Tribhuvan University

Institute of Science and Technology

Transaction Processing System for Cooperative Management

Project work

Submitted to

Central Department of Computer Science and Information Technology

Kirtipur, Kathmandu Nepal

In partial fulfillment of the requirements for the Master's Degree in

Computer Science and Information Technology

By

Dinesh khadka

December, 2019

Supervisor

Asst. Prof. Nawaraj Poudel

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

Student's Declaration

I hereby declare that I am the only author of this work and that no sources other than the listed here have been used in this work.

… … … … … … …

Dinesh Khadka

December, 2019

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

Supervisor's Recommendation

I hereby recommend that this project work prepared under my supervision by Mr. Dinesh Khadka entitled "**Transaction processing system for Cooperative management**" be accepted as partial fulfillment of the requirements for the degree of M.Sc. in Computer Science and Information Technology. In my best knowledge this is an original work.

.....

Asst. Prof. Nawaraj Poudel
Head of Department
Central Department of Computer Science and
Information Technology(CDCSIT)
Tribhuvan University
Kritipur
(**Supervisor**)

Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology

LETTER OF APPROVAL

We certify that we have read this project work and in our opinion it is satisfactory in the scope and quality as a project work in the partial fulfillment for the requirement of Masters Degree in Computer Science and Information Technology.

Evaluation Committee

Asst. Prof. Nawaraj Poudel **Head of Department** Central Department of Computer Science and Information Technology(CDCSIT) Tribhuvan University Kritipur	**Asst. Prof. Nawaraj Poudel** **Head of Department** Central Department of Computer Science and Information Technology(CDCSIT) Tribhuvan University Kritipur (**Supervisor**)
(External Examiner)	(Internal Examiner)

ACKNOWLEDGEMENTS

It is a great pleasure for me to acknowledge the contributions of a large number of individuals to this work. First of all, I would like to thank my supervisor **Asst. Prof. Nawaraj Poudel** for giving me an opportunity to work under his supervision and for providing me guidance and support throughout this work.

I would like to express my gratitude to the respected teachers **Dr. Tanka Dhamala**, **Prof. Sudarshan Karanjeet**, **Prof. Dr. Laxmi P. Gewali** (University of Nevada, Las Vegas, USA), **Prof. Dr. Srinath Srinivasa** (IIIT-B), **Hemanta B. G.C.** and all other teachers who have taught us in our Master Degree.

Finally, I am in debt to my friends **Hem Raj Aryal, Achyut Pd. Pathak, Kamal Bista, Dadhi Ghimire** for their fruitful discussions. Last but not least, I would like to thank my family members for their constant support and encouragement.

Dinesh Khadka

ABSTRACT

Transaction processing system for cooperative management is an application that coordinates the activities and transactions involved in running and working of cooperative organization. The system simplify the transaction including the attributes such deposit, withdraw, interests, loan etc. making it simple for user to work using IT technology. The system helps medium and small co-operatives around Nepal to perform their transaction easily and quickly.

Table of content

List of figure

List of table

List of abbreviations

CSS	:	Cascading Style Sheet
CI-CD	:	Continuous Integrity – Continuous Deployment
CRUD	:	Create Read Update and Delete
ERD	:	Entity Relationship Diagram
HTML	:	Hyper Text Mark-up Language
HTTP	:	Hyper Text Transfer Protocol
ICT	:	Information Communication and Technology
IDE	:	Integrated Development Environment
IT	:	Information Technology
SQL	:	Structured Query Language
UI	:	User Interface
XML	:	eXtensible Markup Language

Chapter 1
Introduction

1.1 Introduction

Transaction processing system for cooperative management is an application that coordinates and integrates all the activities and transactions involved in running and working of cooperative organization. The proposed system will be developed in order to simplify the transaction and working process of cooperative.

This system provides users with the facility of performing every transaction of work involved in a cooperative system. The main idea behind developing this project is to simplify the transaction including the attributes such as credit, debit, deposit, client, transaction, employees, interests, loan etc. making it secure and simple for user to work using IT technology. This system not only helps to insure the easy working of entire organizational work but also helps to ensure the secure and reliable work done.

The employees on cooperative organization can use the application so as to perform the related banking task avoiding the paper work and helping employees to have their work done in minimal time as possible. The system not only allow user to do their input and their work-related task but also keep the record of all the transaction in a secured database.

This system can also be used for calculating interests to their loans provided. It takes all the records of account associated to their respective field of attributes. This application keeps the records of the related account work such as loans, interests and required calculation needed.

1.2. Problem Statements

In the context of Nepal, there is very few digitization in cooperative system and are more based on paper work system. Keeping that in mind this system is developed in order to bring new way of technological system on cooperative field. There were many problems including loss of paper works, and were vulnerable to natural calamities and also the storing of paper work system is impossible at times in a proper way. So in order to make the transactions and work flow of cooperative system using IT technology and making it usable through internet to simplify the work. Some problems in traditional system are:

- Difficult in accessing and searching specific account and their details.

- Traditional paper work methodology
- Security issues
- Loss and theft of data
- Loss of data due to natural calamities

1.3. Objectives

The main objectives of this project work are:

- To automates the transaction processing of cooperative system.
- To provides simple interface for carrying out transactions easily and quickly.

1.4 Scope

There are thousands of medium and small co-operatives around Nepal that rely on traditional paper based financial record management and simple spreadsheet based financial management. This software package will be very handy to this organization. This is robust and handles all the financial need of a cooperative. It can help to automate the business process of those organizations and help to carter financial services to their customers efficiently and overall helps to modernize their service and offerings.

1.5 Report Organization

This report is separated into different chapters for proper readability and organization. The second chapter consists of system analysis which further defines the requirement collection process, all necessary system requirements including both functional and non-functional requirements which are shown using the use case diagram of the system and feasibility study of the application in order to conduct analysis of how different factors can affect the development of a project. In addition, this chapter also includes the data model of the system(ER Diagram) showing the relationships of entity sets stored in a database. Chapter three consists of system design which includes overall system architecture, schema diagram etc. Chapter four is Implementation and Testing which includes development methodology, tools used, test cases etc. Final chapter consists of concluding remarks about the overall project.

Chapter 2
Requirement Analysis

2. System Requirements

The software system requirements are of two types:

1. Functional requirements
2. Non-functional requirements

2.1 Functional Requirements

The functional requirement documents the operations and activities that a system must be able to perform. For cooperative transaction management system, the functional requirements are as follows:

- Cashier should be able to perform daily financial transactions (deposit, withdrawal, transfer etc).
- Loan officer should be able to design loan products, set criteria and approve loans.
- Manager should be able to design products (Fixed deposit, savings, and currents).
- Each user of the system should be able to log in and log out from the system.
- Each user of the system should be able to update his or her profile change password etc.
- Accountants should be able to manage financial transactions and generate financial report.
- Clerk should be able to register new customer and maintain customer records.

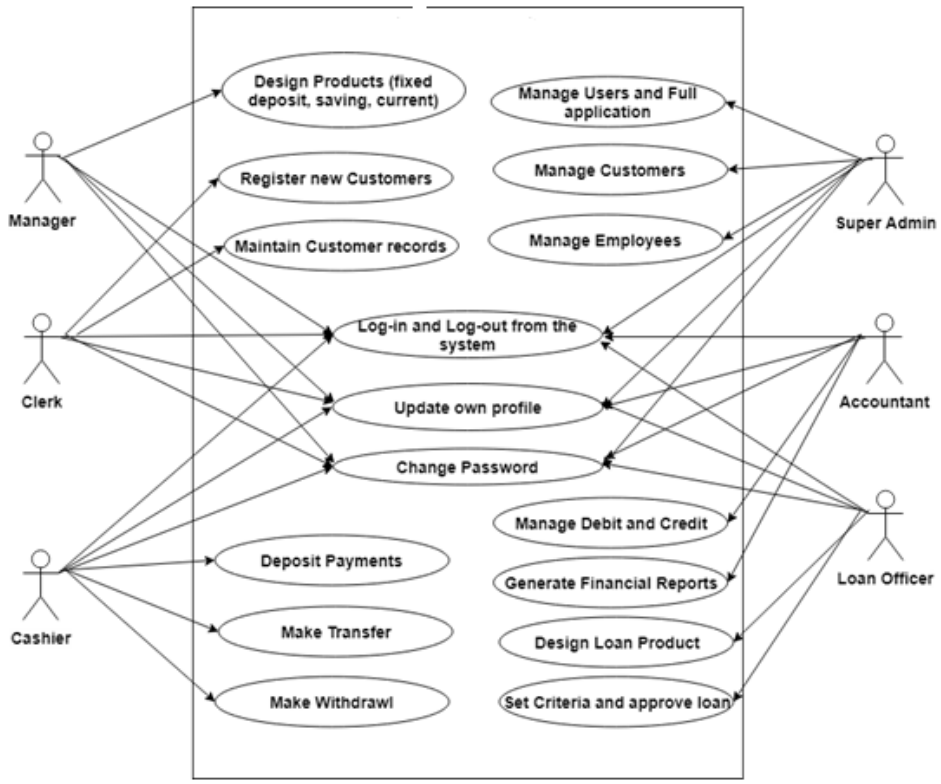


Figure 2. 1: Use case diagram of Transaction processing system

The above use case diagram shows the different actors involved in the system in their roles. There are six actors, they are: Manager. Super admin, clerk, cashier, accountant, loan officer.

All the users login and out of the system and can update their own profile. The manager designs the products, the clerk registers new customers and maintains customer records. Super admin manages the user sand the entire application. Loan officers manages loan related tasks. The cashier payments, makes transfer, makes withdrawal.

2.1.1 Identification of Actors

An actor is the component that interacts with the system or has some role in the system. The actors interacting with the system are as follows:

1. Manager

2. Clerk
3. Super Admin
4. Cashier
5. Loan officer
6. Accountant

2.1.2 Modular Decomposition of System

The whole system in modularly decomposed into standalone modules that can seamlessly integrated into other modules. The major modules of this system are:

- **Loan Product Module**
- **Withdraw Module**
- **Deposit Module**

2.1.3 Use-case Diagram of loan product module

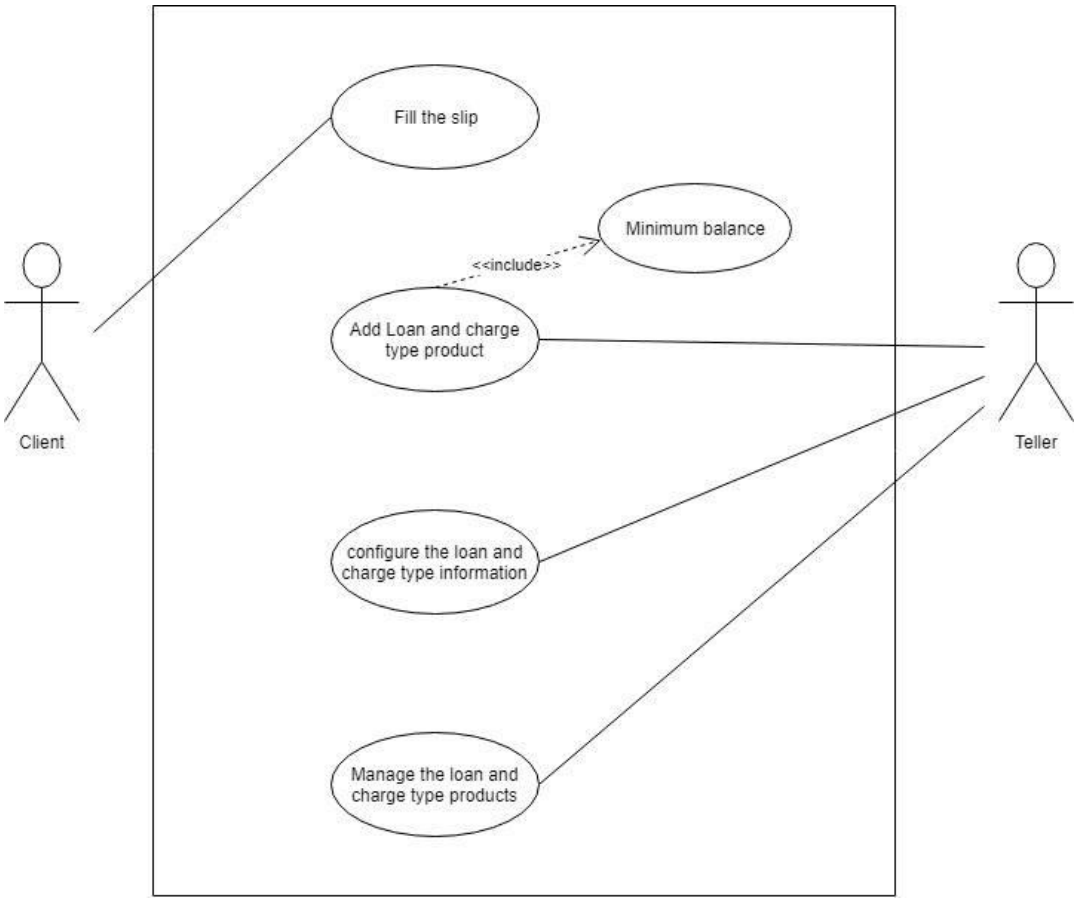


Figure 2. 2: Use case diagram of Loan product module

Client has to fill the form/slip along with official documents for the loan request. The loan request goes through approval process. If the loan is approved the loan detail is added to the clients account. The teller gives loan amount to the customer after the loan is approved. The teller is also responsible for changing product once loan is approved.

2.1.4 Use-case Diagram of Deposit module

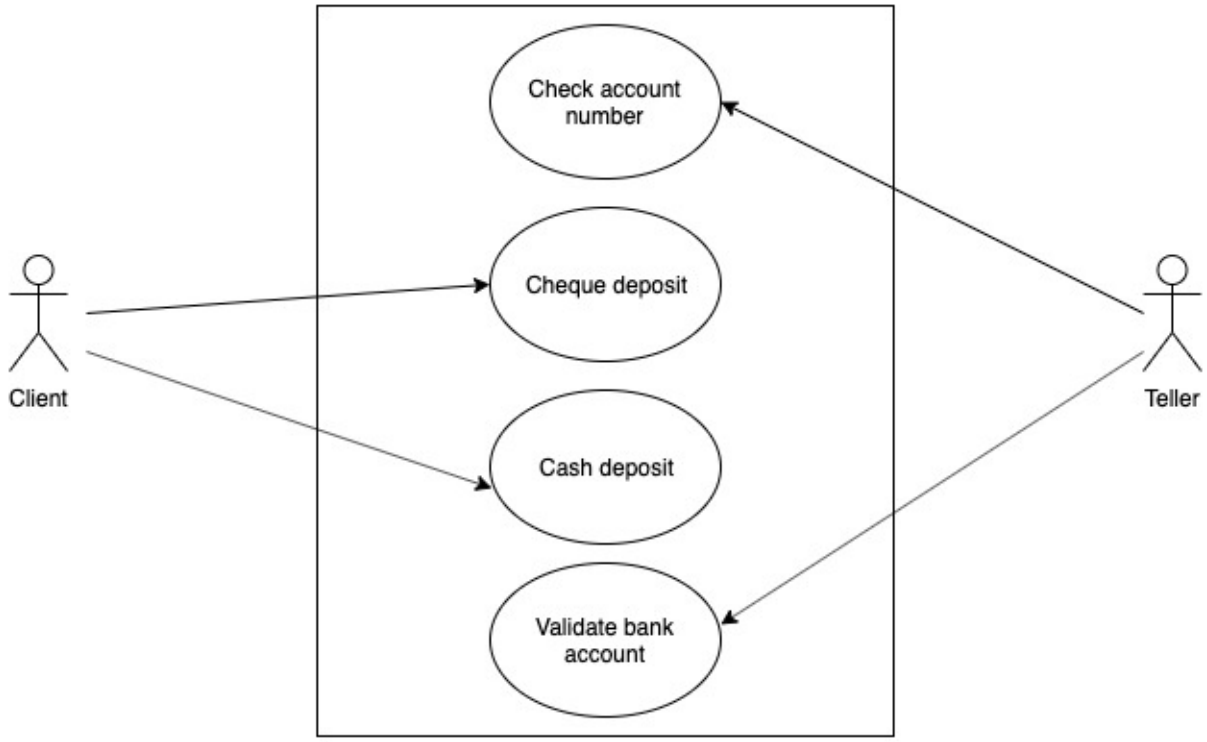


Figure 2. 3: Use case diagram of Deposit module

Clients submits voucher to the teller for the cash or cheque deposits. Teller checks the account and validates the bank account. The corresponding amount is submitted to the account after the validation process.

2.1.5 Use-case Diagram of withdraw module

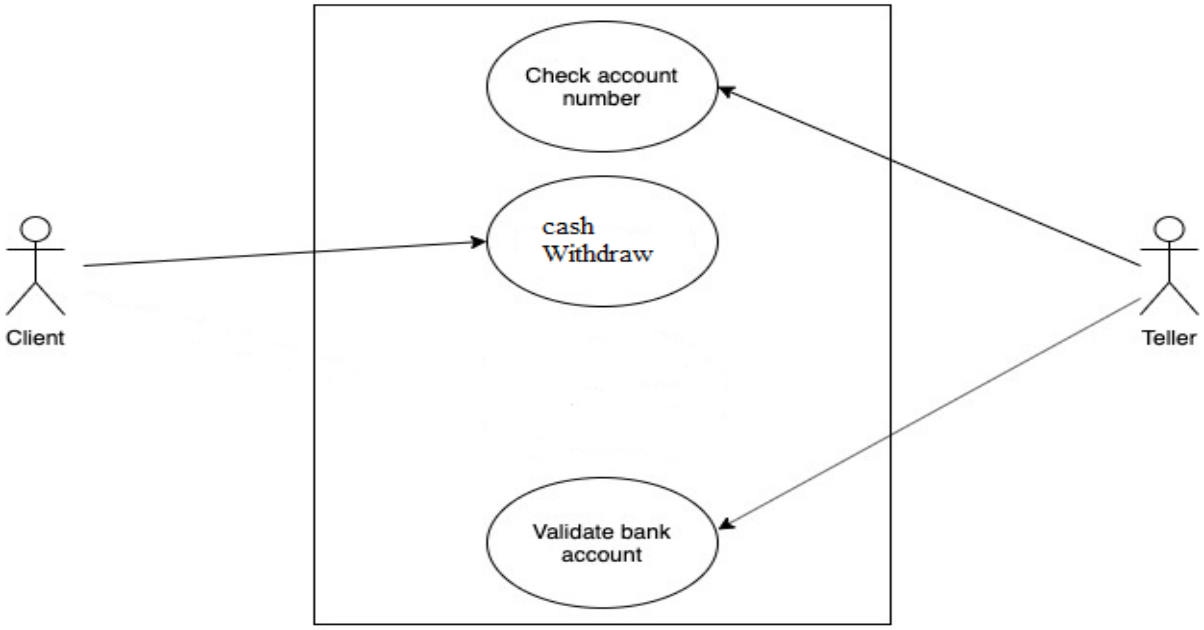


Figure2. 4: Use case diagram of withdraw module

Clients submit cheque or withdraw slip to the teller for the cash withdraw process. Teller checks the account and validates the bank account. The corresponding amount is given to the customer after the validation process.

2.2. Non-functional Requirements

Nonfunctional requirements are requirements that are not directly concerned with the specific delivered by the system to its users. They may relate to emergent system properties such as reliability, response time and store occupancy. The non-functional requirements may come from required characteristics of the software (product requirements), the organizational developing the software (organizational requirements), or from external sources.

1. **Product requirements**

It specify the behavior of the system. It includes how fast the system must execute and how much memory it requires, reliability requirements that set out the acceptable failure rate, security requirements, and usability requirements.

2. **Organizational requirements**

They are broad system requirements derived from policies and procedures. It includes operational process requirements, the development process requirements, the development environment or process standards to be used, and environmental requirements.

3. **External requirements**

They cover all requirements that are derived from factors external to the system and its development process. These may include regulatory requirements, legislative requirements and ethical requirements.

2.3 Feasibility Assessment

The main objective of feasibility study is to test the Technical, Operational, Economic and Schedule feasibility. All systems are feasible only if they are given unlimited resources and infinite time. It helps to determine the benefits of the proposed system in the society and organization. It also determines if the system can be built successfully with cost, time and effort.

Technical Feasibility

The technical feasibility assessment is focused on gaining an understanding of the present technical resources available and their applicability to the expected need of the proposed system. All the necessary technology such as SpringBoot, MySQL, etc. are already available. And also, other resources like Laptops, internet, etc. are available.

Operational Feasibility

In this system, all the features will be implemented using its own databases and through API. And it is compatible for all devices. Therefore, this system will meet the organization's operating requirements.

Economic Feasibility

The technologies and resources needed to build the software, is already available. Users only need internet facility to access this software. So, this software is economical and can serve user's purpose.

2.4. System Modeling

The diagrams used to design and model a system are as follows:

1. Sequence Diagram
2. Activity Diagram

2.4.1. Modeling of loan module

UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of collaboration.

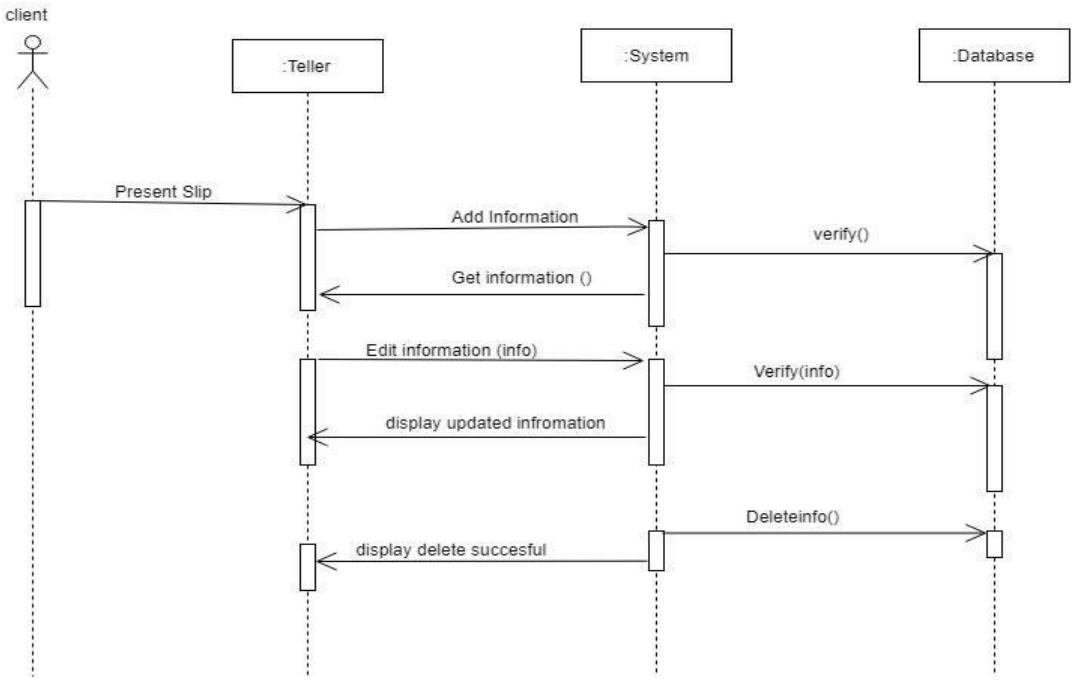


Figure2. 5: Sequence diagram of loan product module

The above sequence diagram shows the interaction between the different objects of the loan product module of the Transaction processing system. The client provides information required for passing the loan, the system verifies it and also updates and deletes the information and the teller provides the loan if everything is validated.

Activity Diagram

An activity diagram visually represents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram. The control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent.

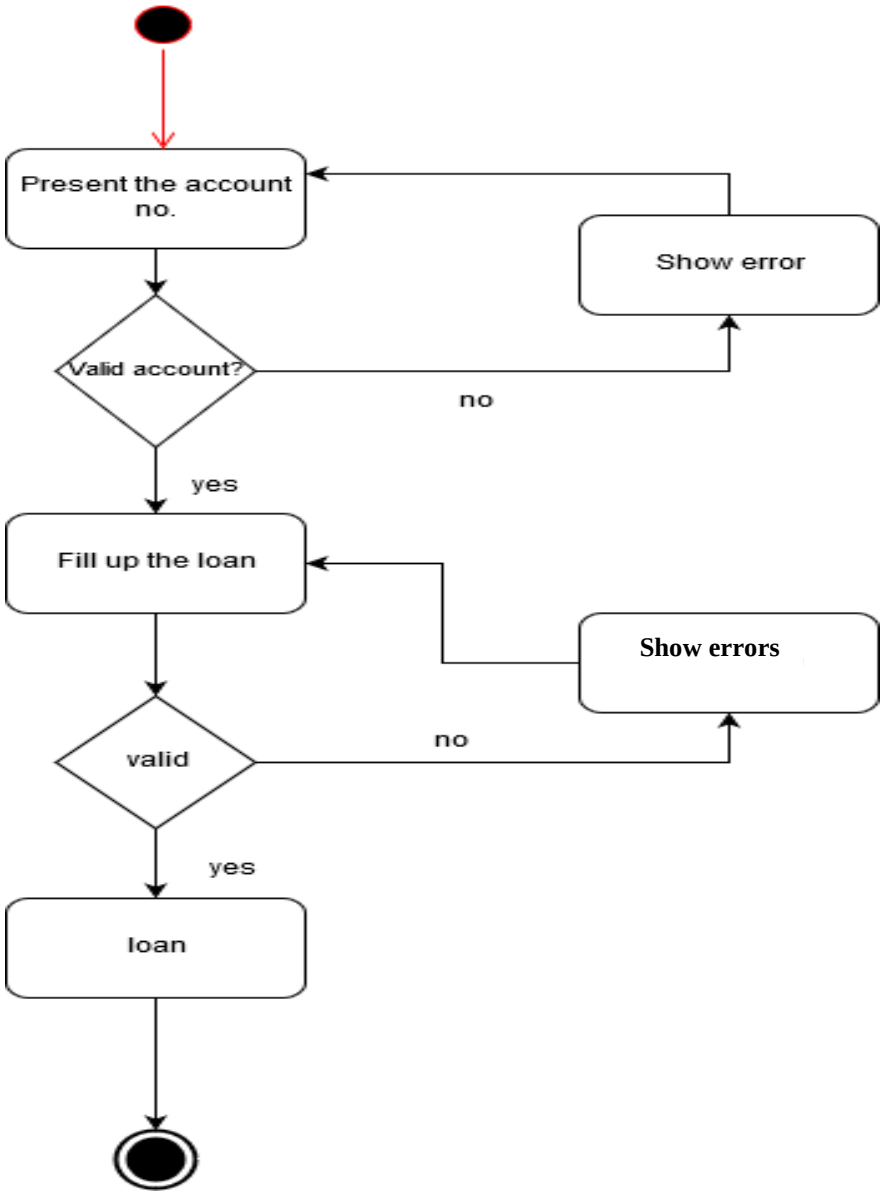


Figure2. 6: Activity Diagram of loan product module of Transaction processing system

The above activity diagram shows the flow of the different activities performed in the loan product module. The client's account is validated, then if the validation is positive, they can fill up the loan form and if the loan form is successfully validated and filled, the loan phase is successful, else an error message is shown.

2.4.2. Modeling of Withdraw module

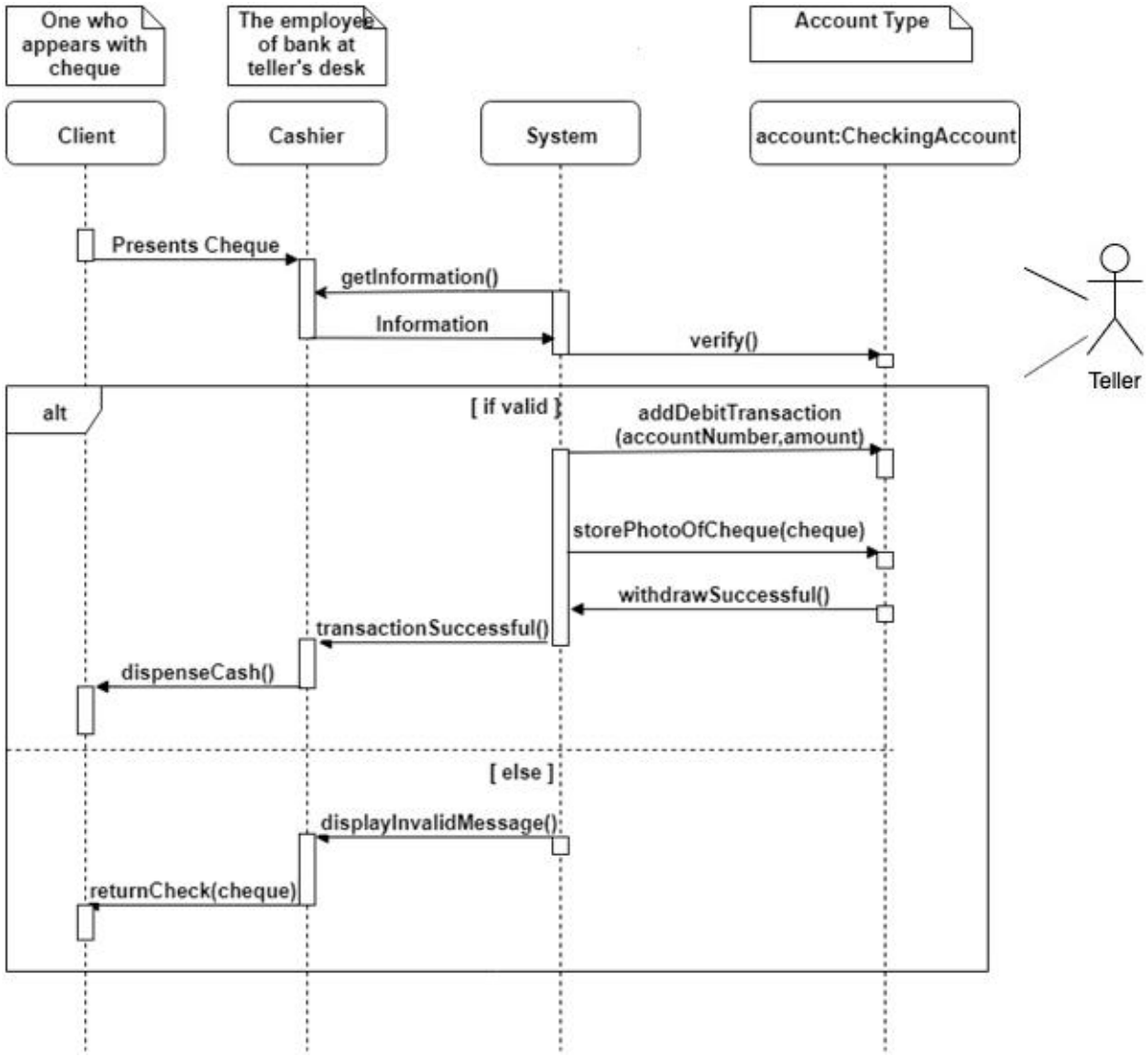


Figure2. 7: Sequence diagram for Withdraw Module

Above sequence diagram shows how withdraw of cash takes place. Firstly, the client presents a cheque to the cashier or teller. The cashier then provides cheque information to the system. The system then verifies the cheque information and define whether it is acceptable or not acceptable. If the cheque and every detail in it is valid, the system performs debit transaction to the account and stores the photo of cheque for future reference. The success message is displayed to cashier and then the cashier dispenses cash to the client. In case of invalid cheque, the system displays invalid message to the cashier and then the cashier marks the cheque as bounced and return to client.

2.4.3. Modeling of deposit module

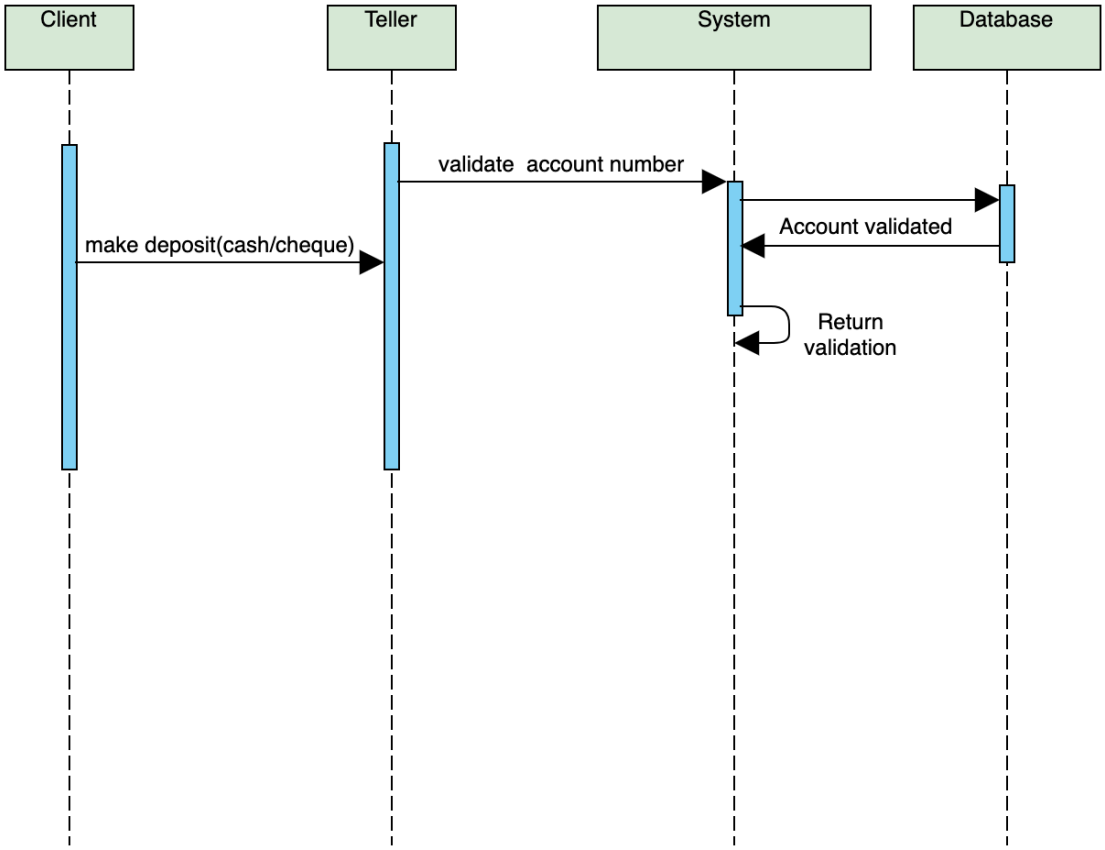


Figure2. 8: Sequence diagram for Deposit Module

Above sequence diagram shows how Deposit of cash/cheque takes place. Firstly, the client presents a cash/ cheque deposit voucher to the cashier or teller. The cashier then provides

account information to the system. The system then verifies the information and shows whether it is acceptable or not acceptable. Finally the deposit is made.

Activity Diagram

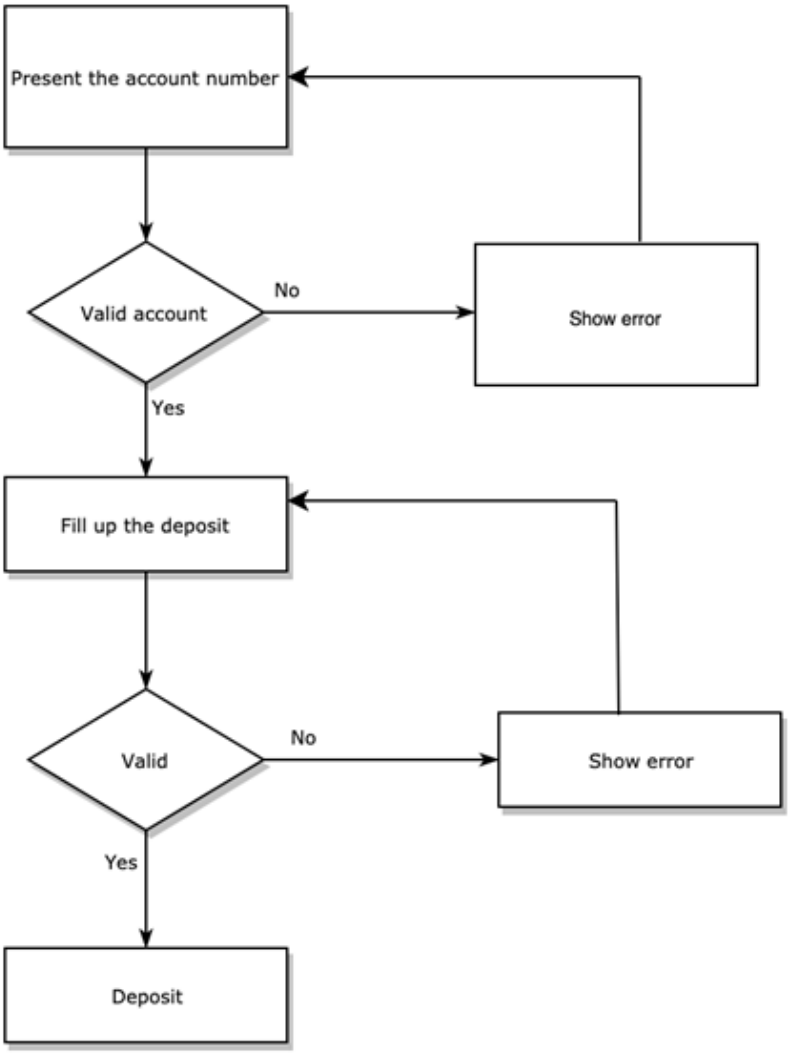


Figure2. 9: Sequence diagram for Deposit Module

The above activity diagram shows the flow of the different activities performed in the deposit module. The client's account is validated, and then if the validation is positive, then the deposit is made, otherwise error message is shown.

2.5. Data Modeling

An entity-relationship diagram (ERD) is a data modeling technique that graphically illustrates an information system's entities and the relationships between those entities. An ERD is a conceptual and representational model of data used to represent the entity framework infrastructure.

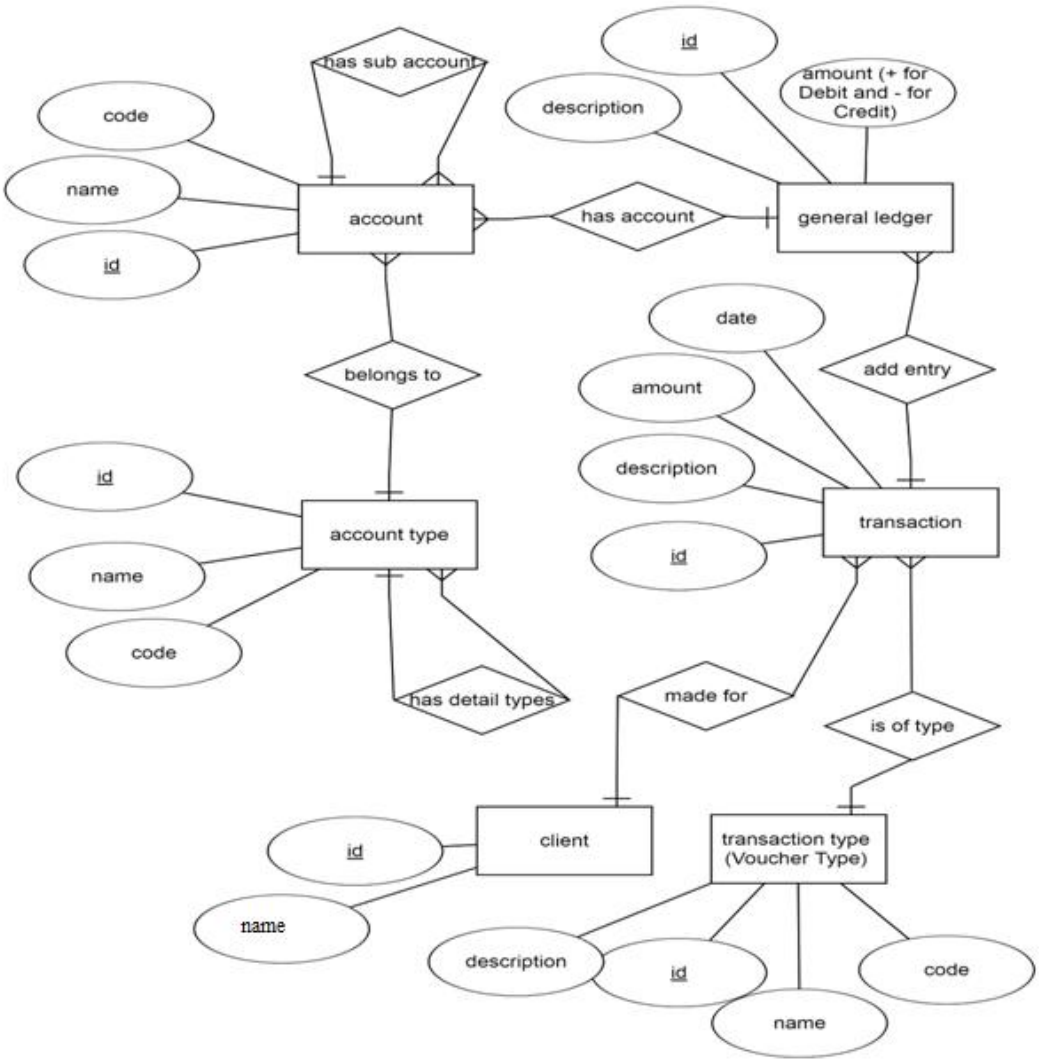


Figure2. 10: ER Diagram for General Transaction Processing

The above diagram shows various entities and their relationship with another entity. Each rectangle known as entity represents a table in database. Each oval known as attributes

represents the attribute of the entity which represents the columns of the table. The underlined attribute represents the primary key. Each diamond represents the relationship between the entities. The ‘account’ entity has some attributes such as id, name and code. Here ‘id’ is the key attribute that is used to link up other entities. It has relationship with ‘general_ledger’ attribute, ‘account_type’ attribute and “account” attribute itself. Similarly, other entities and their relationship with each other shapes up the database. They altogether define the schema of database.

Chapter 3
System Design

3. System Design

System design is the process of defining the architecture, modules, interfaces, and data for a system to satisfy specified requirements. System design could be seen as the application of system theory to product development.

With the detailed study of the requirements, the system architecture was developed. To develop the system architecture, different tools were used.

3.1 Architecture of the system

Systems Architecture is a generic discipline to handle objects (existing or to be created) called "systems", in a way that supports reasoning about the structural properties of these objects. Systems Architecture is a response to the conceptual and practical difficulties of the description and the design of complex systems.

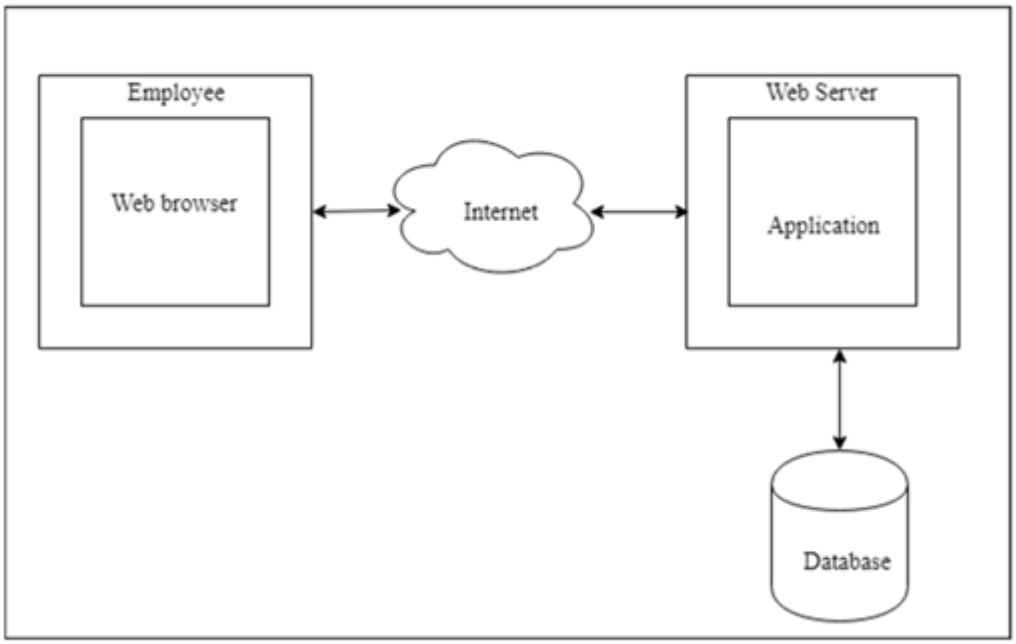


Figure 3. 1: Architecture of transaction processing system

The system is based on the client server architecture as we see in the diagram above that the front end connects to the backend of the system through the network connection. The system can be accessed through any browser by using an URL. The backend consists of server side application programs and a database. All the clients' data and information are stored in backend database.

3.2. Schema Design

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data. A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.

3.2.1 Loan Product Module

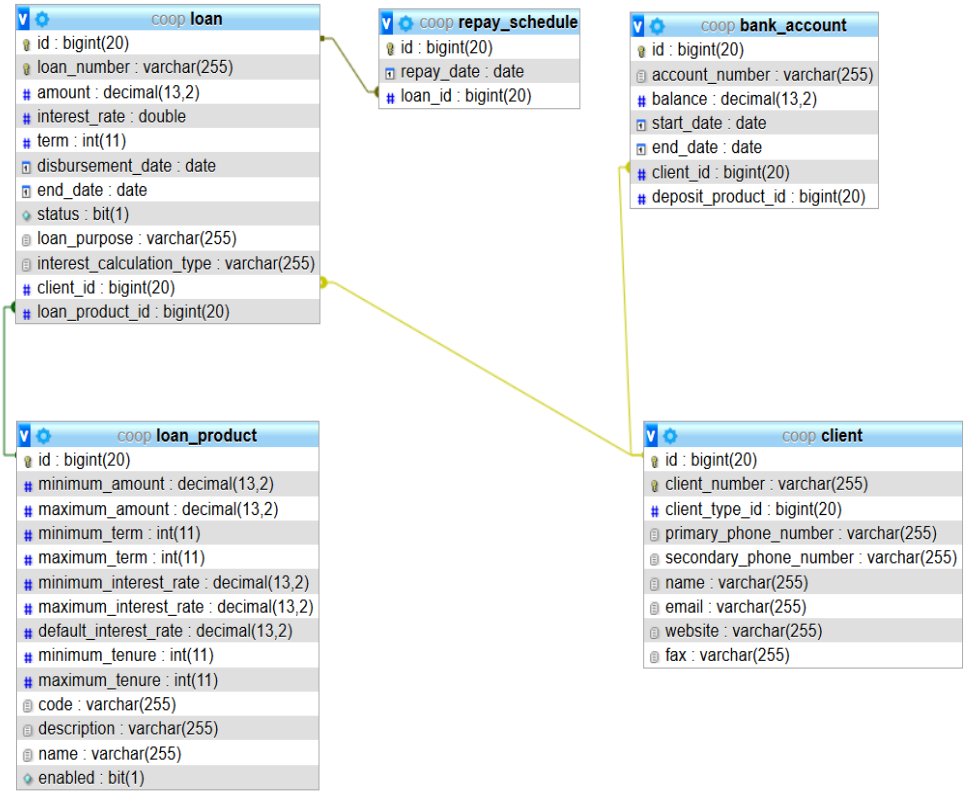


Figure 3. 2 Schema Diagram of loan product module

The loan product module consists of major entities like loan, loan_product, bank_account, client and repay_schedule. The attributes of each entity and the relationship between different entity sets are shown in figure 3.2.

3.2.2 Deposit Module

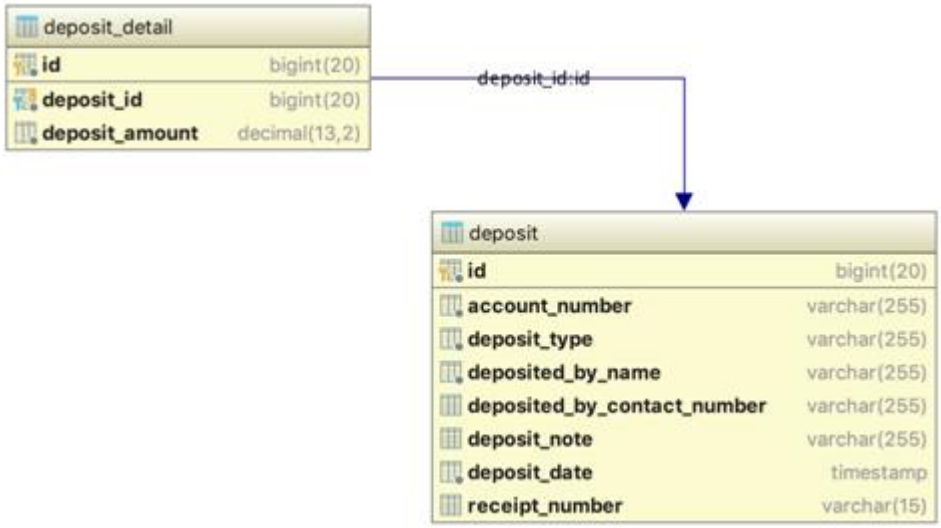


Figure 3. 3: Schema Diagram of Deposit module

The deposit module consists of major entities like deposit_detail and depost. The attributes of each entity and the relationship between different entity sets are shown in figure 3.3.

3.2.1 Withdraw Module

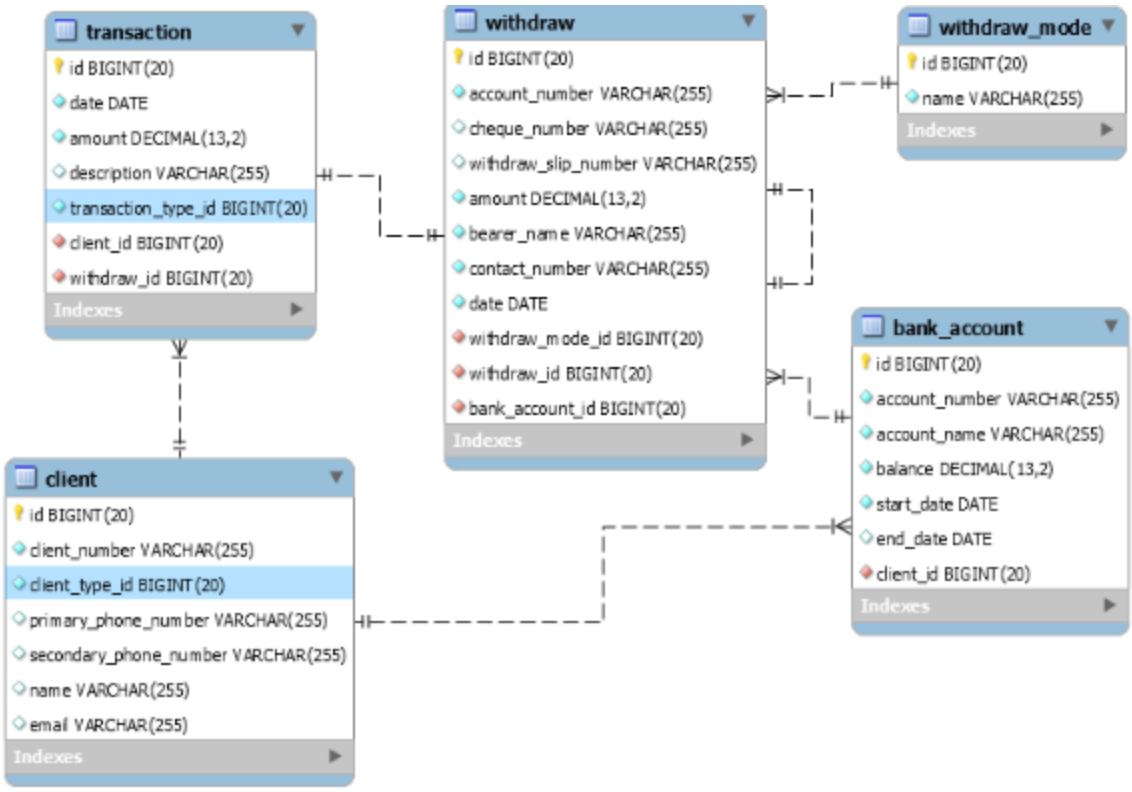


Figure 3. 4: Schema Diagram of Withdraw module

The withdraw module consists of major entities like transaction, withdraw, withdraw_mode, bank_accoun and client. The attributes of each entity and the relationship between different entity sets are shown in figure 3.4.

Chapter 4
Implementation and Testing

With inputs form the system design, the system was developed in small modules. The integration of all the modules forms a complete system. Each module were developed and tested for their functionality.

4.1 Tools Used

Various tools were used to achieve the desired output. Some important tools used are explained below:

1) Front end
    - HTML
    - CSS
    - Bootstrap
    - Angular 5
2) Java and Spring Boot framework
3) MYSQL
4) JSON
5) XML
6) Postman and Swagger
7) Liquibase
8) Git Lab

1) Front end

Designing front end includes design of responsive system's user interface (UI). It is what the user sees and interacts with. We used following tools to create UI platform let users communicate with our system:

HTML is the markup language that we use to structure and give meaning to our web content, for example defining paragraphs, headings, and data tables, or embedding images and videos in the page.

CSS is a language of style rules that we use to apply styling to our HTML content, for example setting background colors and fonts, and laying out our content in multiple columns.

Bootstrap is a free front-end framework for faster and easier web development. It includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other, as well as optional JavaScript plugins. It also gives us the ability to easily create responsive designs.

Angular is a platform that makes it easy to build applications with the web. It combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. It empowers developers to build applications that live on the web, mobile, or the desktop. It is used to develop front-end in this work.

2) JAVA and Spring Boot framework

For the back-end purpose, we used JAVA and Spring Boot framework with MAVEN as software project management and comprehension tool.

Java is a general-purpose computer-programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

Spring Boot is a JAVA framework that makes it easy to create stand-alone, production-grade Spring based Applications that we can "just run". It provides

opinionated view of the Spring platform and third-party libraries so we can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration. It also provides embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files).

3) MYSQL

MYSQL database is to deal data storage and manipulation operation. MYSQL is the most popular Open Source SQL database management system that is developed, distributed, and supported by Oracle Corporation. MYSQL databases are relational. Its Server is very fast, reliable, scalable, and easy to use. The Server works in client/server or embedded systems. It is used to create database.

4) JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

When exchanging data between a browser and a server, the data can only be text. JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server. We can also convert any JSON received from the server into JavaScript objects. This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

5) XML

XML stands for eXtensible Markup Language. It is a markup language much like HTML. It is designed to store and transport data. It is designed to be self-descriptive. We used XML for defining liquibase change set.

6) Postman and Swagger

Postman is a powerful HTTP client for testing web services.

Swagger is an open source software framework backed by a large ecosystem of tools that helps developers design, build, document, and consume RESTful Web services. While most users identify Swagger by the Swagger UI tool, the Swagger toolset includes support for automated documentation, code generation, and test case generation. We used both Postman and Swagger for testing APIs.

7) LIQUIBASE

Liquibase is an open-source database-independent library for tracking, managing and applying database schema changes. It allows easier tracking of database changes, especially in an agile software development environment. It provides version controlling mechanism for database.

8) GIT LAB

GitLab is a web-based Git-repository manager with wiki, issue-tracking and CI/CD pipeline features, using an open-source license, developed by GitLab Inc.The Continuous Integration, Delivery and Deployment (CI/CD) goal is pushing code frequently, and having it tested, built, and deployed.

We used GitLab as a repository management tool and CI/CD feature to get our code tested, built and deployed and to help locating errors if any.

4.2 Development Methodology

Evolutionary Development methodology was applied for the development of this project.

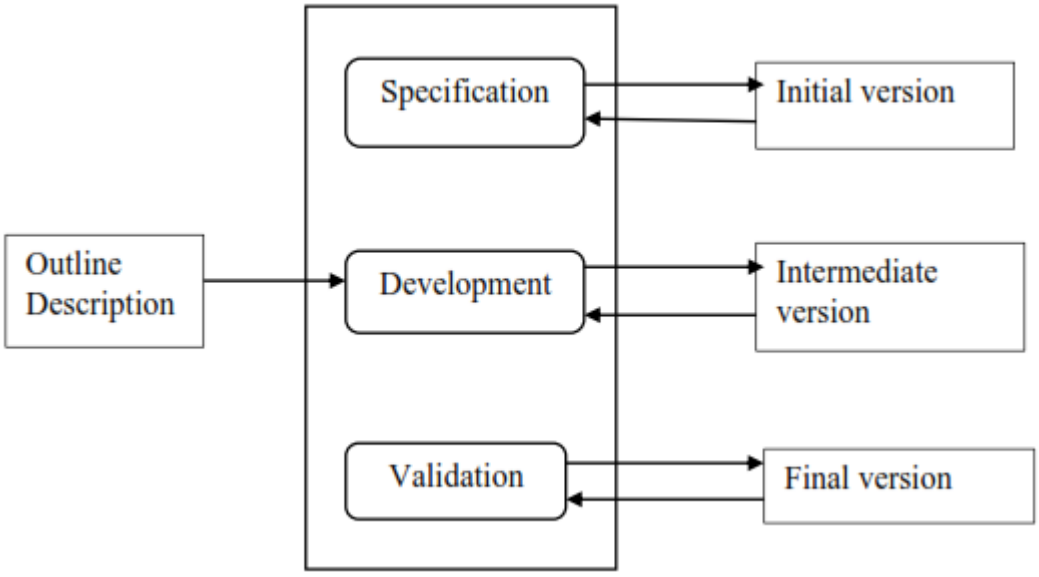


Figure 4. 1: Evolutionary Development Method

This model is based on the idea developing an initial implementation, exposing this to user comment and evolving it through several versions until the adequate system has been developed. Once the prototype is no longer required, it is discarded. It reflects a way to solve problems. It is a parallel model where backtracking is possible.

4.3 Integration and Testing

All the modules developed in the development phase were integrated together to form the system. Various modules are being merged together using GitLab interface. The integration of different modules made various problems to rise up. These errors were tackled with the use of GitLab CI/CD and also with the help of supervisor. Testing is being done only in module level.

In unit testing, individual modules or components of the system are tested. It is done to check that the module is working as intended. Testing of the withdraw module is shown by

the help of test case in the table below. Similarly, unit testing of all the other modules of the system have been done.

Test case for Withdraw module

Table 4.1: Test case for Withdraw module

Test case id	1
Module to be tested	Withdraw module
Assumption	Database contains Client and Account related data
Test Data	Account number:10101010 Cheque number: 10000000 Amount:50000
Test steps	Entered data are validated as: Account number exists Cheque number matches the one in database Amount is enough
Expected result	Withdrawl will be success
Result	Withdrawl successful
Comment	Module worked as expected

The withdraw module was tested to assure that it performs the function it is meant to perform. The account number is searched in database to make sure it exists. The cheque number is verified to assure that it is one of the cheque number that was issued to that particular account holder. The amount is checked to assure that the amount in the account of account holder is enough to make payment.

The module was executed, and mock account number and cheque number were used to assure the expected output. As the input matches and satisfies the data in database, the withdraw become success as expected.

Test case for Add Client

Table 4.2: Test case for Add Client Function

Test case id	2
Module to be tested	Add client module
Test Data	Name: Ram Kumar Nepal Gender: male Nationality: Nepal Date of birth: 12/12/1999 Citizenship no: 1101 Issue date: 01/012016 Place: KTM Address: Kirtipur-3 Phone: 334576
Test steps	All the required field checked
Expected result	Client addition will be success
Result	Addition successful
Comment	Module worked as expected

Test Case for Add Deposit

Table 4.3: Test case for Add Deposit module

Test case id	3
Module	Add Deposit
Assumption	Database contains Client and Account related data
Test Data	Account Number:10001005 Account Type: Saving Deposited By: Ram Sharma Contact: 9844544322 Note: Personal Deposit
Test steps	Field Validation for account number account type
Expected result	Deposit addition will be success
Result	Addition successful
Comment	Module worked as expected

CHAPTER 5
Conclusion and future work

Conclusion

Working as scheduled, finally an application was built using the tools such as angular, spring framework, MYSQL etc. The application provide simple standard UI which makes easier to perform transaction.

By using the system the cooperative users can add client to the system, design the products for account such as payroll account, youth account, normal saving account etc. The user can also design the loan product such as car loan, bike loan, education loan, house loan etc. The system also allows performing deposit, withdrawal, editing user profile etc. Although system has limited features, using the application cooperative will be benefitted by performing the transaction quickly, easily and also by reducing paperwork.

Future work

Since this work was completed within confined time period with limited resources, it will have some limitation such as it does not support the features of adding the detail of shareholder and share, employee, user authentication mechanism.

The module that deals with shareholder, share, and user's authentication employee can be added in future.

Bibliography

[1] (Docs: Angular, 2018), https://angular.io/docs

[2] (Documentation:Bootstrap, 2018), https://getbootstrap.com/

[3] (Spring:JournalDev, 2018), https://www.journaldev.com/

[4] (java: javaTpoint, 2018), https://www.javatpoint.com/

[5] (Docs:Gitlab, 2018), https://docs.gitlab.com/

[6] (*CSS-tutorial*, 2018), https://www.w3schools.com/css/

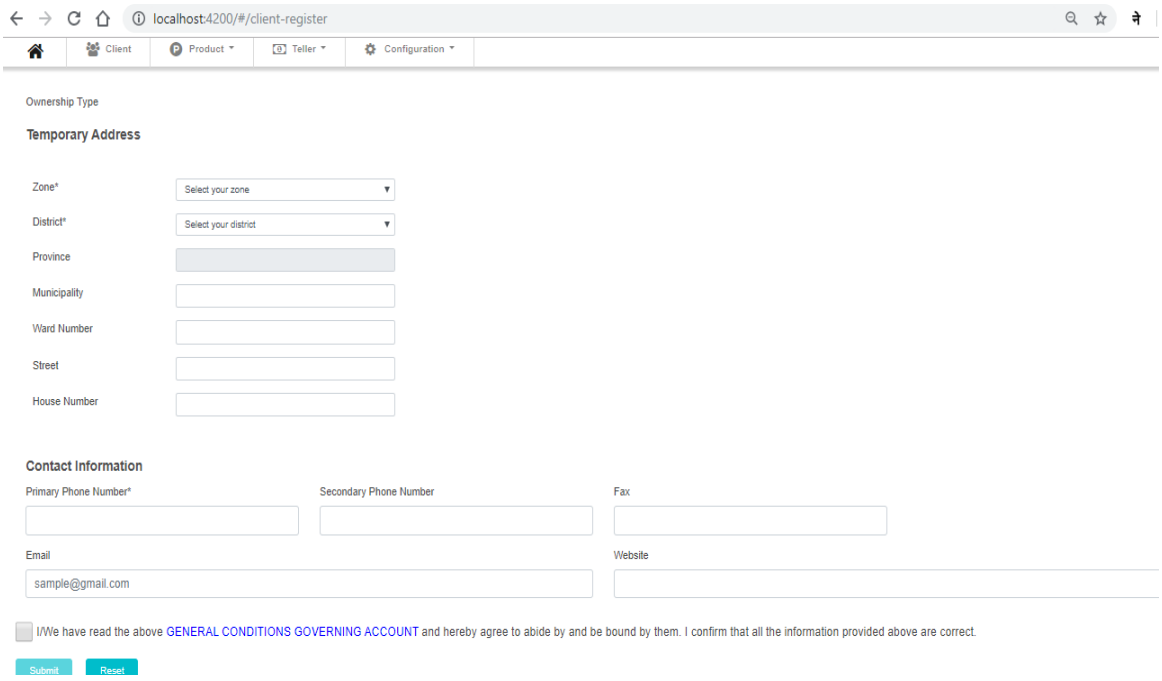
[7] (*CRUD in a Java App*, 2018), https://www.javatpoint.com/crud-create-read-update-delete-java-app/

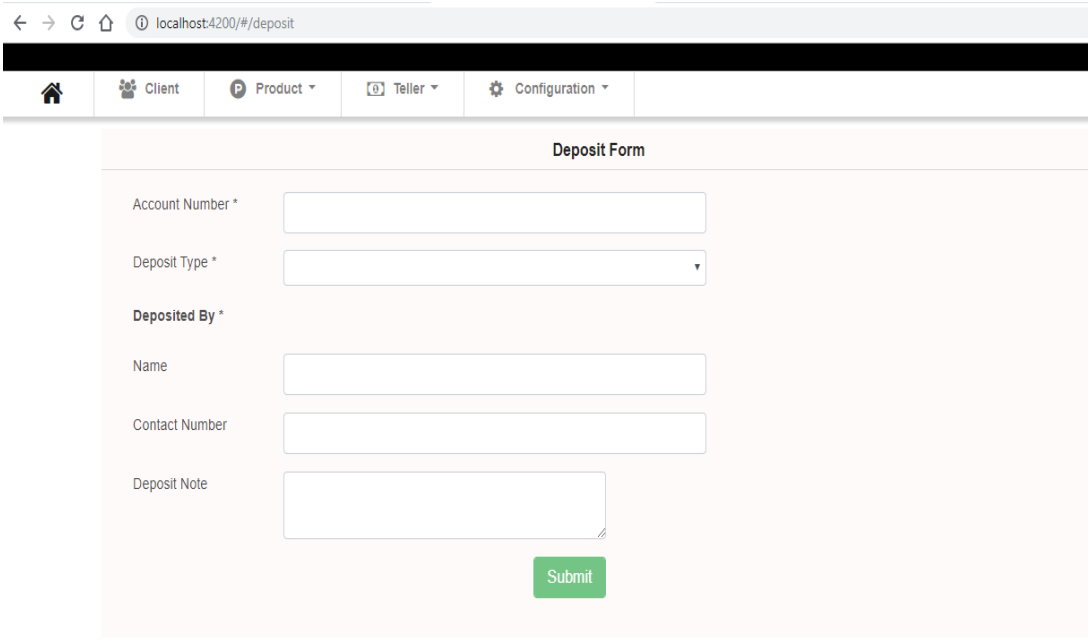
[8] (*RxJS 6 - What Changed? What's New?* , 2018). https://www.academind.com/learn/javascript/rxjs-6-what-changed/

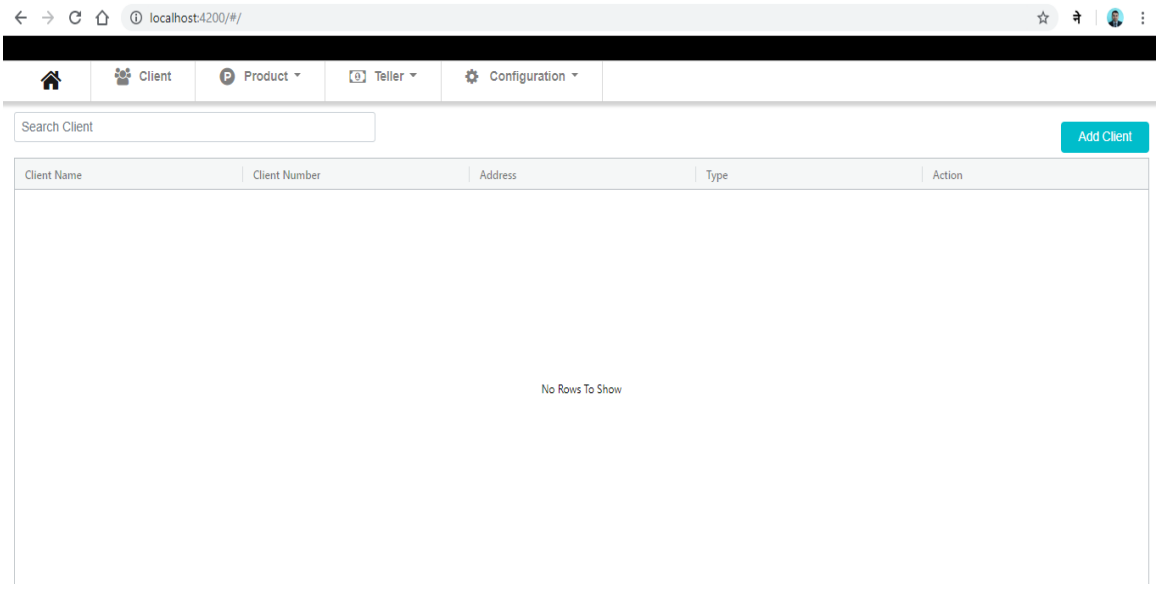
[9] (*Angular-Getting started*, 2018), https://angular.io/guide/quickstart

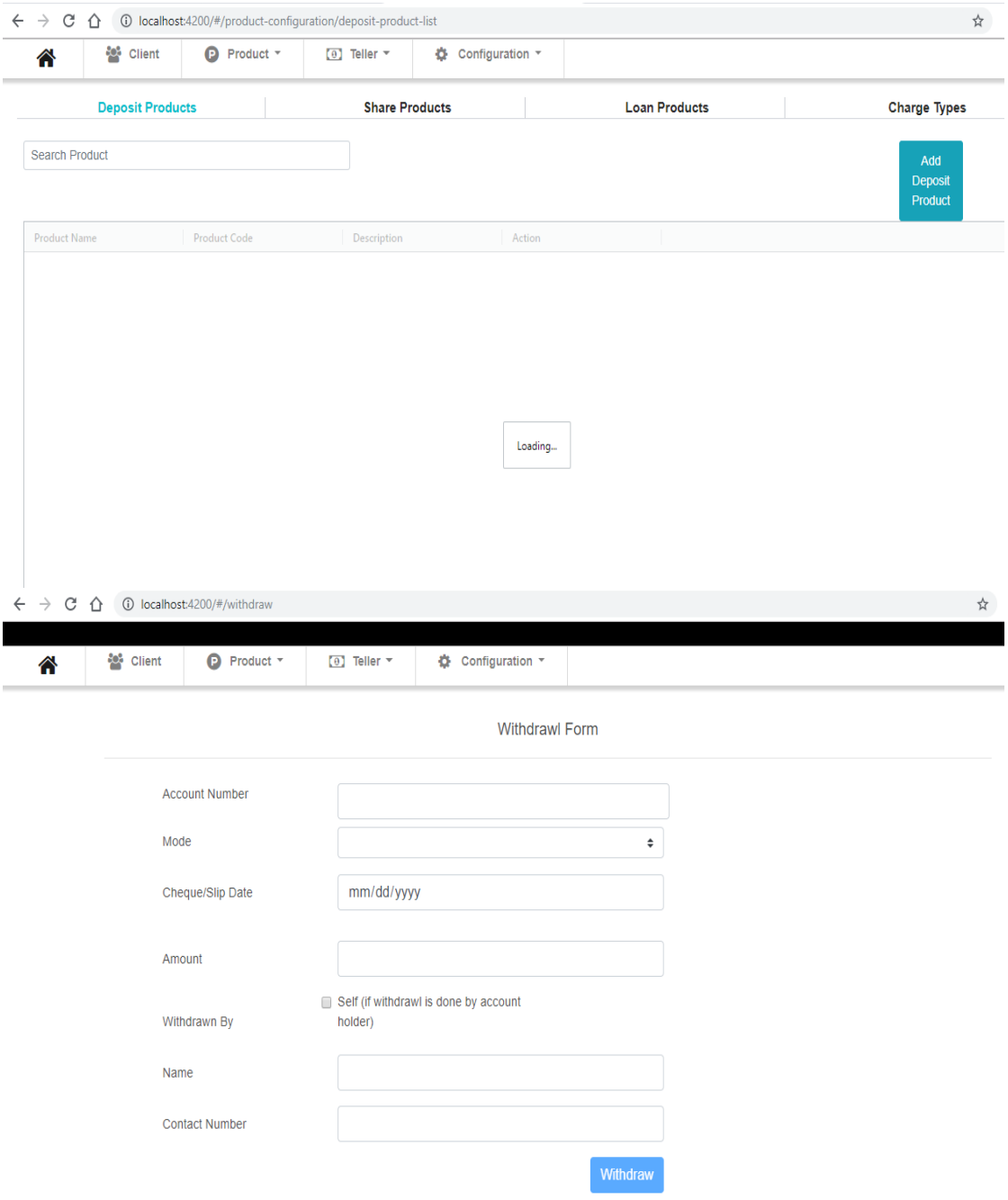
APPENDICES

Appendix: 1 Snapshots




localhost:4200/#/deposit
Client
Product
Teller
Configuration
Deposit Form
Account Number *
Deposit Type *
Deposited By *
Name
Contact Number
Deposit Note
Submit


localhost:4200/#/
Client
Product
Teller
Configuration
Search Client
Add Client
Client Name
Client Number
Address
Type
Action
No Rows To Show


localhost:4200/#/product-configuration/deposit-product-list
Client
Product
Teller
Configuration
Deposit Products
Share Products
Loan Products
Charge Types
Search Product
Add Deposit Product
Product Name
Product Code
Description
Action
Loading...
localhost:4200/#/withdraw
Client
Product
Teller
Configuration
Withdrawl Form
Account Number
Mode
Cheque/Slip Date
mm/dd/yyyy
Amount
Withdrawn By
Self (if withdrawl is done by account holder)
Name
Contact Number
Withdraw

Appendix: 2 Code

Withdraw.java

```
package com.proj.coop.withdraw;
import com.proj.coop.baseapi.BaseEntity;
import com.proj.coop.withdraw.withdraw_mode.WithdrawMode;
import lombok.Data;
import lombok.EqualsAndHashCode;
import javax.persistence.Column;
 import javax.persistence.Entity;
import javax.persistence.JoinColumn;
 import javax.persistence.ManyToOne;
 import javax.persistence.Table;
import java.time.LocalDate;

    @Data
    @Entity
    @Table(name = "withdraw")
    @EqualsAndHashCode(callSuper = false)
    public class Withdraw extends BaseEntity {
       @Column(name = "account_number")
       private String accountNumber;
       @ManyToOne(optional = false)
       @JoinColumn(name = "withdraw_mode_id")
       private WithdrawMode withdrawMode;
       @Column(name = "cheque_number")
       private String chequeNumber;
       @Column(name = "withdraw_slip_number")
       private String withdrawSlipNumber;
       @Column(name = "amount")
       private double amounts;
```

```
    @Column(name = "bearer_name")
    private String bearerName;
    @Column(name = "contact_number")
    private String contactNumber;
    @Column(name = "date")
    private LocalDate date;
}
```

WithDrawDto.java

```
package com.proj.coop.withdraw;
import com.fasterxml.jackson.annotation.JsonFormat;
import com.proj.coop.baseapi.BaseDto;
import lombok.Getter;
import lombok.Setter;
import java.time.LocalDate;

@Getter
@Setter
public class WithdrawDto extends BaseDto {
    private       String       accountNumber;
    private      BaseDto      withdrawMode;
    private String chequeNumber; private
    String  withdrawSlipNumber;  private
    double amounts;
    private String bearerName;
    private String contactNumber;
    @JsonFormat(pattern = "dd-MM-yyyy")
    private LocalDate date;
}
```

WithdrawMapper.java

```
package com.proj.coop.withdraw;
import com.fasterxml.jackson.annotation.JsonFormat;
import com.proj.coop.baseapi.BaseDto;
import lombok.Getter;
import lombok.Setter;
import java.time.LocalDate;

@Getter
@Setter
public class WithdrawDto extends BaseDto {

   private      String      accountNumber;

   private     BaseDto     withdrawMode;

   private      String      chequeNumber;

   private  String  withdrawSlipNumber;

   private double amounts;

   private String bearerName;

   private String contactNumber;

   @JsonFormat(pattern = "dd-MM-yyyy")
   private LocalDate date;
}
```

WithdrawMapper.java

```
package com.proj.coop.withdraw;
import   com.proj.coop.baseapi.mapper.BaseMapper;   import
com.proj.coop.baseapi.mapper.ReferenceMapper;       import
org.mapstruct.Mapper;
```

```
@Mapper(componentModel = "spring", uses = {ReferenceMapper.class})
public interface WithdrawMapper extends BaseMapper<WithdrawDto, Withdraw> {

}
```

WithdrawRepository.java

```
package com.proj.coop.withdraw;
import com.proj.coop.baseapi.BaseRepository;
public interface WithdrawRepository extends BaseRepository<Withdraw> {
}
```

WithdrawResource.java

```
package com.proj.coop.withdraw;
import com.proj.coop.baseapi.BaseResource;
import org.springframework.web.bind.annotation.CrossOrigin;
import
org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.cors.CorsConfiguration;

@RestController
@CrossOrigin(value= CorsConfiguration.ALL)
@RequestMapping(value = BaseResource.BASE_URL
+ WithdrawResource.RESOURCE_URL)
class WithdrawResource extends BaseResource<Withdraw, WithdrawDto> {
public static final String RESOURCE_URL = "/withdraw";
public WithdrawResource(WithdrawService withdrawService, WithdrawMapper
withdrawMapper) {
    super(withdrawService, withdrawMapper, Withdraw.class,
    QWithdraw.withdraw);
```

```
    }
}
```

WithdrawService.java

```
package com.proj.coop.withdraw;
import com.proj.coop.baseapi.BaseService;

public interface WithdrawService extends BaseService<Withdraw> {

}
```

WithdrawService Impl.java

```
package com.proj.coop.withdraw;
import com.proj.coop.baseapi.BaseServiceImpl;
import org.springframework.stereotype.Service;
import javax.inject.Inject;
@Service
public class WithdrawServiceImpl extends BaseServiceImpl<Withdraw> implements
WithdrawService {

    @Inject
    public WithdrawServiceImpl(WithdrawRepository withdrawRepository) {
        super(withdrawRepository);
    }
}
```

withdraw.component.ts

```
import { Component, ElementRef, Input, OnInit, Output, ViewChild } from
'@angular/core';
import { Router } from '@angular/router';
import { WithdrawService } from './withdraw.service';
import { Withdraw } from './withdraw.model';
```

```
import { debounceTime, distinctUntilChanged, switchMap } from 'rxjs/operators';
import { Observable } from 'rxjs/internal/Observable';
import { Subject } from 'rxjs/internal/Subject';
import { BankService } from '../bank-account/bank.service';
import { WithdrawModeService } from './withdraw-mode/withdraw-mode.service';
import { Bank } from '../bank-account/bank.model';
import { WithdrawMode } from './withdraw-mode/withdraw-mode.model';

@Component({
  selector: 'app-withdraw',
  templateUrl: './withdraw.component.html',
  styleUrls: ['./withdraw.component.scss']
})

export class WithdrawComponent implements OnInit {
  withdraw: Withdraw = new Withdraw();
  checkBoxFlag: boolean = false;
  dateFormatFlagChild: boolean;
  futureDateFlagChild: boolean;
  pastDateFlagChild: boolean;

  accounts$: Observable<Bank[]>;
  private searchTerms = new Subject<string>();
  searchDisplayFlag = false;
  accountNumber = '';
  clientName = '';
  clientPhoneNumber = '';
```

```
selectedModeValue: number;
withdrawModeValues: WithdrawMode[];


constructor(
  private router: Router,
  private withdrawService: WithdrawService,
  private bankService: BankService,
  private withdrawModeService: WithdrawModeService) {
}

getDateFormatFlagChild(flag: boolean): void {
  this.dateFormatFlagChild = flag;
}

getFutureDateFlagChild(flag: boolean): void {
  this.futureDateFlagChild = flag;
}

getPastDateFlagChild(flag: boolean): void {
  this.pastDateFlagChild = flag;
}

createWithdraw(withdraw: Withdraw): void {
  this.withdrawService.createWithdraw(this.withdraw).subscribe(data => {
   //  this.toasterService.pop('success', 'Withdraw', 'Operation Successful :)');
    alert("withdrawl successful");
  });
  this.withdraw.chequeNumber          =          null;
  this.withdraw.withdrawMode.id       =          null;
  this.withdraw.withdrawSlipNumber    =          null;
  this.withdraw.accountNumber         =          null;
  this.withdraw.amounts               =          null;
  this.withdraw.bearerName            =          null;
```

```
  this.withdraw.contactNumber = null;
  this.withdraw.date =null;
  this.withdraw.checkBox=false;
 }
 isChecked(element: HTMLInputElement) {
  if (element.checked) { this.checkBoxFlag = true;
   this.withdraw.bearerName = this.clientName;
   this.withdraw.contactNumber = this.clientPhoneNumber;
  }
  else
   {
  this.checkBoxFlag = false; this.withdraw.bearerName = '';
  this.withdraw.contactNumber = '';
   }
  return this.checkBoxFlag;
 }

 searchAccount(term: string): void {
  this.searchTerms.next(term);
  this.searchDisplayFlag = true;
 }

 getWithdrawMode() {
  this.withdrawModeService.getWithdrawMode().subscribe(response =>
this.withdrawModeValues = response);
 }

 getListValue(accountNumber: string, clientName: string, clientPhoneNumber: string):
void {
  this.searchDisplayFlag = false;
  this.withdraw.accountNumber = accountNumber;
  this.accountNumber = accountNumber;
```

```
  this.clientName = clientName;
  this.clientPhoneNumber = clientPhoneNumber;
 }

 getWithdrawModeValue(modeValue: number) {
  this.selectedModeValue = modeValue;
  return this.selectedModeValue;
 }

 validateNumber(event: any) {
  if (event.keyCode === 38 || event.keyCode === 40) {
     event.preventDefault();
  }
 }

  validateName(event: any) {
  if (!(event.keyCode < 48 || event.keyCode > 57)) {
     event.preventDefault();
  }
 }

 resetSlipNumber(): void {
  this.withdraw.withdrawSlipNumber = null;
  resetChequeNumber(): void {
  this.withdraw.chequeNumber = null;
 }

  ngOnInit(): void {
  this.accounts$ = this.searchTerms.pipe(
    debounceTime(200), distinctUntilChanged(),
    switchMap((term: string) => this.bankService.searchAccount(term)),
  );
  this.searchDisplayFlag = true;
  this.getWithdrawMode();
 }
}
```

Deposit module

```
@Data
@Entity
@Table(name = "deposit")
public class Deposit extends BaseEntity {

    @Column(name = "account_number")
    private String accountNumber;

    @Column(name = "deposit_type")
    private String depositType;

    @Column(name = "deposited_by_name")
    private String depositedByName;

    @Column(name = "deposited_by_contact_number")
    private String depositedByContactNumber;

    @Column(name = "deposit_note")
    private String depositNote;

    @Column(name = "deposit_date")
    @CreationTimestamp
    @JsonFormat(pattern = "yyyy-MM-dd, HH:mm:ss")
    private LocalDateTime depositDate;

    @Column(name="receipt_number")
    private String receiptNumber;

    @OneToMany(mappedBy = "deposit", cascade = CascadeType.ALL)
```

```
    private List<DepositDetail> depositDetail;

    public void setDepositDetail(List<DepositDetail> depositDetail) {
        for (DepositDetail depositDetail1 : depositDetail) {
            depositDetail1.setDeposit(this);
        }
        this.depositDetail = depositDetail;
    }
}
```