digital Transformation domain and providing Industrial solutions with prime focus on sustainability and RoI.

For developing its products and solutions it is leveraging various **Cutting Edge Technologies e.g. Internet of Things (IoT), Cyber Security, Cloud computing (AWS, Azure), Machine Learning, Communication Technologies (4G/5G/LoRaWAN), Java Full Stack, Python, Front end** etc.

uct
Uniconverge Technologies
IIOT Products
We offer product ranging from Remote IOs, Wireless IOs, LoRaWAN Sensor Nodes/ Gateways, Signal converter and IoT gateways
IIOT Solutions
We offer solutions like OEE, Predictive Maintenance, LoRaWAN based Remote Monitoring, IoT Platform, Business Intelligence…
OEM Services
We offer solutions ranging from product design to final production we handle everything for you..

i. UCT IoT Platform (uct Insight)

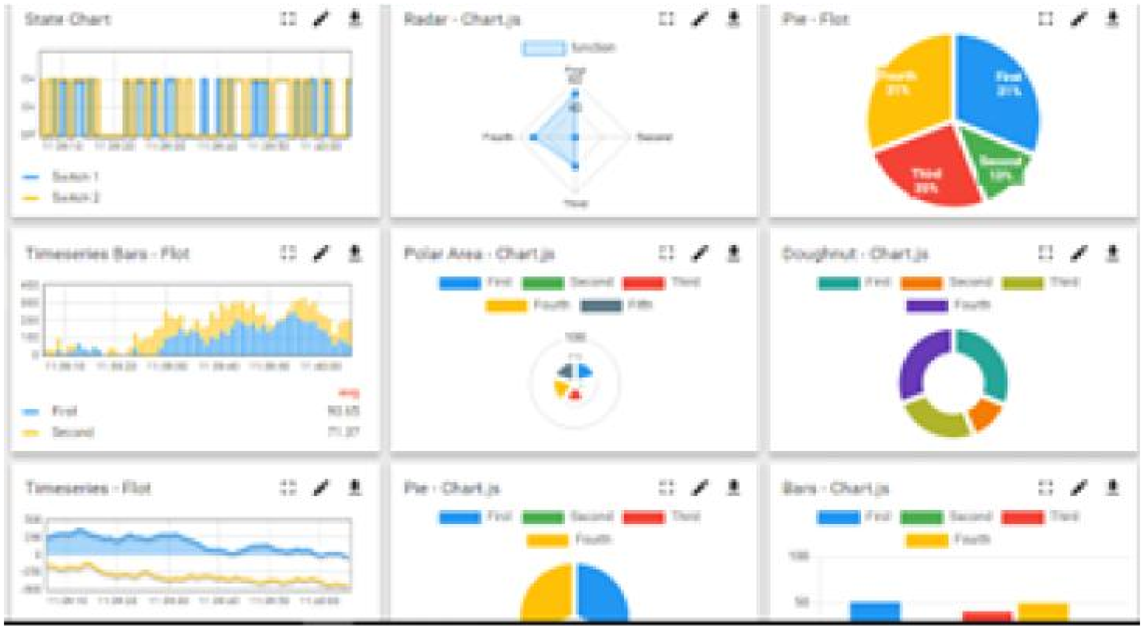
UCT Insight is an IOT platform designed for quick deployment of IOT applications on the same time providing valuable "insight" for your process/business. It has been built in Java for backend and ReactJS for Front end. It has support for MySQL and various NoSql Databases.

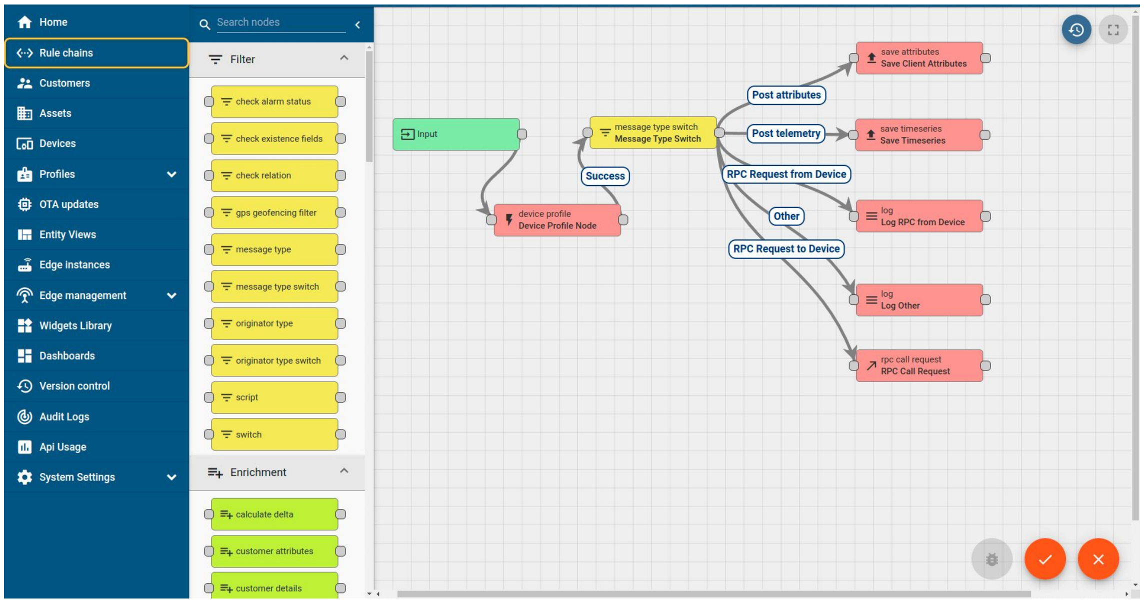
- It enables device connectivity via industry standard IoT protocols - MQTT, CoAP, HTTP, Modbus TCP, OPC UA

- It supports both cloud and on-premises deployments.

It has features to

• Build Your own dashboard
• Analytics and Reporting
• Alert and Notification
• Integration with third party application(Power BI, SAP, ERP)
• Rule Engine





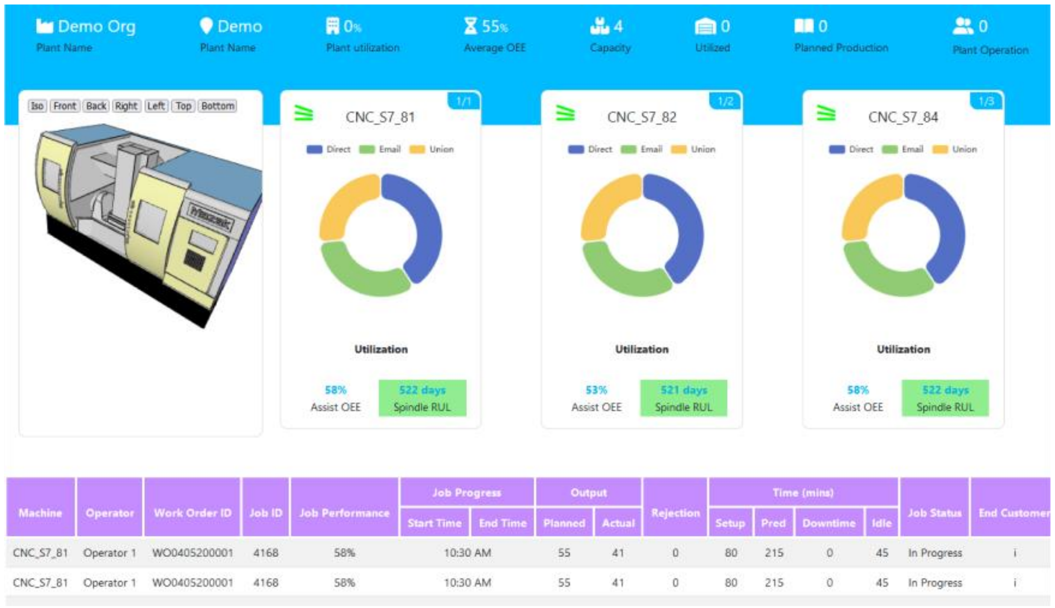
ii. Smart Factory Platform (FACTORY WATCH)

Factory watch is a platform for smart factory needs.

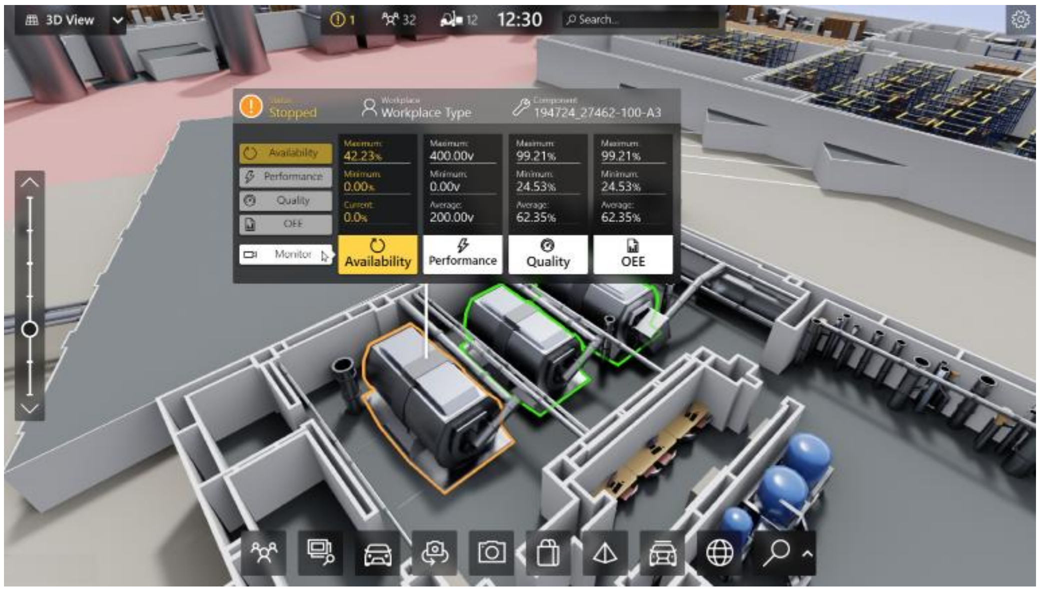
It provides Users/ Factory

- with a scalable solution for their Production and asset monitoring
- OEE and predictive maintenance solution scaling up to digital twin for your assets.
- to unleased the true potential of the data that their machines are generating and helps to identify the KPIs and also improve them.
- A modular architecture that allows users to choose the service that they what to start and then can scale to more complex solutions as per their demands.

Its unique SaaS model helps users to save time, cost and money.



Machine	Operator	Work Order ID	Job ID	Job Performance	Job Progress		Output		Rejection	Time (mins)				Job Status	End Customer
					Start Time	End Time	Planned	Actual		Setup	Pred	Downtime	Idle		
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i

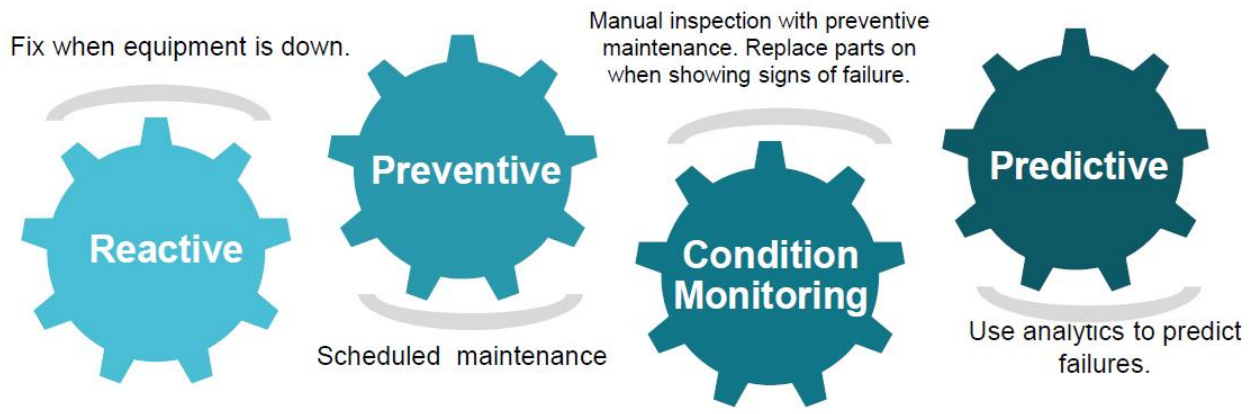


iii. LoRaWAN™ based Solution

UCT is one of the early adopters of LoRAWAN teschnology and providing solution in Agritech, Smart cities, Industrial Monitoring, Smart Street Light, Smart Water/ Gas/ Electricity metering solutions etc.

iv. Predictive Maintenance

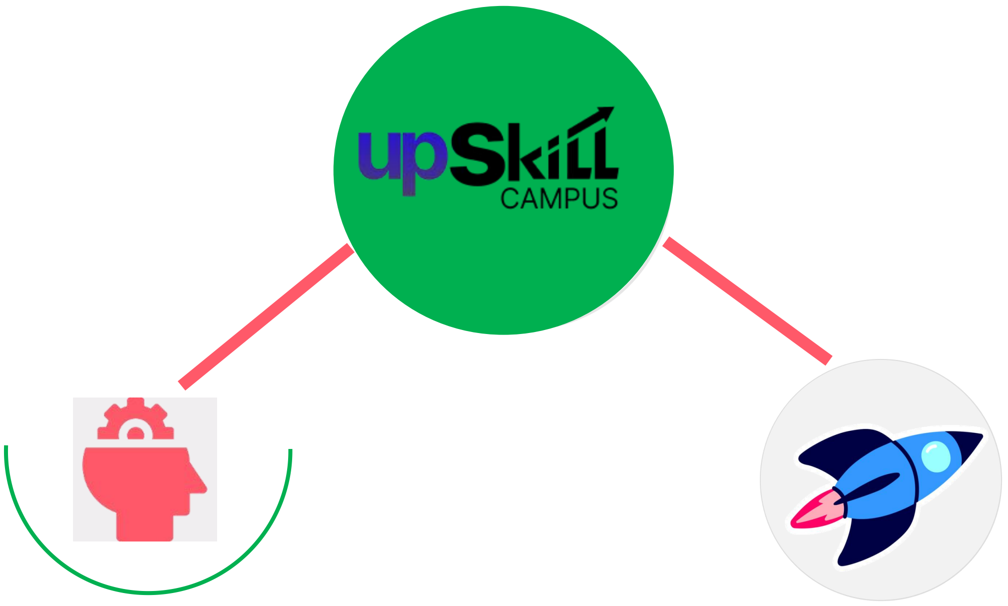
UCT is providing Industrial Machine health monitoring and Predictive maintenance solution leveraging Embedded system, Industrial IoT and Machine Learning Technologies by finding Remaining useful life time of various Machines used in production process.


Fix when equipment is down.
Reactive
Preventive
Scheduled maintenance
Manual inspection with preventive maintenance. Replace parts on when showing signs of failure.
Condition Monitoring
Predictive
Use analytics to predict failures.

2.2 About upskill Campus (USC)

upskill Campus along with The IoT Academy and in association with Uniconverge technologies has facilitated the smooth execution of the complete internship process.

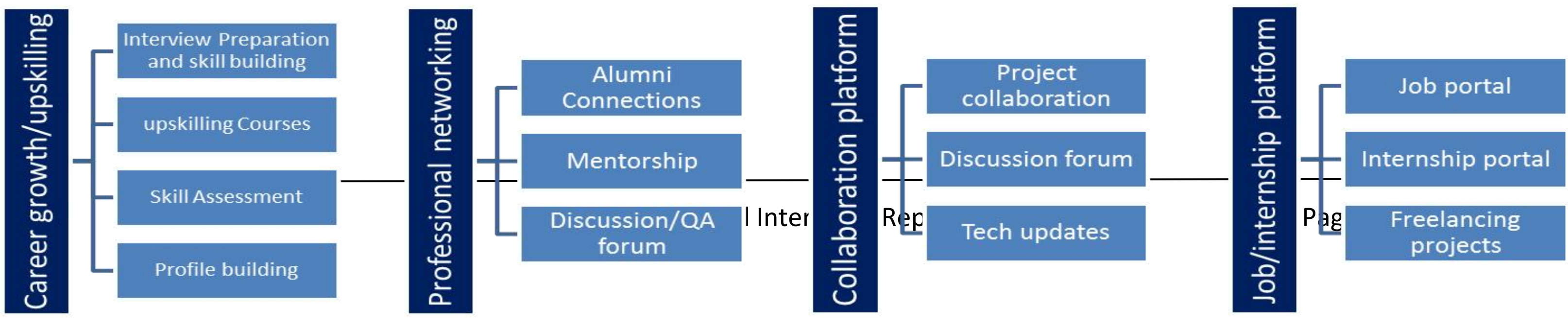
USC is a career development platform that delivers **personalized executive coaching** in a more affordable, scalable and measurable way.



Seeing need of upskilling in self paced manner along-with additional support services e.g. Internship, projects, interaction with Industry experts, Career growth Services

upSkill Campus aiming to upskill 1 million learners in next 5 year

https://www.upskillcampus.com/



2.3 The IoT Academy

The IoT academy is EdTech Division of UCT that is running long executive certification programs in collaboration with EICT Academy, IITK, IITR and IITG in multiple domains.

2.4 Objectives of this Internship program

The objective for this internship program was to

☛ get practical experience of working in the industry.

☛ to solve real world problems.

☛ to have improved job prospects.

☛ to have Improved understanding of our field and its applications.

☛ to have Personal growth like better communication and problem solving.

2.5 Reference

[1] https://learn.upskillcampus.com/

[2] https://www.geeksforgeeks.org/java-application-to-implement-bank-functionality/

2.6 Glossary

Terms	Acronyms
JavaFX	special effects in the Java language
Java	It is a Object Oriented Programming language.
MVC	Model-View-Controller, a design pattern for organizing code in web applications
Java.io	It is a package in java used to perform input and output operations in a file.
CSV	Comma Seperated Values.
class	It is an entity which contains functions and variables which is accessed by using objects.

3 Problem Statement

3.1.1 Problem Statement for Banking System Application (Java GUI)

Problem Overview:

The project aims to develop a **Banking System Application** using **Java** with a **Graphical User Interface (GUI)**. The application will simulate essential banking operations such as creating new accounts, viewing balances, making deposits, and withdrawing funds. Instead of using a traditional database like **JDBC**, the application will rely on **CSV files** to manage and store account data.

The application must be user-friendly and efficient, ensuring secure handling of user data while providing seamless interaction through a GUI. The system must also handle invalid operations, such as overdrafts or invalid input, with appropriate error messages.

3.1.2 Scope of the Project

3.1.2.1 Functional Scope:

Account Management:

1. **Account Creation**: Users should be able to create new accounts, providing their name, initial deposit, and account type. The account information will be stored in a **CSV file**.
2. **Account Login**: Users must log in using their account details, which will be validated by reading the **CSV file**.
2.

Banking Operations:

1. **View Balance**: After logging in, users can view their current account balance, which will be retrieved from the **CSV file**.
2. **Deposit Money**: Users can deposit money by entering an amount. The system will validate the input and update the balance in the **CSV file**.
3. **Withdraw Money**: Users can withdraw money by entering an amount. The system will check if there is enough balance, and if valid, it will deduct the amount from the balance and update the **CSV file**.

Data Persistence using CSV:

1. **CSV-based Data Storage**: Instead of a traditional database, user accounts and transactions will be stored in a **CSV file**. The application will read from and write to the CSV file to manage account data.
2. **Transaction Records**: Each transaction (deposit or withdrawal) will be logged in the **CSV file** to maintain a history of the user's banking activities.

User Interface (GUI):

1. The application will have a **Java Swing** or **JavaFX**-based GUI to provide an intuitive and user-friendly experience. Users will navigate through options such as viewing balances, making deposits, withdrawing funds, and exiting.
2. The interface will display data read from the **CSV files** and allow users to perform banking operations.

Error Handling:

1. The system should handle cases like insufficient balance during withdrawals or invalid input formats when entering transaction amounts.
2. Proper error messages will guide the user when an invalid action is performed (e.g., attempting to withdraw more than the available balance).

3.1.2.2 Non-Functional Scope:

Security:

1. Since CSV files are being used to store sensitive account data, the application should ensure data integrity by handling file read/write operations securely and preventing unauthorized access.
2. Basic user authentication (login system) should ensure only authorized users can access their accounts.

Performance:

1. Even with a CSV file storage system, the application should maintain fast read and write operations to ensure smooth interaction with the user interface.
2. The system should be optimized to handle a large number of accounts and transactions without noticeable delays.

Usability:

1. The GUI will be simple and easy to navigate. Users with minimal technical knowledge should be able to create accounts, deposit money, withdraw funds, and view balances without confusion.

Maintainability:

1. The application will be built using modular code to ensure that future updates or feature additions (such as improved account tracking or new transaction types) can be made easily.
2. Using the **MVC (Model-View-Controller)** pattern will ensure the separation of data, logic, and presentation, making the code more maintainable and scalable.

3.1.3 Use Cases for CSV Data Management:

Account Creation:

1. The user inputs details like name and initial deposit.
2. The system checks for any existing accounts with the same name.
3. The system appends the new account information to the **CSV file**.

Viewing Balance:

1. After login, the user's account information is retrieved from the **CSV file**, and their balance is displayed in the GUI.

Deposit/Withdraw Operations:

1. The user inputs the amount they wish to deposit/withdraw.
2. The system reads the **CSV file** to check the balance, validates the operation, and updates the **CSV file** with the new balance after the transaction is completed.

4 Existing and Proposed solution

In current banking software, complex interfaces can be challenging for users, especially in small-scale banking systems. My proposed solution is a simple and user-friendly GUI that allows basic operations to be conducted efficiently, minimizing user errors and streamlining processes.

4.1 Code submission (Github link):

4.2 Report submission (Github link) : first make placeholder, copy the link.

5 Proposed Design/ Model

5.1.1 Proposed Design for Banking System Application (Java GUI with CSV Storage)

The proposed design for the Banking System Application is based on the **Model-View-Controller (MVC)** architectural pattern, which ensures separation of concerns, making the system more maintainable and scalable. The application will handle user interactions through a **Graphical User Interface (GUI)**, perform operations like deposits and withdrawals, and store data using **CSV files**.

5.1.2 Model-View-Controller (MVC) Pattern

The MVC pattern divides the application into three main components:

1. **Model**: Represents the data and business logic (banking operations).
2. **View**: Represents the GUI that the user interacts with.
3. **Controller**: Handles the user inputs and updates the Model and View accordingly.

5.1.3 1. Model (Data and Business Logic)

The **Model** is responsible for the core business logic of the application. In this case, it handles the creation of accounts, retrieval of balances, deposit and withdrawal operations, and updating the data in the **CSV file**.

5.1.3.1 Key Components:

Account Class: Represents a bank account, storing attributes such as:

- o Account holder's name.
- o Account number.
- o Balance.

CSV File Handling: This part of the Model handles reading and writing to the CSV file, which acts as the data storage for accounts and transaction logs. Operations performed include:

- o **Reading CSV**: Fetching account information and balance.

- **Writing CSV**: Updating the balance after deposits/withdrawals or adding new account details.

5.1.3.2 *Business Logic Operations:*

Create Account: Add a new account by storing its details in the CSV file. The system should ensure account uniqueness (e.g., checking if an account with the same name already exists).

View Balance: Retrieve the account balance by searching through the CSV file based on the user's account information.

Deposit: Validate the amount entered, and if valid, update the CSV file with the new balance.

Withdraw: Check whether the account has sufficient funds for withdrawal. If valid, deduct the amount and update the CSV file.

5.1.3.3 *File Structure:*

The CSV file will store information in a structured format, such as:

```
Account_Number, Name, Balance
101, John Doe, 5000
102, Jane Smith,8000
```

5.1.4 2. View (User Interface)

The **View** represents the **Graphical User Interface (GUI)**, built using **Java Swing** or **JavaFX**, which the user interacts with to perform banking operations.

5.1.4.1 *Key Components of the GUI:*

Login Screen: Allows the user to enter their account number and log in to access the system.

Main Menu: After logging in, the user can choose from the following options:

- View balance.
- Make a deposit.
- Make a withdrawal.
- Exit the application.

Forms for Deposits/Withdrawals: These screens will allow users to input the amount they want to deposit or withdraw. The system will validate the input and display the updated balance or an error message in case of invalid operations.

5.1.4.2 *User Interaction:*

- The GUI will guide the user through the application with prompts and buttons. For example:
  - After login, the user is presented with a menu to either view their balance, deposit money, or withdraw money.
  - Error messages will be displayed if the user enters invalid input (e.g., negative numbers, overdraft attempts).
  - All updates to account balances and transaction results will be reflected in real-time on the GUI.

5.1.4.3 *Main GUI Components:*

1. **Menu Screen**: Buttons for navigating to different operations (e.g., "View Balance", "Deposit", "Withdraw", "Exit").
2. **Forms for Operations**: Separate input forms for performing operations like deposits or withdrawals. These forms will collect input, validate it, and then trigger the appropriate actions in the Model.
3. **Error Handling**: Display pop-up error messages for any invalid actions like overdraft attempts, incorrect input, or missing data.

5.1.5 3. Controller (Input Handling and Coordination)

The **Controller** acts as the middleman between the **Model** and the **View**. It processes user input from the GUI, makes requests to the Model, and updates the View accordingly.

5.1.5.1 *Key Responsibilities:*

Handling User Actions:

- The Controller receives user inputs (e.g., clicking buttons, entering deposit/withdrawal amounts) from the GUI.
- It passes the relevant data to the Model for processing, such as retrieving the account balance or performing a transaction.

Validating Input:

- Before passing data to the Model, the Controller ensures that the input is valid (e.g., positive amounts for deposits, sufficient funds for withdrawals).

Updating the View:

- After the Model processes the data (e.g., after a deposit or withdrawal), the Controller will instruct the View to refresh and display the updated information (e.g., show the updated balance).

Managing CSV Interaction:

- The Controller ensures that the Model's methods for reading and writing to the **CSV files** are executed correctly, ensuring that the account data is updated after every transaction.

5.1.6 Data Flow in the MVC Pattern:

User Action (Input):

- The user clicks a button on the GUI (View) or enters data (e.g., deposit amount). This triggers an event handled by the Controller.

Controller Processing:

- The Controller validates the user input and sends the request to the Model.

Model Operations:

- The Model reads/writes data from/to the **CSV file** (e.g., updates the balance after a deposit).

View Update:

- Once the Model completes the operation, the Controller updates the View (e.g., displays the new balance on the GUI).

5.1 High Level Diagram :

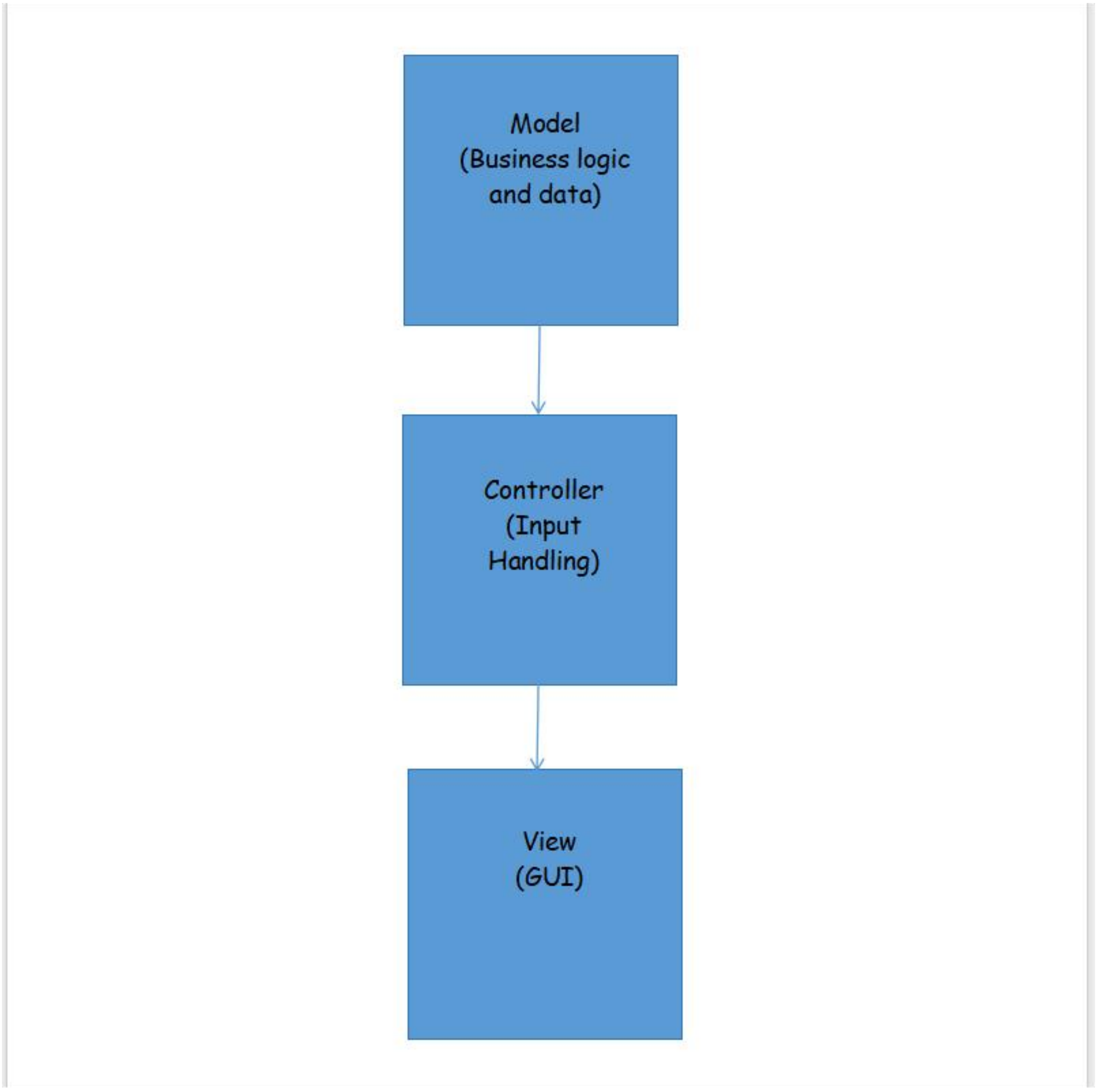
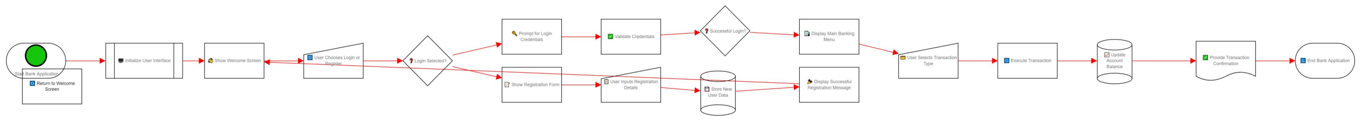


Figure 1: HIGH LEVEL DIAGRAM OF THE SYSTEM

5.2 Interfaces :


Start Bank Application
Return to Welcome Screen
Initialize User Interface
Show Welcome Screen
User Chooses Login or Register
Login Selected?
Prompt for Login Credentials
Validate Credentials
Successful Login?
Display Main Banking Menu
Show Registration Form
User Inputs Registration Details
Store New User Data
Display Successful Registration Message
User Selects Transaction Type
Execute Transaction
Update Account Balance
Provide Transaction Confirmation
End Bank Application

This flowchart explains about the working of the program of my Bank Application.

6 Performance Test

Performance testing is critical to ensure that your **Banking System Application** operates efficiently and can handle user interactions smoothly. Given that the application is built using **Java (Swing/JavaFX)** with **CSV files** for data storage, performance testing focuses on the application's ability to manage file I/O, GUI responsiveness, and overall execution time during typical banking operations.

5.1.7 1. Key Areas of Performance Testing

The performance testing for this application will assess the following key areas:

File I/O Performance (CSV Storage):

1. Since the application uses **CSV files** to store and retrieve account data, file read/write operations need to be efficient, especially when dealing with large amounts of data.
2. File operations should not cause noticeable delays when users perform actions like viewing balances, making deposits, or withdrawals.

GUI Responsiveness:

1. The **Java Swing/JavaFX** GUI should be responsive to user inputs, displaying results (e.g., updated balances) in real-time without noticeable lag.
2. The user experience must remain smooth, regardless of the complexity of operations or the size of the dataset.

Memory Usage:

1. The application should manage system resources (like memory) efficiently, especially during file read/write operations, avoiding unnecessary memory consumption.
2. The system should handle multiple user accounts and transactions without memory overflows or excessive use of resources.

Execution Time:

1. The time taken to complete operations like reading the CSV file, writing updates, or refreshing the GUI should remain minimal, even as the amount of stored data grows.

5.1.8 2. Performance Test Plan

5.1.8.1 Test Case 1: File Reading Performance

Objective: Evaluate the time taken to read data from the CSV file, especially when the number of user accounts increases.

Test Procedure:

1. Create a CSV file with an increasing number of records (e.g., start with 10 accounts, then test with 100, 1000, etc.).
2. Perform operations that require reading from the file (e.g., viewing balance after login).
3. Measure the time taken to load the data from the CSV file and display it in the GUI.

Expected Outcome:

- The application should read the CSV file and display the account balance within a reasonable time (e.g., <1 second for 1000 accounts). File reading should not significantly degrade as the dataset size grows.

5.1.8.2 Test Case 2: File Writing Performance (Deposit/Withdraw)

Objective: Assess the efficiency of writing updates to the CSV file, especially after a deposit or withdrawal operation.

Test Procedure:

1. After a user logs in, perform a deposit or withdrawal operation.
2. Measure the time taken to update the balance in the CSV file after the operation.
3. Repeat the process for varying numbers of accounts in the CSV file.

Expected Outcome:

- The application should be able to update the CSV file quickly, ensuring that the deposit/withdrawal is reflected in the balance without noticeable delay (e.g., <1 second for typical operations).

5.1.8.3 Test Case 3: GUI Responsiveness

Objective: Ensure the GUI remains responsive when performing user actions, such as switching between different screens, viewing balances, and executing transactions.

Test Procedure:

1. Navigate through the application's GUI (e.g., log in, view balance, make deposits and withdrawals).
2. Check if the GUI responds immediately to user inputs (e.g., clicking buttons or submitting forms).
3. Introduce stress by performing multiple transactions in quick succession or working with large datasets (e.g., 1000+ accounts in the CSV file).
4. Measure the delay in response time for each user action.

Expected Outcome:

- The GUI should remain responsive with minimal lag when navigating between screens or performing transactions. The system should handle multiple interactions without freezing or significant delays.

5.1.8.4 Test Case 4: Memory Usage

Objective: Evaluate the application's memory usage during normal operation and under stress (with a large number of accounts).

Test Procedure:

1. Start the application and monitor its memory usage at various stages (e.g., during login, viewing balances, and performing transactions).
2. Simulate the use of large datasets (e.g., CSV files with thousands of accounts).
3. Identify whether the memory usage increases significantly as more data is loaded or more transactions are performed.

Expected Outcome:

- Memory usage should remain stable and within acceptable limits. The application should not consume excessive memory, even with a large number of user accounts or transactions.

5.1.8.5 *Test Case 5: Overall Execution Time*

Objective: Measure the overall time taken by the system to complete common banking operations, including file reading/writing, balance updates, and GUI rendering.

Test Procedure:

1. Execute various banking operations (e.g., login, view balance, deposit, withdraw).
2. Measure the time taken for the entire operation, from user input to displaying the result in the GUI.
3. Perform the test under different conditions, such as varying file sizes and different numbers of user accounts.

Expected Outcome:

- The overall execution time for typical operations should be minimal (e.g., under 2 seconds for viewing balances or making transactions). Performance should not degrade significantly as the dataset grows.

5.1.9 3. Optimization Strategies

Based on the results of the performance tests, the following optimizations can be applied if performance bottlenecks are identified:

CSV File Handling Optimization:

- If file reading or writing is slow, consider using buffered I/O to reduce file access time.
- Avoid loading the entire CSV file into memory if not necessary. Load only the relevant portions (e.g., the user's account) to minimize memory usage and speed up access.

Asynchronous GUI Updates:

- To maintain GUI responsiveness, use background threads for operations that take longer to complete (e.g., reading large files). This prevents the GUI from freezing while waiting for file operations to finish.

Lazy Loading of Data:

- Instead of reading the entire CSV file at once, implement **lazy loading** where data is only read when needed. For example, load account data only when a user logs in, rather than loading the entire file at startup.

Memory Management:

- Use proper memory management techniques to avoid memory leaks. Ensure that objects are dereferenced when no longer needed, especially when working with large datasets.

Profiling and Debugging:

- Use profiling tools (e.g., **VisualVM** for Java) to identify memory leaks, long-running processes, or inefficient code segments. Focus optimization efforts on the parts of the system that consume the most resources.

5.1.10 4. Profiling Tools

To accurately measure performance, the following tools can be used:

- **JVisualVM**: A Java profiler that can track memory usage, CPU time, and thread activity. It helps identify bottlenecks in the code, such as inefficient file handling or memory leaks.
- **cProfile** (for file operations): Can be used to analyze the time spent in various methods, particularly for file read/write operations.
- **JConsole**: Monitors Java applications' performance, including memory consumption and thread execution.

5.1.11 5. Performance Outcome

After conducting the performance tests, the expected results should be:

1. **File I/O Performance**: Efficient reading and writing to the CSV file, with minimal delays as the number of accounts grows.
2. **GUI Responsiveness**: The GUI should remain responsive, even with large datasets, ensuring a smooth user experience.
3. **Memory Usage**: Memory usage should be stable, with no significant increase even with a large number of accounts or transactions.
4. **Overall Execution Time**: The system should execute typical banking operations quickly (within 1–2 seconds), without noticeable delays.

7 My learnings

During this internship, I deepened my understanding of Java programming, specifically the Swing library for GUI development. I also became proficient in integrating Java with databases through JDBC, handling data flow between the user interface and backend operations. The challenges I faced in managing GUI responsiveness and data accuracy have improved my problem-solving and debugging skills.

8 Future work scope

There are several areas where this project can be expanded:

1. **Database Integration**: Introduce a more advanced DBMS such as MySQL for efficient data handling.
2. **User Authentication**: Implement user authentication and secure login features.
3. **Advanced Banking Features**: Add functionalities like loan management, interest calculation, and financial reports.
4. **Improved GUI**: Refine the GUI with more advanced Java libraries like JavaFX for better user experience.