T.C. ADANA ALPARSLAN TÜRKEŞ UNIVERSITY
FACULTY OF COMPUTER AND INFORMATICS
DEPARTMENT OF COMPUTER ENGINEERING

CEN 303

SOFTWARE ENGINEERING

BANK MANAGEMENT SYSTEM

Prepared by

Berkay Dursun	210101157
Murat Eker	200101007
Muharrem Demir	200101010

Instructor

Assoc. Prof. Dr. Mümine KAYA KELEŞ

CONTENTS

1. INTRODUCTION

1.1 Purpose and Scope

1.1.1 The Purpose of the Project

The aim of the project is to mange bank systems and provide high-quality service to customers.

1.1.2 Scope of The Project

This project is produced to used by both the system manager and the customers of the bank. The project aims to deliver a comprehensive digital banking solution, focusing on essential functionalities and providing a secure and user-friendly experience for both customers and bank personnel.

1.2 Goals and Success Criteria

Goals:

- Fast and User-Friendly Application
- High Availability
- Core Banking Operations
- Security and Authentication
- Customer Satisfaction
- Delivery As Fast As Possible
- Innovative Design

Success Criteria:

- Smooth Operation
- Timely Delivery
- Good Planning
- Follow The Project Stages
- User-Friendly Interface
- Follow Customer Feedbacks

1.3 Overview

1.3.1 System requirements:

- Must be a user login
- Must be an admin login
- Must have user restrictions
- Admin should manage the system
- Customer is able to make core bank operations

1.3.2 Member Requirements:

- Every member must have a username and password
- Making the authorization
- Every customer must have an bank account

1.3.3 Test Items:

1.3.3.1 Features To Be Tested

- Login/logout
- Interface
- Record
- Security
- Consistency

1.3.3.2 Features Not To Be Tested

- Overloads
- Hardware errors

1.3.3.3 Item Pass/Fail Criteria

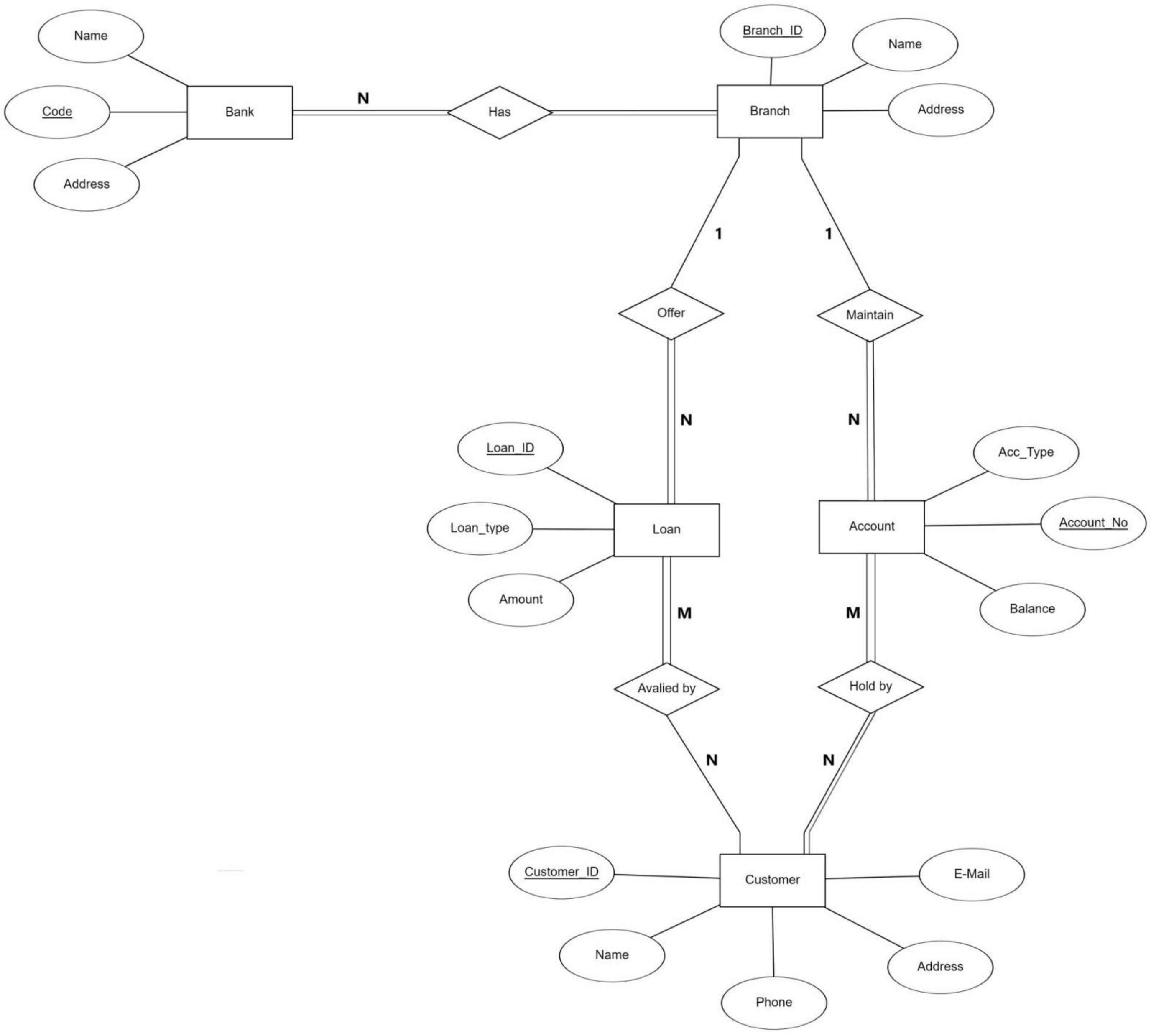
- Have security
- Correct functioning of functions
- Correct storage of data
- Being able to do the desired situations

1.3.4 Software Risk Issues:

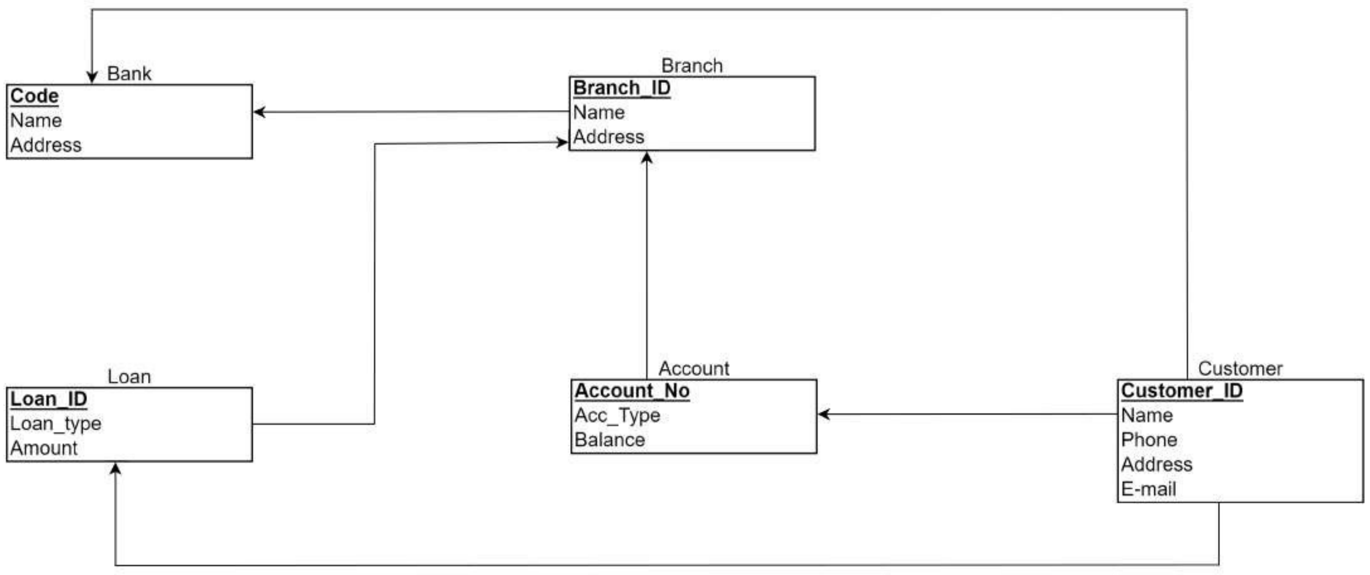
- Complex code
- Logic errors
- Open source code

1.4 Data Models

1.4.1 E-R Diagram



1.4.2 Relational Tables



1.5. Software Processes and Process Models

1.5.1 Selected 5 Software Process Models and Reasons

Incremental Development:

Banking applications often involve complexity and multiple components. The Incremental Development model allows software to be developed incrementally, yielding a usable product at each stage. This facilitates modular development of banking applications and enables receiving customer feedback at each stage.

Iterative Waterfall Model:

The Iterative Waterfall Model is a progressive and iterative version of the Waterfall model. Each phase includes iterations, making it suitable for reducing risks and incorporating customer feedback frequently in the development of banking applications.

Prototyping Model:

Understanding the user interface and functionality accurately is crucial for banking applications. The Prototyping Model allows for the creation of rapid prototypes to gather customer feedback early in the development process, ensuring that the application is user-friendly and meets requirements.

Spiral Model:

The Spiral Model is a risk-focused model. Banking applications often involve complex business processes, making it essential to identify and mitigate potential risks. The Spiral Model facilitates risk analysis at each stage of the project and allows for project adaptation.

Agile Model:

Agile Model may be preferred for banking applications to adapt to rapidly changing requirements. This model emphasizes flexibility, customer-centrality, and rapid development through frequent iterations. Agile supports continuous alignment with customer feedback and swift delivery.

1.5.2 Decided Software Process Model and Reason

Agile Model:

- Flexibility and Adaptability: Agile is known for its ability to quickly adapt to changing requirements and conditions. This facilitates easy adjustment to uncertainties that may arise during the progression of the project.

- Customer-Centric Approach: Agile encourages continuous customer engagement and feedback. This allows for a better understanding of customer expectations and shaping the project according to customer needs.

- Frequent Iterations and Rapid Delivery: Agile includes frequent and small iterations. This allows for regular assessments of the project and the rapid delivery of usable features. Early value delivery and feedback collection become possible.

- Risk Management: Agile involves regularly assessing and managing project risks. This capability helps in identifying and addressing potential issues at earlier stages of the project.

- Test-Driven Development (TDD): Agile adopts Test-Driven Development principles. This ensures that software is developed through testing, aiding in the early detection of potential risks.

2. Requirements

2.1 Functional Requirements

• Create, update, and delete user accounts.

• Define user roles and permissions (customer, administrator, etc.).

• Open new accounts and update existing accounts.

• View and perform transactions on account balances.

• Receipt of the customer's personal information.

• Perform credit transfers, wire transfers, and automatic payments.

• Manage transfer limits and permissions.

• Apply for, approve, or reject credit applications.

• Manage credit account summaries and debt.

2.2 Non-Functional Requirements

2.2.1. In Terms of Security:

• Open to admin access.

• User will have a password and username

• Users must authenticate according to strong password policies.

• User account locked out after consecutive incorrect logins

2.2.2. In Terms of Constraints (System Constraints):

• Assignments should be made(Admin,customer,worker)

2.2.3. In Terms of Usability:

• System will have a login screen

• The system should be available 24/7.

• A user-friendly interface design should be implemented.

• Users should receive alerts before critical transactions.

2.2.4. In Terms of Performance:

• The system should never collapse.

• Speed

• An understandable screen

• Inteface should be easy to use

• The page should load into 3 seconds.

• Money order should be as fast as possible.

• The system should support at least 1000 users simultaneously.

2.2.5. In Terms of Supportability (Maintenance):

• Planning should be done

• Software updates should be easily applied without affecting users.

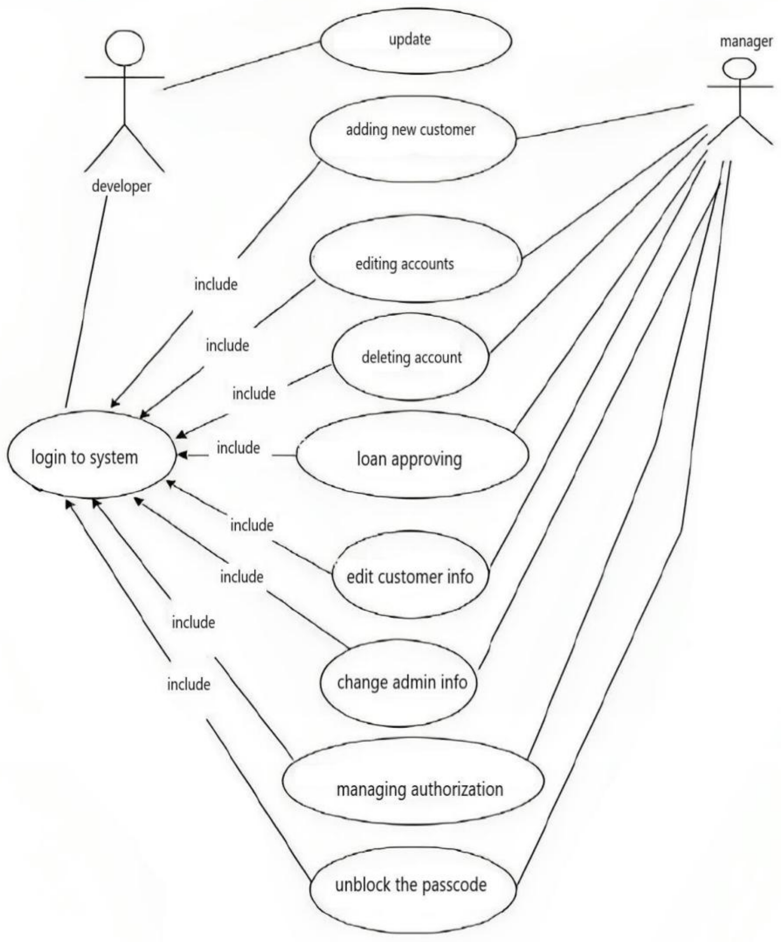
• Maintenance processes should be planned and conducted regularly

• The system should easily expand to adapt to new technologies.

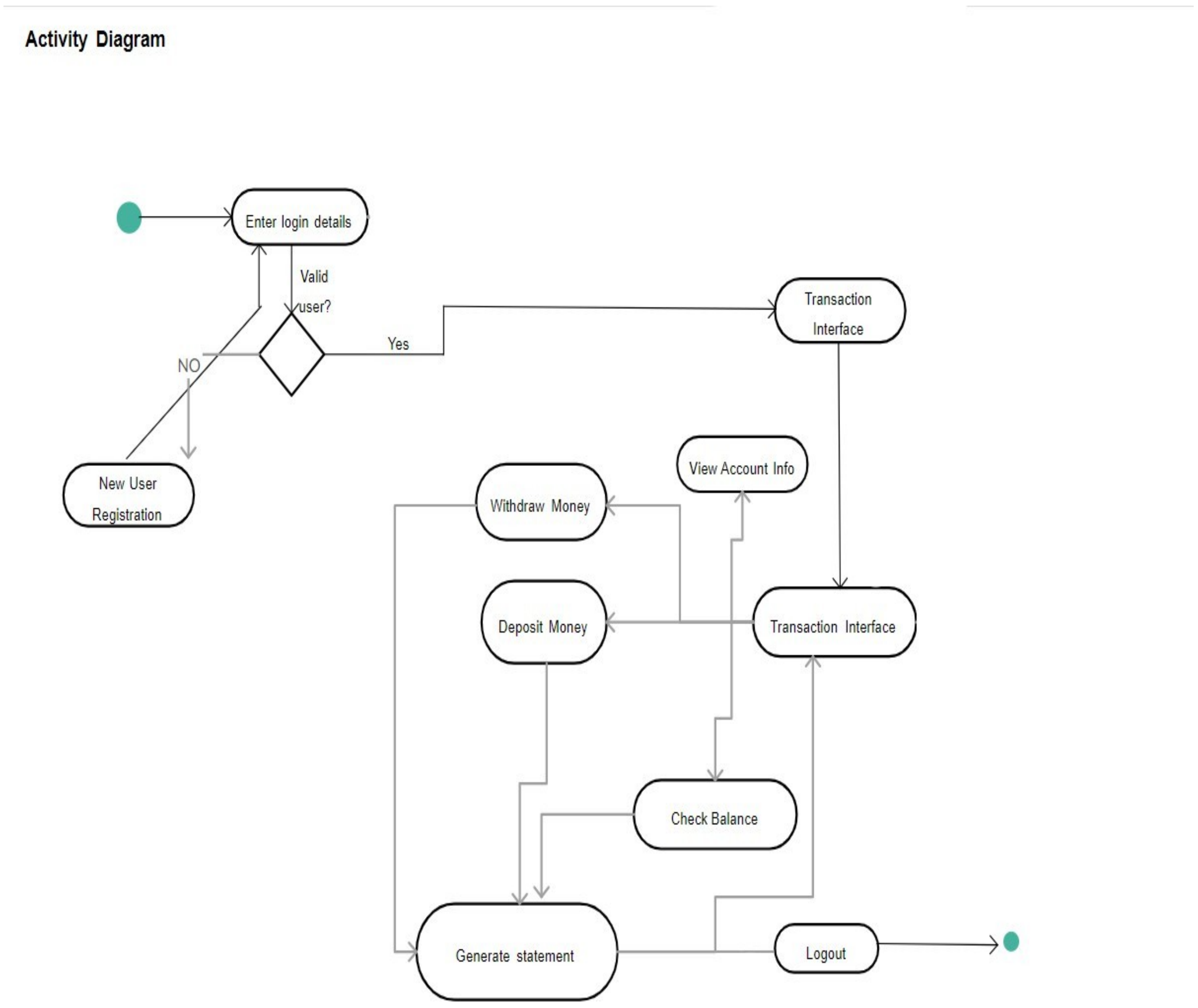
3. System Modelling with UML

3.1 Behavioral Diagrams

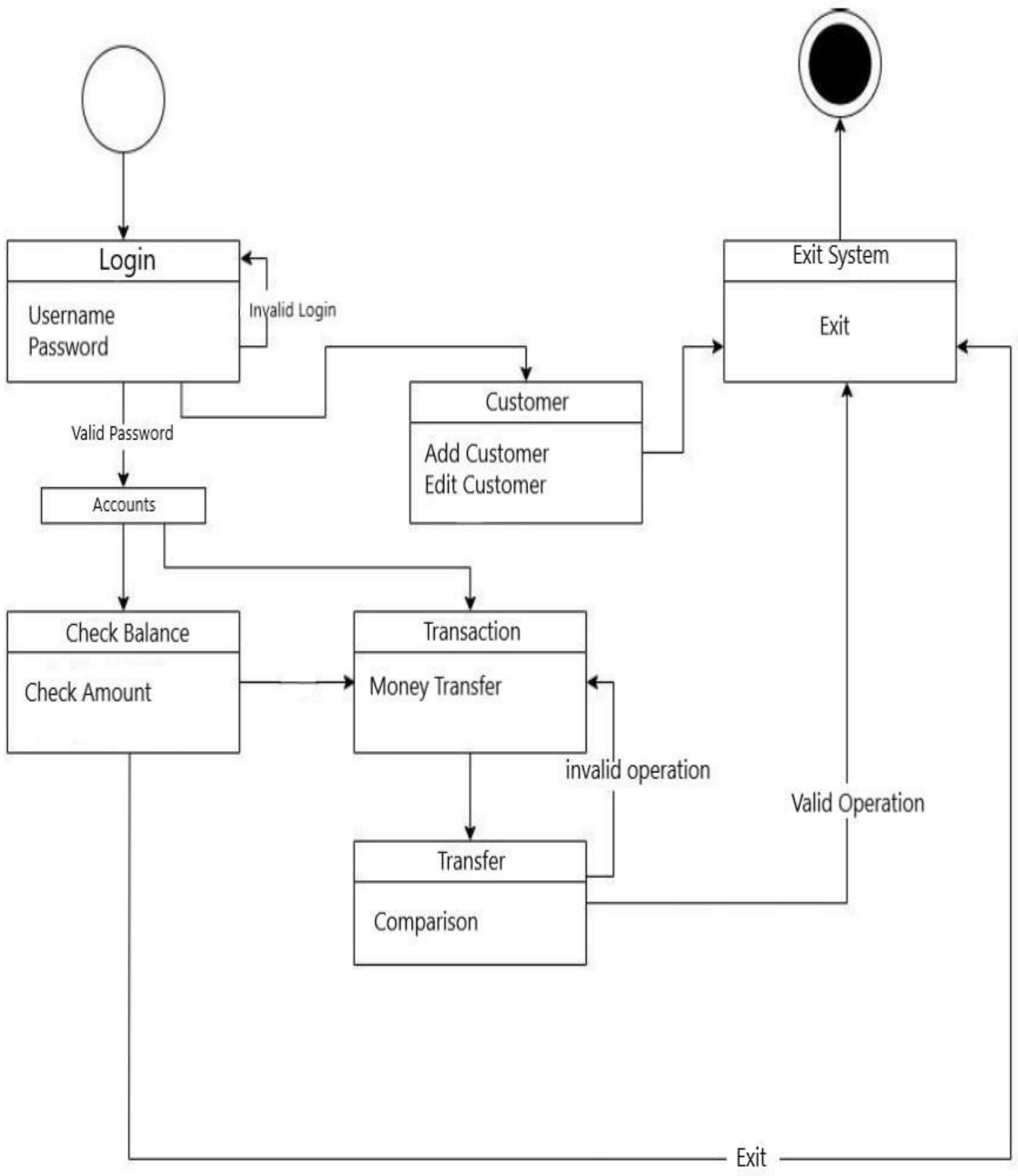
3.1.1 Use-Case Diagram



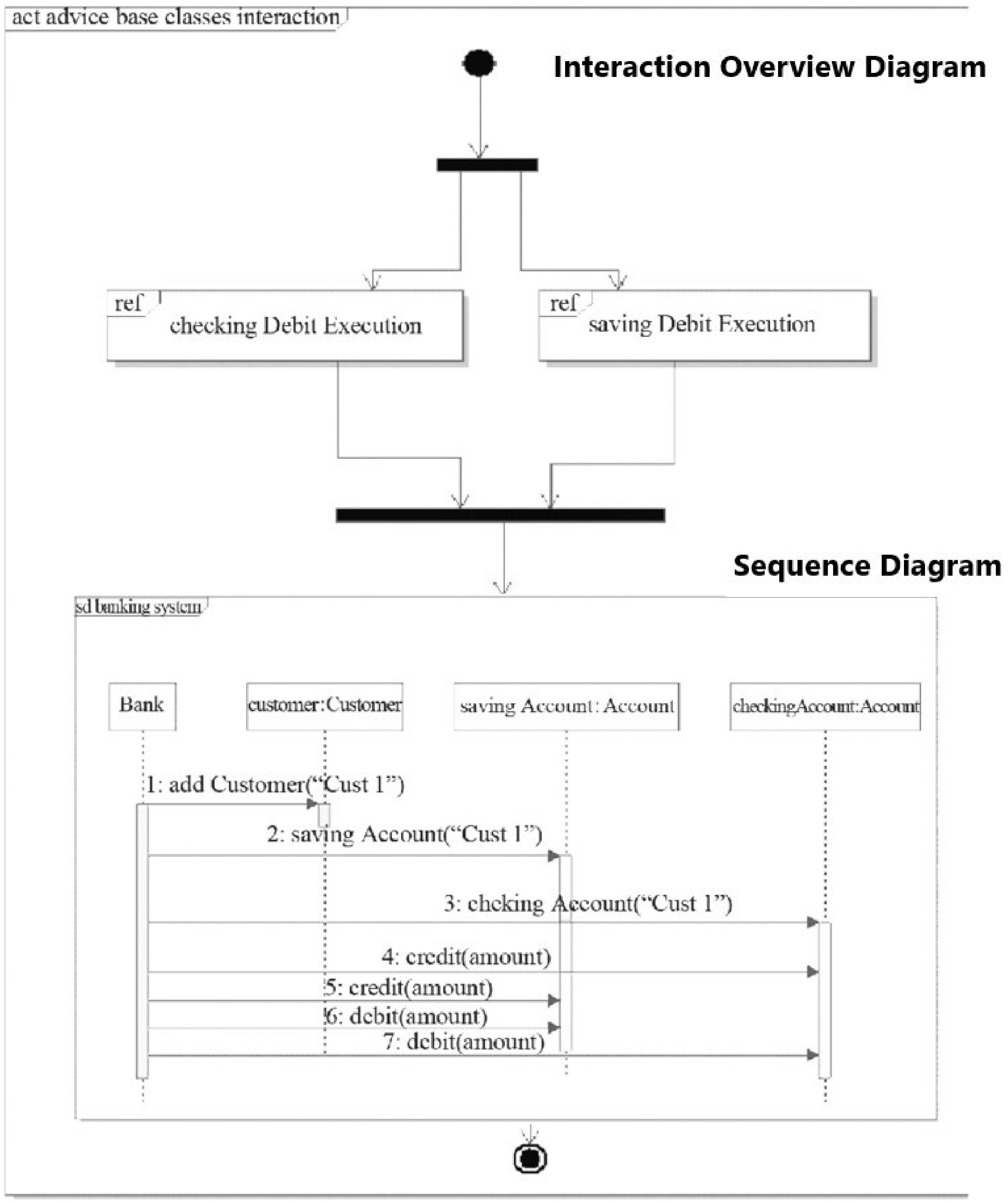
3.1.2 Activity Diagram



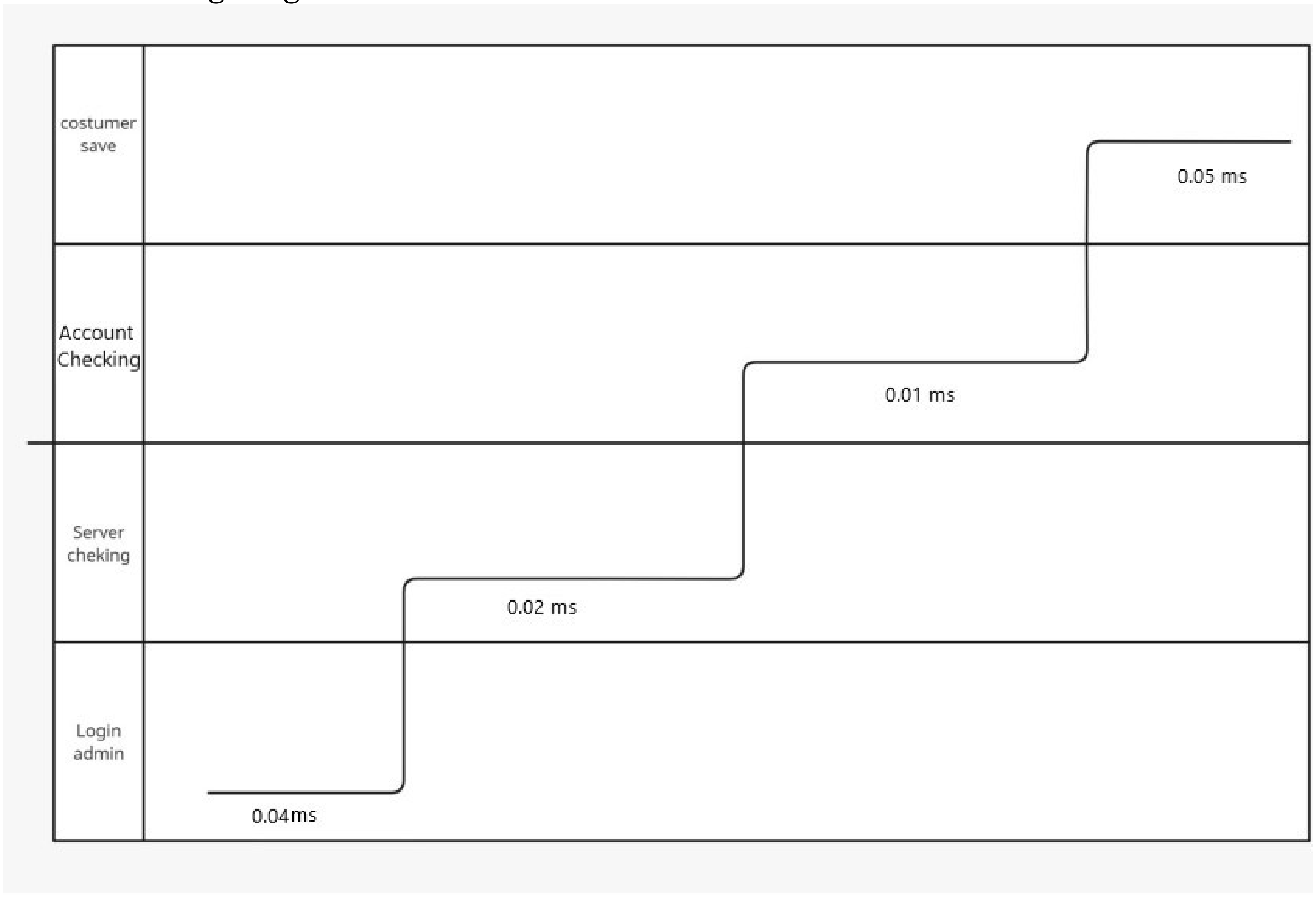
3.1.3 State (State Machine) Diagram



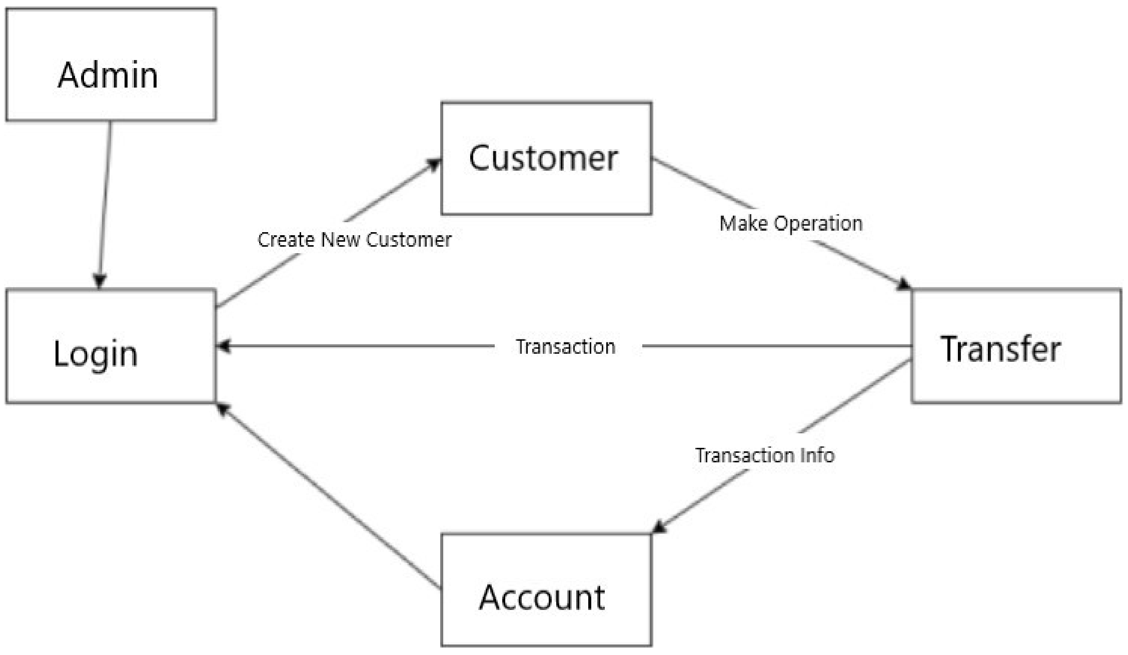
3.1.4 Interaction Diagram (3.1.4.1 - 3.1.4.2)



3.1.4.3 Timing Diagram

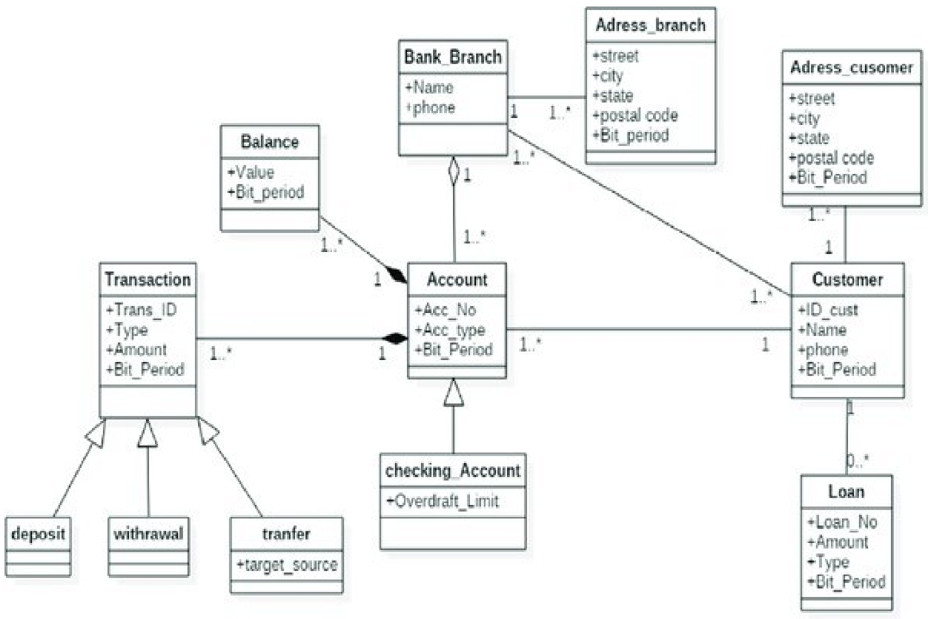


3.1.4.4 Collaboration Diagram (Communication Diagram)

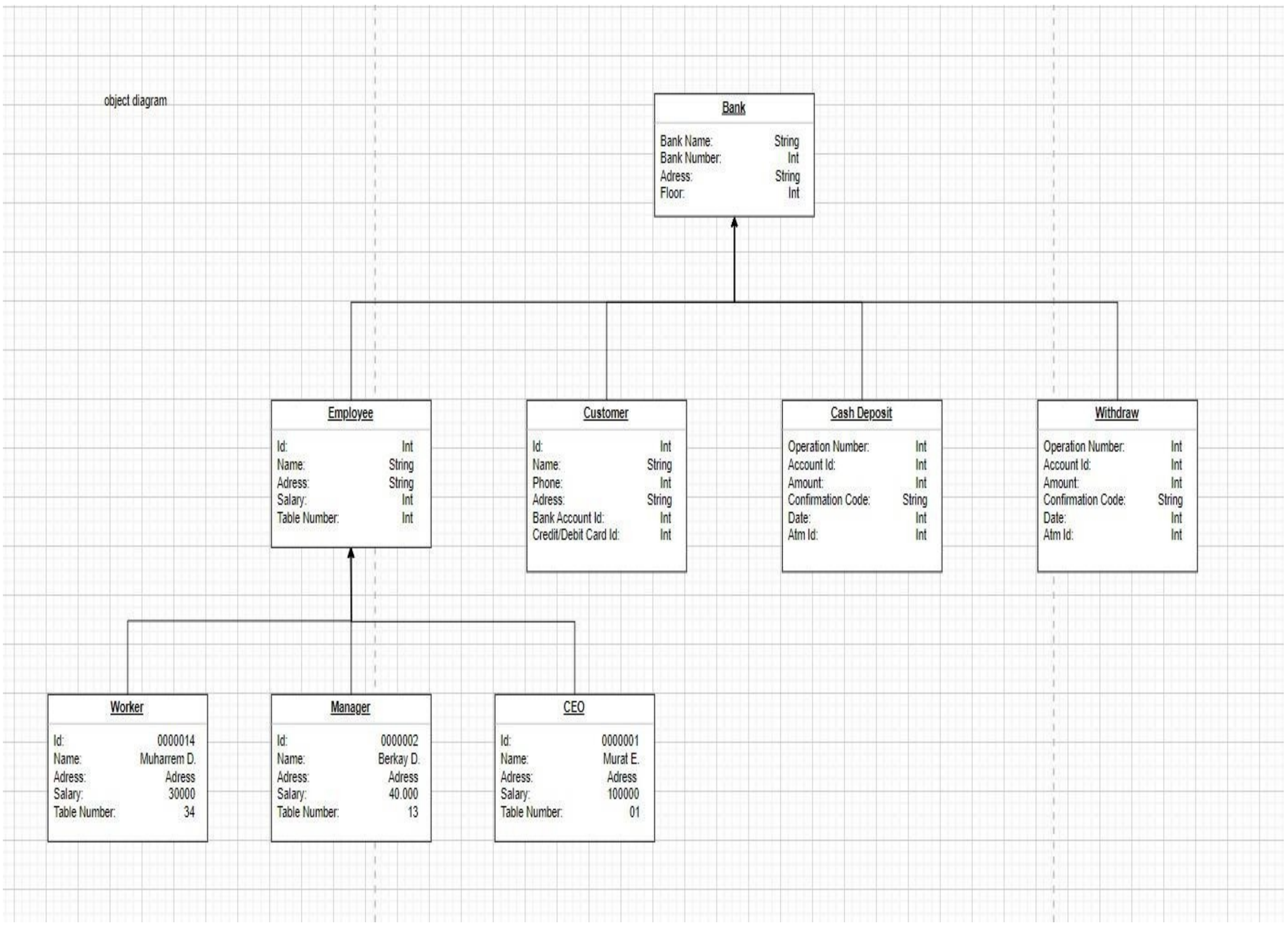


3.2 Structural Diagrams

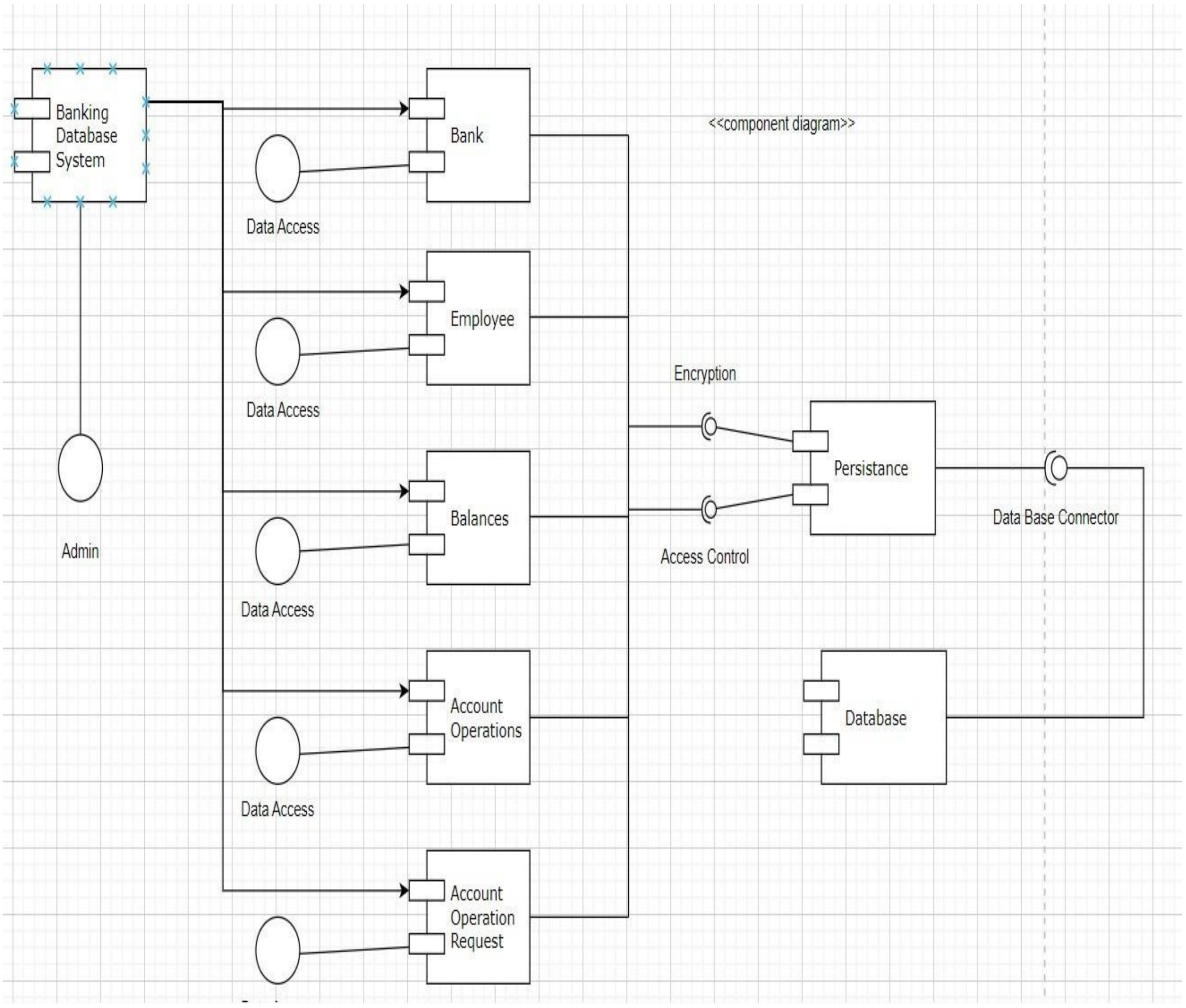
3.2.1 Class Diagram



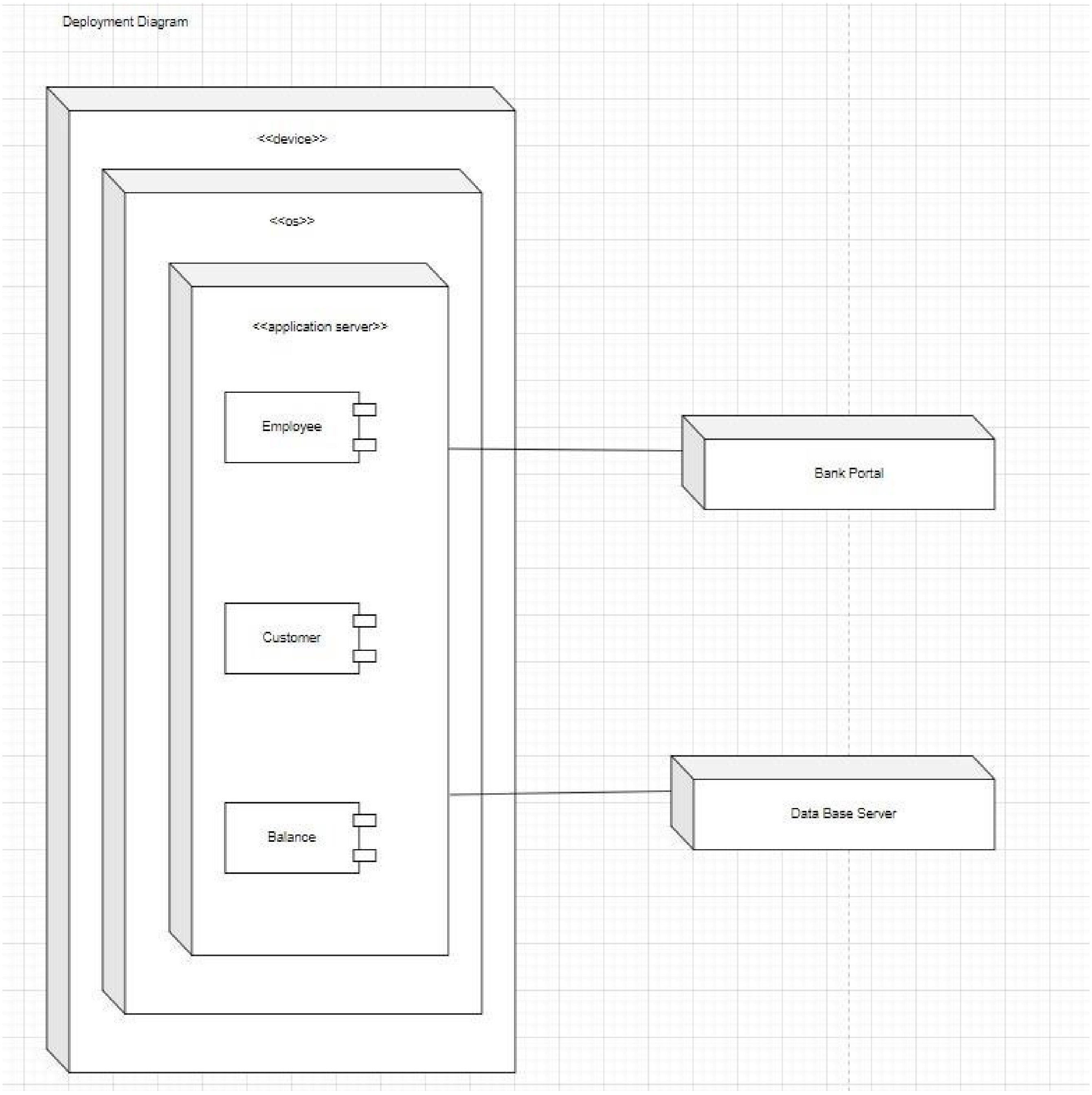
3.2.2 Object Diagram



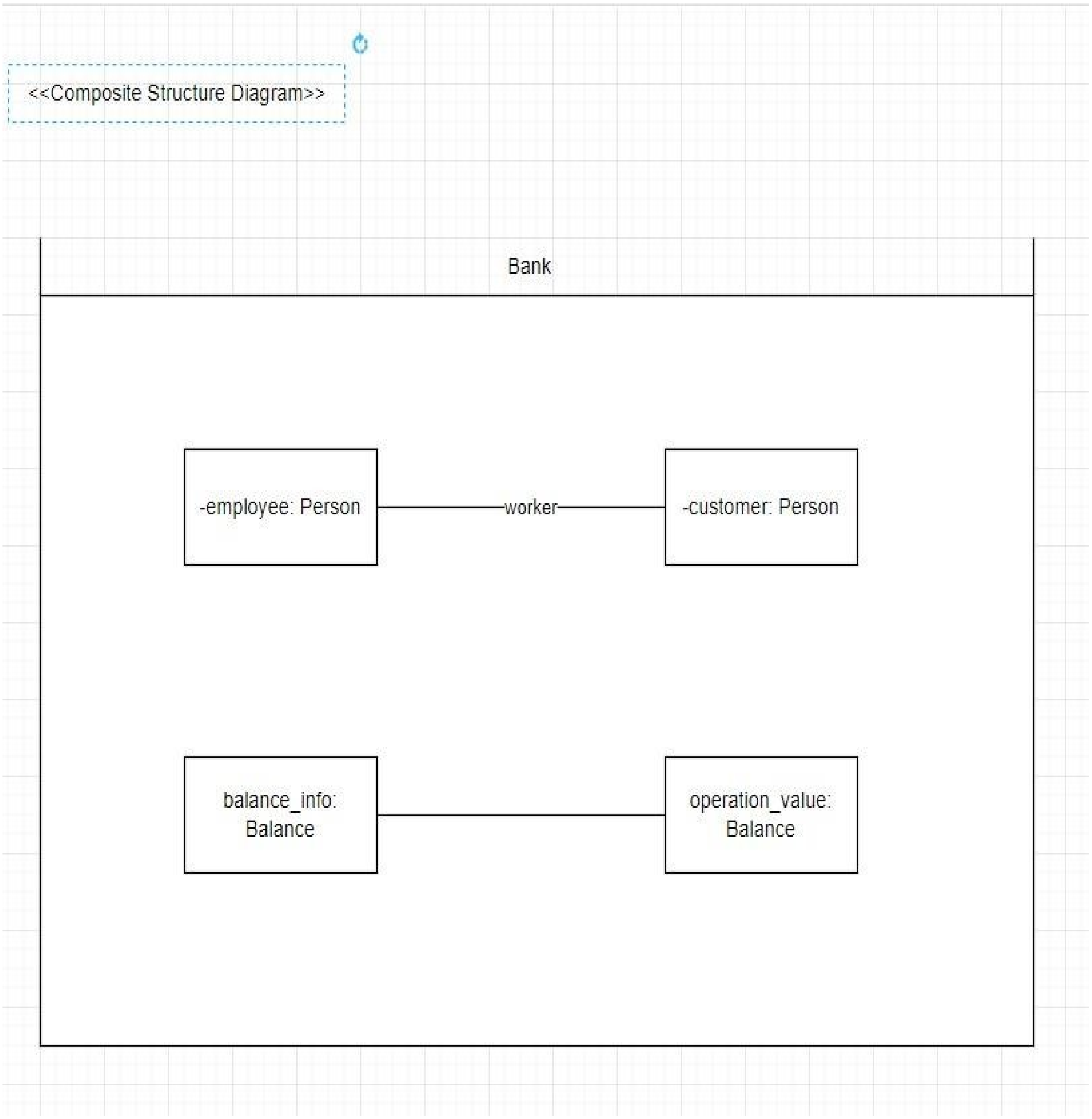
3.2.3 Component Diagram



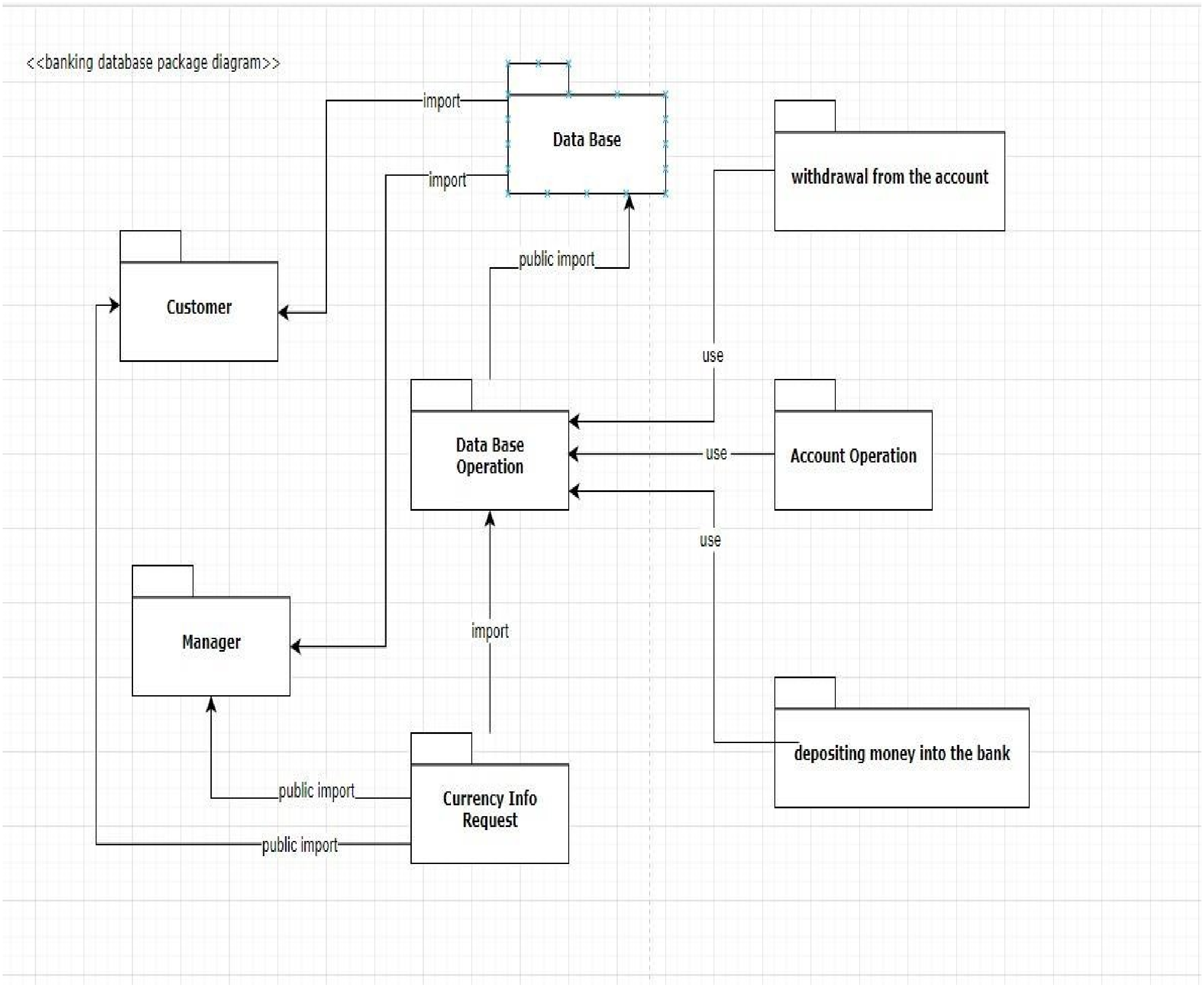
3.2.4 Deployment Diagram



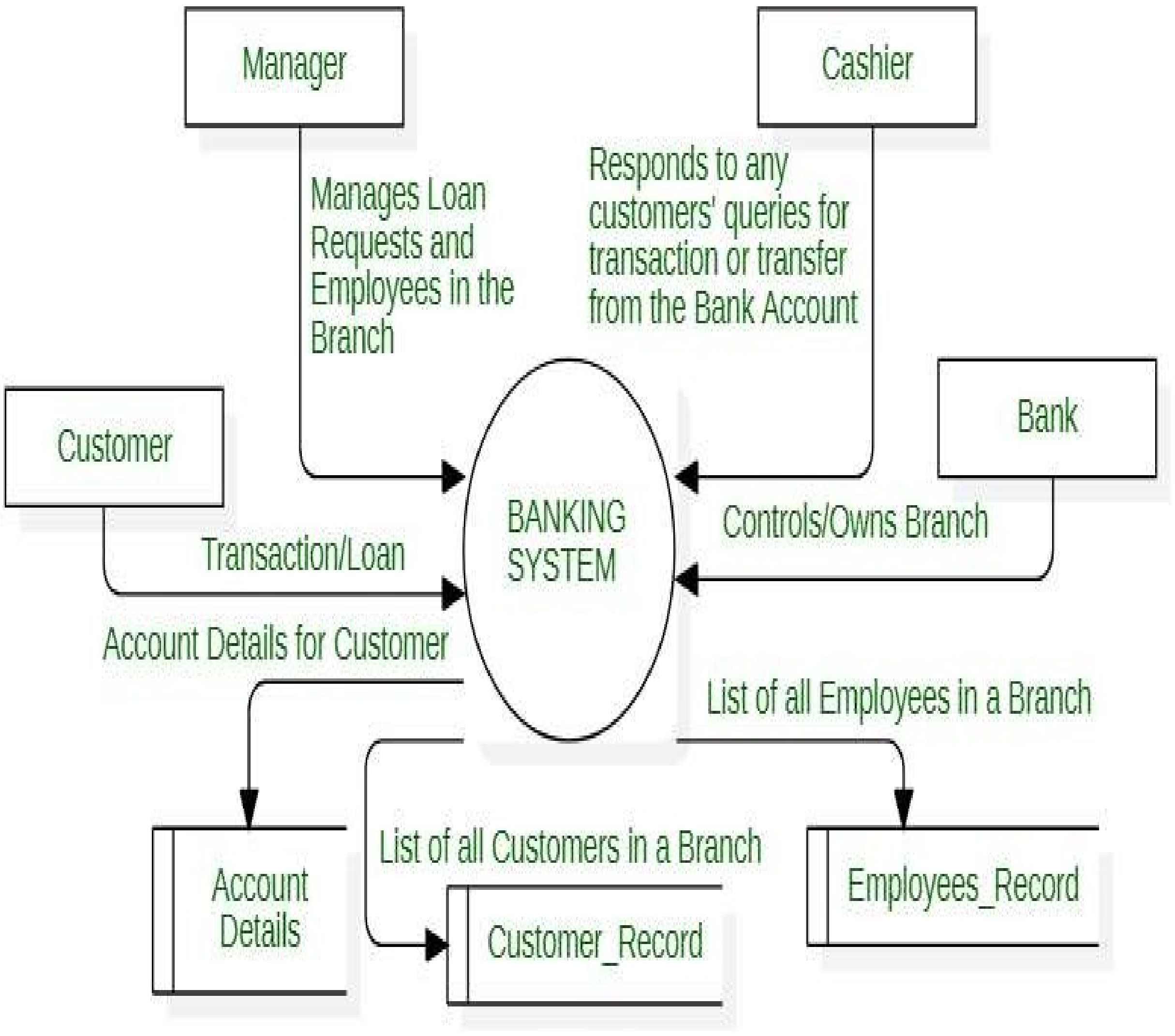
3.2.5 Composite Structure Diagram



3.2.6 Package Diagram



3.2.7 Data-Flow Diagram


Manager
Cashier
Manages Loan Requests and Employees in the Branch
Responds to any customers' queries for transaction or transfer from the Bank Account
Customer
BANKING SYSTEM
Bank
Transaction/Loan
Controls/Owns Branch
Account Details for Customer
List of all Employees in a Branch
List of all Customers in a Branch
Account Details
Customer_Record
Employees_Record

LEVEL - 0

4. Software Testing

4.1 Development Testing

1. Register a user.
2. Check if restrictions are checked when registering a user.
3. Login as a agent.
4. Login as a admin.
5. Check if the user login is successful.
6. Check if the login screen is working properly.
7. Is the app warning you if you didn't choose the role?
8. Change theme.
9. Change admin's password.
10. Reset buton in the main menu.
11. Is the registration successful?
12. Save customer information.
13. Open the app.
14. Can the admin edit the accounts?
15. Can the customer check the account information?
16. Check system uptime.
17. Are there any delays?
18. Log out of the system.
19. Is the system shutting down correctly?
20. Login again and check if there is a problem with the information.
21. Enter wrong username and password.
22. Wrong transaction, wrong destination or amount.
23. Wrong withdraw amount.

4.2 Unit Testing

The customer entered the system. Error messages at the login screen is working as expected. Customer main menu works. Customer can withdraw and deposit money. Customer can check the balance. Customer can transfer money successfully. Error messages at the transactions screen is working as expected. Customer can change theme and password. Customers able to see all the accounts and change the admin password, and that causes security vulnerability, so necessary adjustments has to made.

The system admin entered the system. Admin menu works successfully. Admin can manage accounts. For instance, admin can add, edit or delete a customer. Admin can change the theme and its own password.

Result:

Adjustments has made and security vulnerability has removed, so the system became much safer for the user.

4.2.1 White Box Testing

White box testing is made for testing the software by understanding its internal structure and code. The database operations are performed correctly by examining the internal structure of the bank system.

4.3 Integration Testing

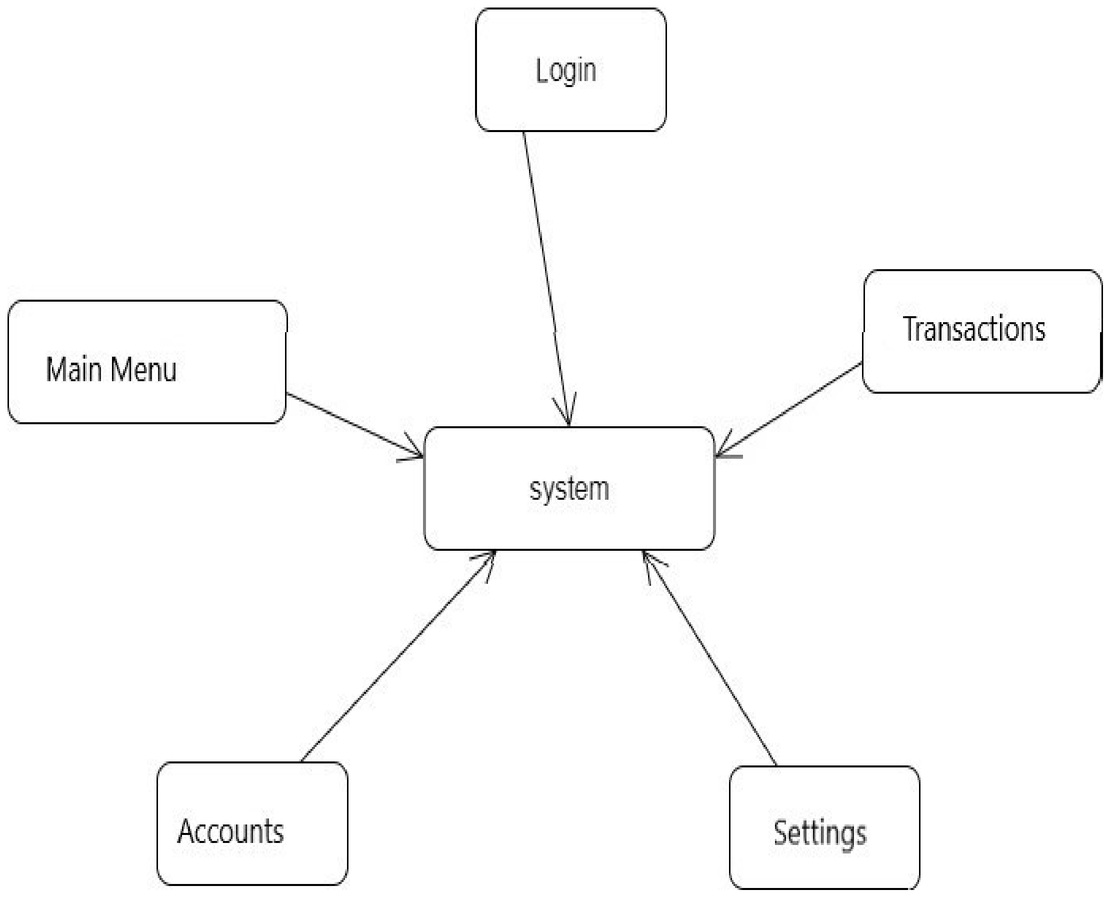
Integration testing is a test method that is used to provide the precision of the software modules, they are consolidated and tested collectively. Although individual methods may pass unit tests successfully, potential errors may arise when integrated into modules. Integration tests play a crucial role in swiftly confirming the seamless operation of the entire software product, allowing us to address potential issues before deployment in the production environment or the introduction of newly developed modules.

Modules:

1. **Login**
2. **Main Menu**
3. **Transactions**
4. **Settings**
5. **Accounts**

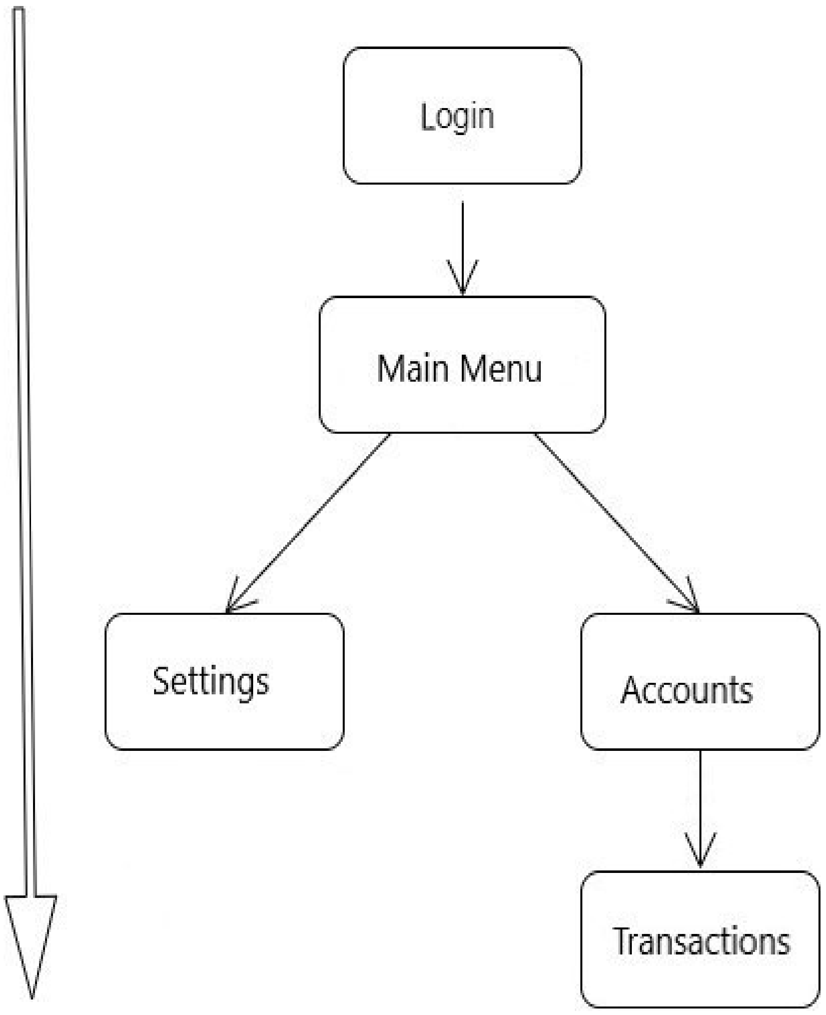
4.3.1 Big-bang

This approach, where all the modules are combined and the functionality is verified after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested.



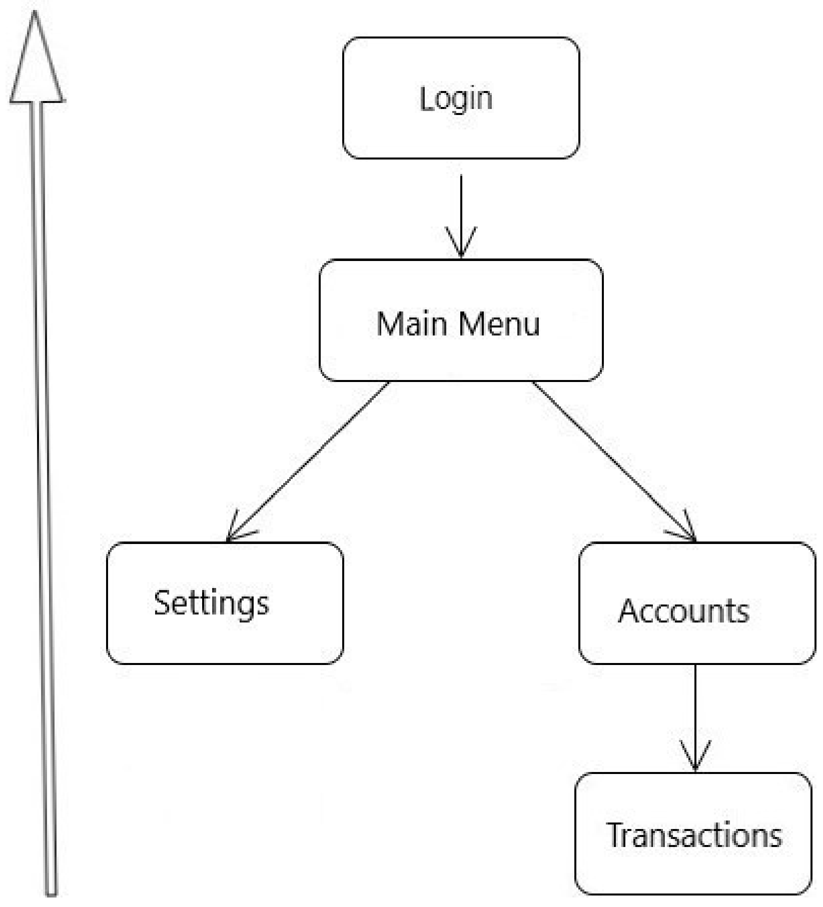
4.3.2 Top-down

In this integration testing, testing takes place from top to bottom.



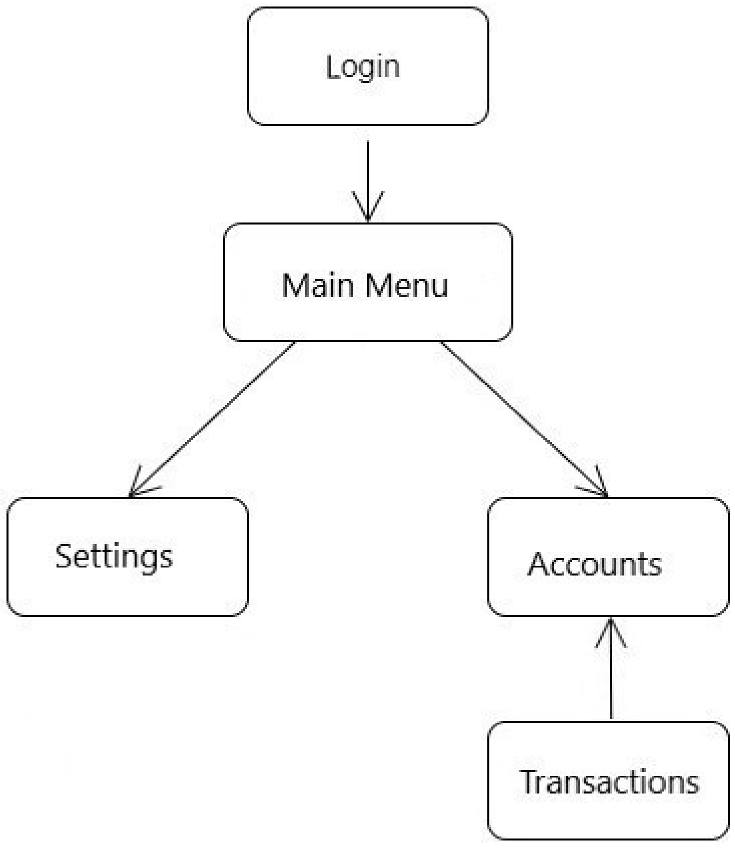
4.3.3 Bottom-up

In bottom-up testing, each module at lower levels are tested with higher modules until all modules are tested.



4.3.4 Hybrid

A mixed integration testing is also called sandwiched or hybrid integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches.



4.4 System Testing

4.4.1 Black Box Testing

Black box testing is made for testing only the inputs and outputs without knowing the internal structure of the software. In this test we verified the outputs if they are as expected. For instance, transferring money is working correctly.

4.5 User Testing/Acceptance Testing

The system tested by the end users in the real conditions. Then, it revised for approval.

4.5.1 Alpha Testing

Alpha Testing serves as a crucial phase in software testing, aiming to uncover and rectify bugs prior to the product's release to actual users or the public. Positioned within the realm of user acceptance testing, it earns its "alpha" designation due to its timing-occurring towards the final stages of software development. Typically conducted by in-house software engineers or quality assurance teams, alpha testing represents the final testing frontier before the software transitions into the live environment. The test was made by the developer team.

4.5.2 Beta Testing

Beta testing is a crucial step in the software development process, involving external users who explore the application and provide feedback before its public release. This phase assesses the application's performance, user interface, and overall reliability in real-world scenarios. Feedback from beta testers helps identify and address potential issues, refine features, and enhance the user experience. Ultimately, beta testing ensures that the bank application is secure, robust, and capable of meeting the diverse needs of its user base. The test was made by a limited group of users.

4.5.3 User Acceptance Testing

User Acceptance Testing (UAT) is a critical step in software development where actual users assess the application to ensure it meets their expectations and business requirements before deployment. This phase confirms the software's functionality, usability, and performance in real-world scenarios. UAT allows stakeholders to identify discrepancies and provide feedback for refinement, serving as a key indicator for the readiness of the software for production release.

4.5.4 Operational Acceptance Testing

Operational Acceptance Testing (OAT) is a vital phase in software testing that assesses a system's ability to operate effectively in its intended environment. Unlike other testing stages, OAT focuses on operational aspects, ensuring the software can handle routine tasks, monitor performance, and recover from failures. This phase is essential for confirming alignment with operational requirements, seamless integration into existing infrastructure, and identifying potential challenges. Successful completion of OAT indicates the software is operationally ready for deployment, assuring its effectiveness in real-world operational scenarios.

4.6 Debugging

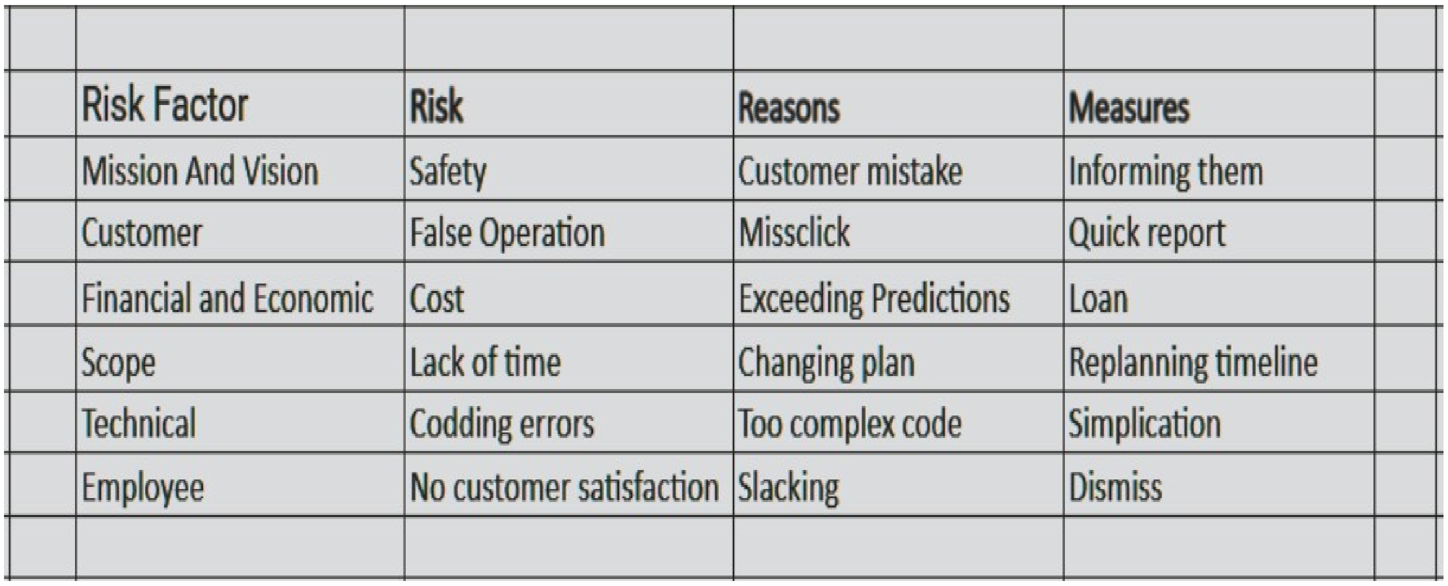
In this phase the system debugged by the development team, then security vulnerabilities are removed. Finally the bank system has became successfully usable and secure for the end users.

5. Identification of Project Risks and Project Risk List

5.1 Identification of Project Risks

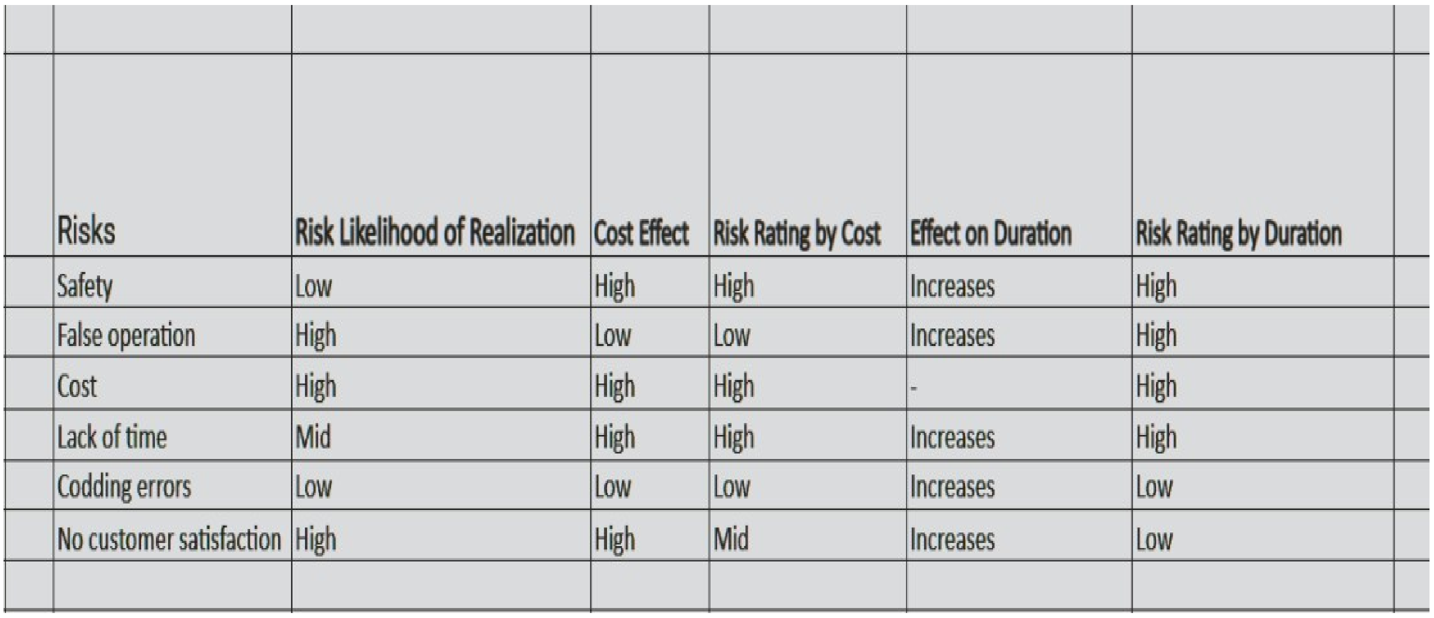
1- Account privacy
2- Currency conversion on app.
3- Changing instantanously the currency and balance values.
4- Finding trustworthy foreign currency databases.

5.2. Project Risk List



Risk Factor	Risk	Reasons	Measures
Mission And Vision	Safety	Customer mistake	Informing them
Customer	False Operation	Missclick	Quick report
Financial and Economic	Cost	Exceeding Predictions	Loan
Scope	Lack of time	Changing plan	Replanning timeline
Technical	Codding errors	Too complex code	Simplication
Employee	No customer satisfaction	Slacking	Dismiss

5.3 Planning Risks and Contingencies



Risks	Risk Likelihood of Realization	Cost Effect	Risk Rating by Cost	Effect on Duration	Risk Rating by Duration
Safety	Low	High	High	Increases	High
False operation	High	Low	Low	Increases	High
Cost	High	High	High	-	High
Lack of time	Mid	High	High	Increases	High
Codding errors	Low	Low	Low	Increases	Low
No customer satisfaction	High	High	Mid	Increases	Low

6.1 Responsibilities of Members

Members generally acted collectively. **Murat** was in constant contact with the customer. **Berkay**, **Muharrem** acted together in the assembly part and the interface design.
Murat, Berkay did the Test parts and mostly **Muharrem** did Coding parts.

Murat :
Class diagram , timing diagram , object diagram ,pakect diagram,deployment diagram develelopment test

Muharrem:
Component diagram, Er diagram, Relation table, Composite structure, Data flow, interaction overview diagram,

Berkay:
Integration testing, state diagram, activity diagram, sequence diagram, white/black box test, unit test

Also, while we developing the project, we considered Murat as a customer. We asked questions and get some feedbacks.

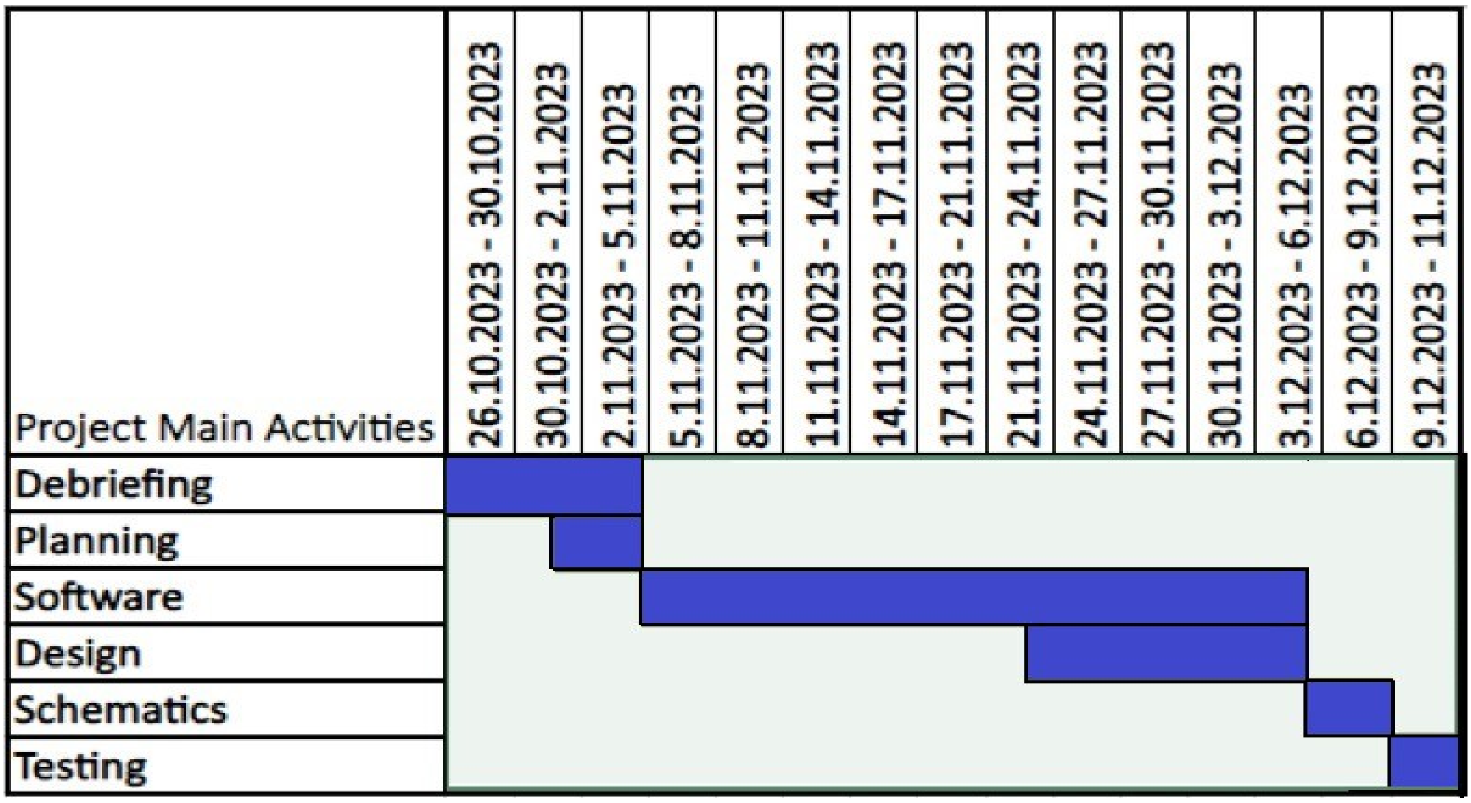
6.2 Delivery Time/Schedule:

First week :
ER diagram, activity diagram , use case diagram, sequence diagram, class diagram and testing
Second week :
Timing diagram, composite structure diagram,collobration diagram, deployment diagram, Relation diagram and testing
Last week :
Component diagram, data flow diagram, state machine diagram, object diagram, testing and delivery

6.3 Schedule of the Project

Project Main Activities	Assigned to	Duration in Days	Starting Date	End Date	Cost (TL)
Debriefing	Muharem-Berkay	7	26.10.2023	3.11.2023	6100
Planning	all 3 together	3	3.11.2023	6.11.2023	5750
Software	all 3 together	33	6.11.2023	8.12.2023	20000
Design	Murat-Muharrem	12	8.12.2023	20.12.2023	580
Schematics	Muharrem-Berkay	3	20.12.2023	23.12.2023	1000
Testing	Murat-Berkay	2	23.12.2023	25.12.2023	12000

Gantt Chart Representation


Project Main Activities
Debriefing
Planning
Software
Design
Schematics
Testing