# Software Requirements Specification (SRS)

Project Title

# Banking Management System

Batch - 2


**Guide**

Dr. M. Priyadarshini

**Team Members**

Tharshith Goud – 19BCN7112

Sirisha Danda – 19BCN7261

Buddhsen Tripathi – 19BCE7134

Abhishek Basu – 19BCE7326

# 0. Problem Statement

Keeping track the all activities and their record on paper is very in-efficient, time-consuming process, unreliable and error prone. With the manual system it is difficult to update and maintain data. It may lead to inconsistency in data and security problems. To counter these Problems, we can use an Online Banking System.

# 1. Problem Analysis

## 1.1 Overview /Introduction

The purpose of this document is to present a detailed description of the **Banking Management System.** It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for both the stakeholders and the developers of the system.

## 1.2 Scope

An online banking system will be applicable everywhere, where banking exists. It will be more efficient and easier way to have a record on systems through which everyone can easily access it according to his rights as compared to the traditional banking system. Every bank will prefer the online banking system instead of the traditional banking system as it contains many useful features and fastest methods for the transactions.

## 1.3 Objectives/Purpose

The main purpose of using online banking system instead of traditional banking is that we can get the things (money transfer, Balance Enquiry, Bill Payments. Etc) done easily without visiting the bank Physically.

## 1.4 Infrastructure / Tools & Technologies

Front-end: ReactJs

Back-end: ExpressJs

Database: MySQL

Testing: Selenium

Deployment:

- Front-end: Heroku
- Back-end: AWS Elastic Beanstalk
- Database Queries: AWS Lambda
- Database Hosting: AWS RDS

## 1.5 Features

- Admin Panel for Bank Managers and Executives
- Simple Front-end Responsive UI for the customers to easily Interact with different options.
- Security features like amazon VPC will be used to keep the database private.
- Session Handling
- All the other basic banking features will also be implemented

# 2. Software Requirement Analysis and Planning

## 2.1 Description of Individual Module

### 2.1.1 User Characteristics

**Administrator:** Has full access to the application. Admin can create, read, delete and update all the data.

**Branch-Manager:** Has Access to the customers of a particular branch. Branch-Manager can create, read, delete and update data of his/her branch customers.

**Customer:** Customer can use all the basic banking features. He/she will be able to check balance, transfer money, retrieve statement.

### 2.1.2 General Constraints

Some general constraints should be defined which will have a great part in the overall succession of the online banking project.

**1) Hardware Requirements:** As this system is an online Web-based application so a client server will be the most suitable Organizational style for this system. Computer systems will be needed by each of the actor as well as that user must be connected to the internet. So, concisely following hardware will be needed.

**a)** Computer systems **b)** Internet availability

**2) Safety and Security:** This Project must be safe and secure because customers will directly contact their account through the internet. Software will have to identify the valid customer according to his/her bank details and password. So, it is a difficult task to prevent the system by major disasters by preventing the unauthorized access to the system.

### 2.1.3 Assumptions

Following are the assumptions and dependencies which are related to this online banking project.

**1)** This project is a stand-alone project so it will not affect the system where it will be embedded.

**2)** This project is a web-based project while the staff was addict of using traditional methods of data storage and retrieval so they will be trained a bit to jump to it.

**3)** This system will not depend on any other module. It will be a web-based so everyone will independently contact it.

**4)** It is will not affect the environment at all.

**5)** Banks will feel free to adopt it because it will not be so much expensive.

**6)** As this project contains valuable and new features so it will probably remove the previous online banking systems embedded in some banks.

### 2.1.4. Functional Requirements

Following are the services which this system will provide. These are the facilities and functions required by the customer.

**a)** Online balance check.

**b)** Balance transfer.

**c)** Online data entry by the staff.

**d)** Updating the data.

**e)** Check book Allotment.

### 2.1.5 Identify individual module deliverables

**Customer Login:** Each Customer will have its account Id and password. This page will require both attributes for them to access their account.

**Bank Features:** It isn't sure that each visitor of the Bank's website will be a customer. He/she would be a normal visitor interested in reading the features bank provides. The website's main page should provide him the basic features and benefits of the bank to these types of users.

**Order for an Account:** A new visitor the Bank's website would be interested in opening a new account in the Bank. So, he must be provided an easy path to create a new account in the bank.

**Fill the Form:** Newcomer should have to fill the form to register him/her with the bank. After filling the form, If the values inputted by the user were logical correct, his contact details will be sent to the administration block else he will be asked to input the values again.

**Welcome Page:** After a user will be login, he will provide an interface offering different tasks (Here this interface will provide many of the functionalities, which the customer needs in the software). He must choose a task to carry on his work.

**Staff Login:** On the Website main page, A staff login link will also be provided. Bank staff will use to input their ID's and passwords to access their account. Here the type of staff will also be recognized, if he will be of administration block, he will be sent to the administration module else he will be sent to the record management module.

**Check the balance:** After logging in, if the user wants to check his balance, he will have to click the balance check link. It will tell him his current balance of the account through which he is logged in.

**Transfer Balance:** If user wants to transfer his money to some other account, then this module will provide him this opportunity. He will input the account details of the receiver. After this process, server will check the balance of the

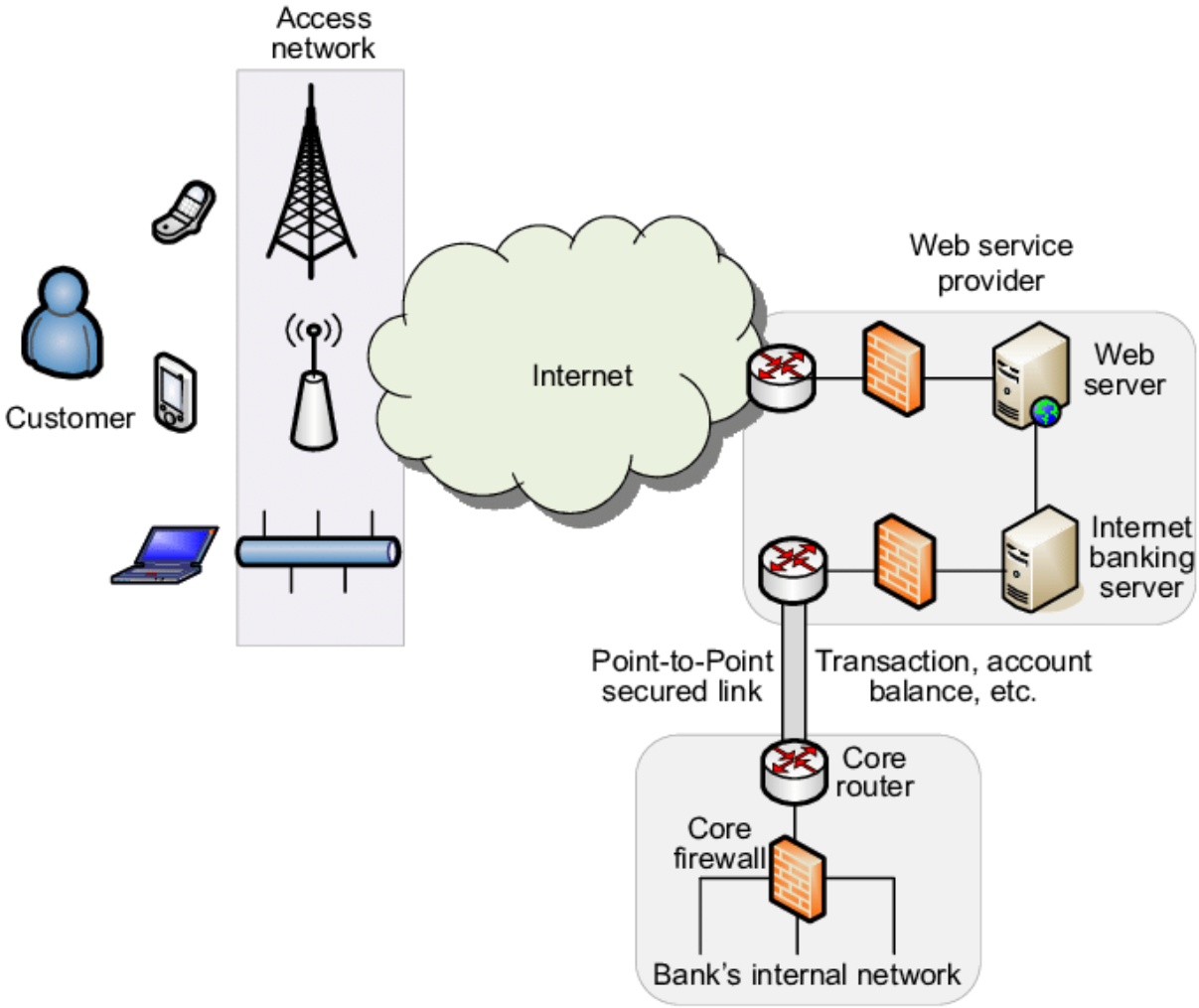
user and if the transfer balance will be less than the account balance then transfer will take place else, he will be alarmed that he has lo balance.

**Account detail teller:** If the user physically contacts the Bank branch, then he will provide his account detail to the management staff who will inform him about his account. User will be able to do every task at the branch that he can do online from his home.

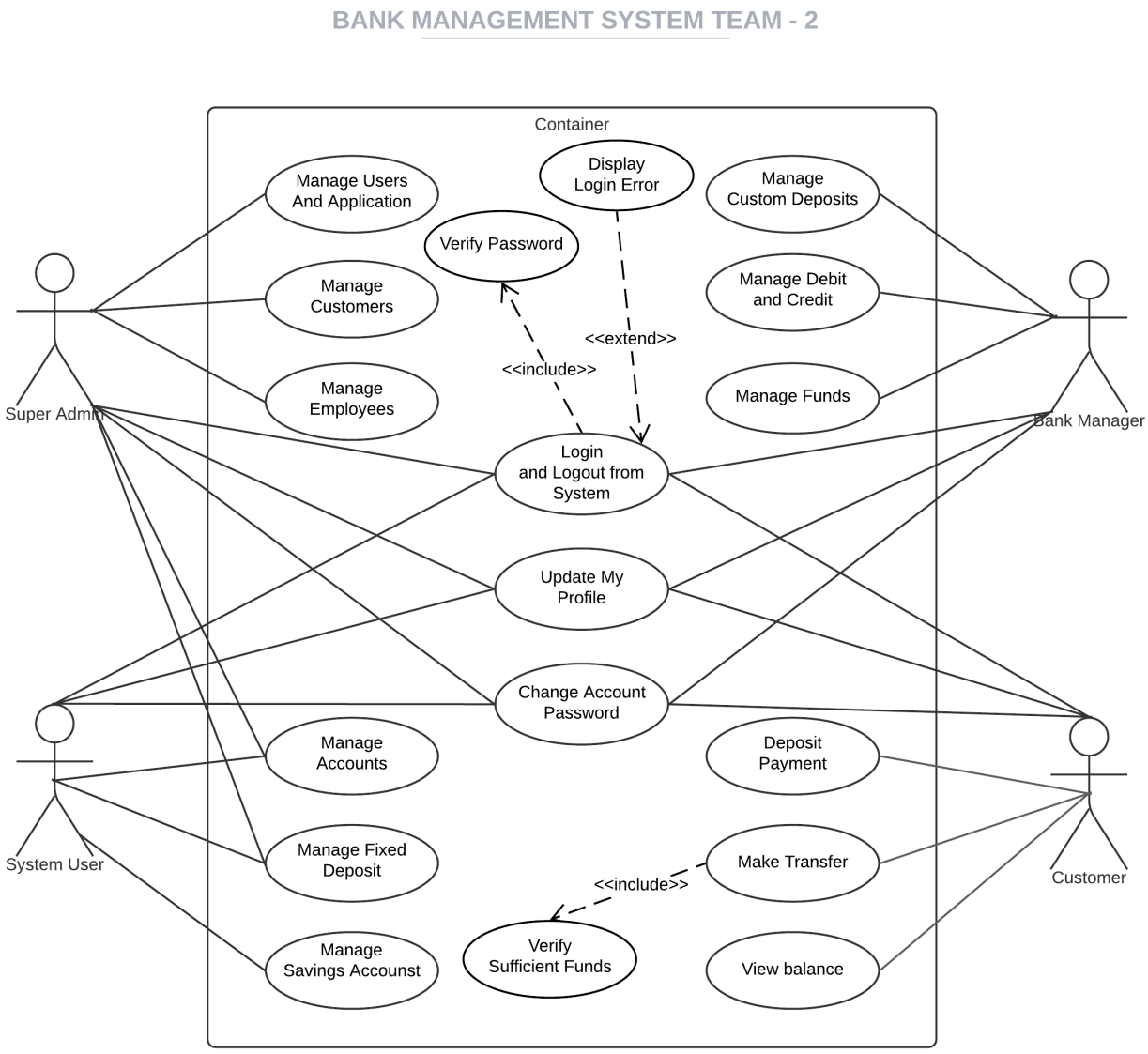
**Order Cash Book:** If user's Cheque book has been finished, he will be able to order a new cheque book from this module.

## 3. Data Modelling

### 3.1 Architecture Diagram

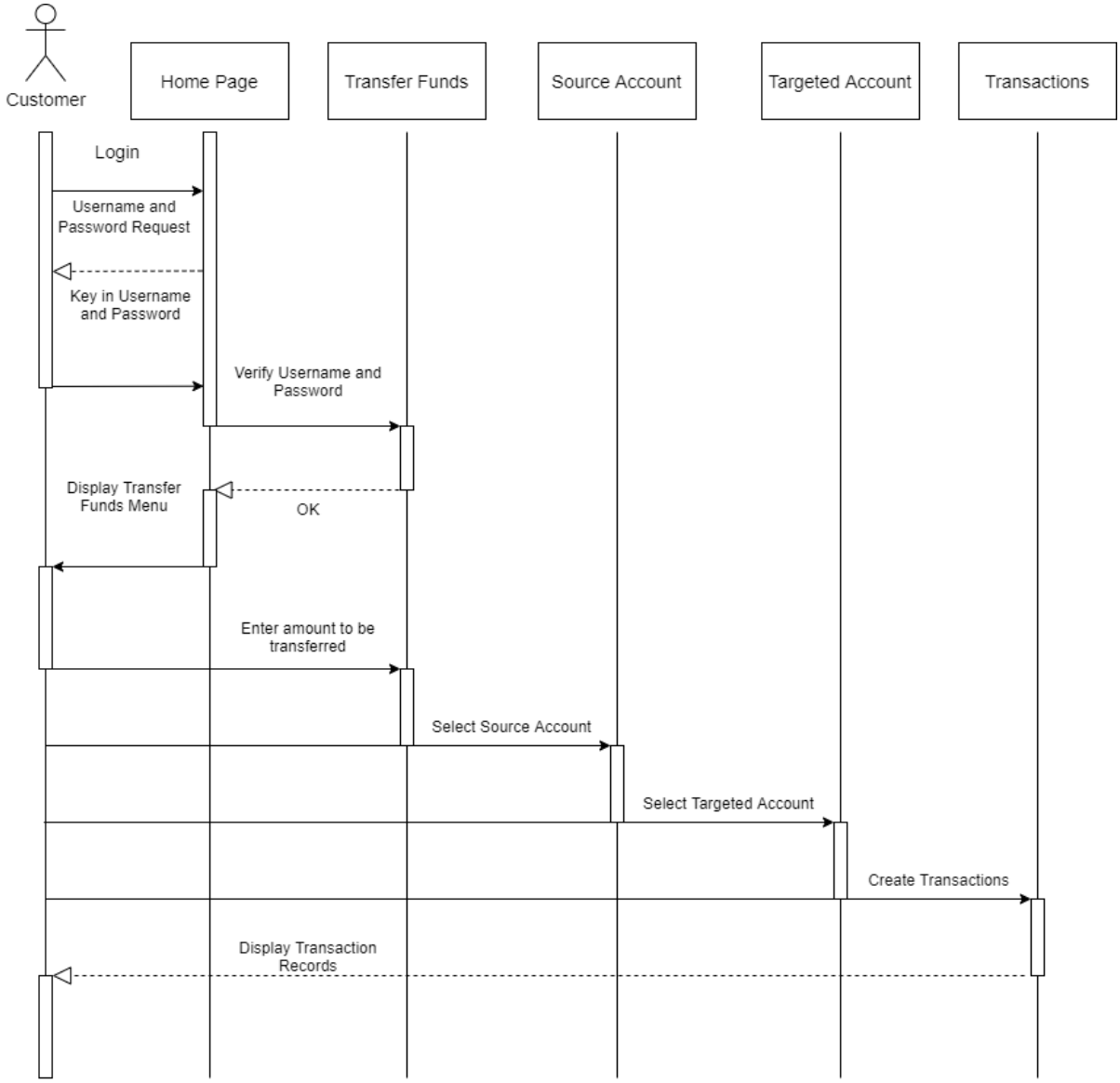


## 3.2 Use-case Diagram



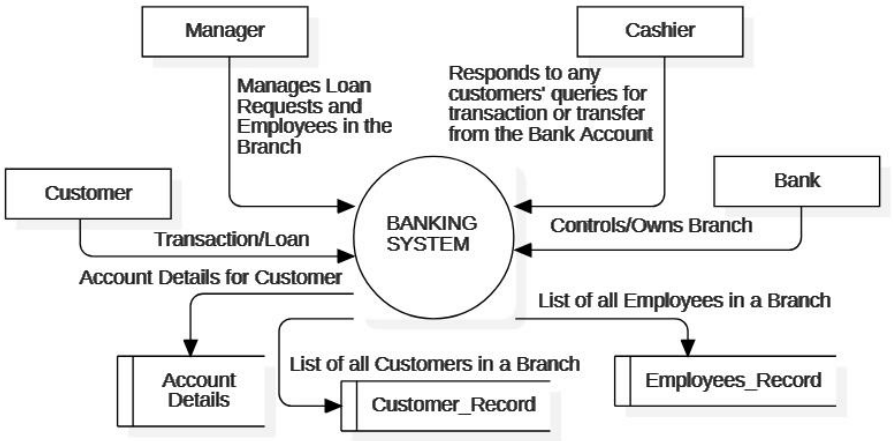
# 3. Data Modelling

## 3.3.1Activity Diagram (Single / Module-wise)

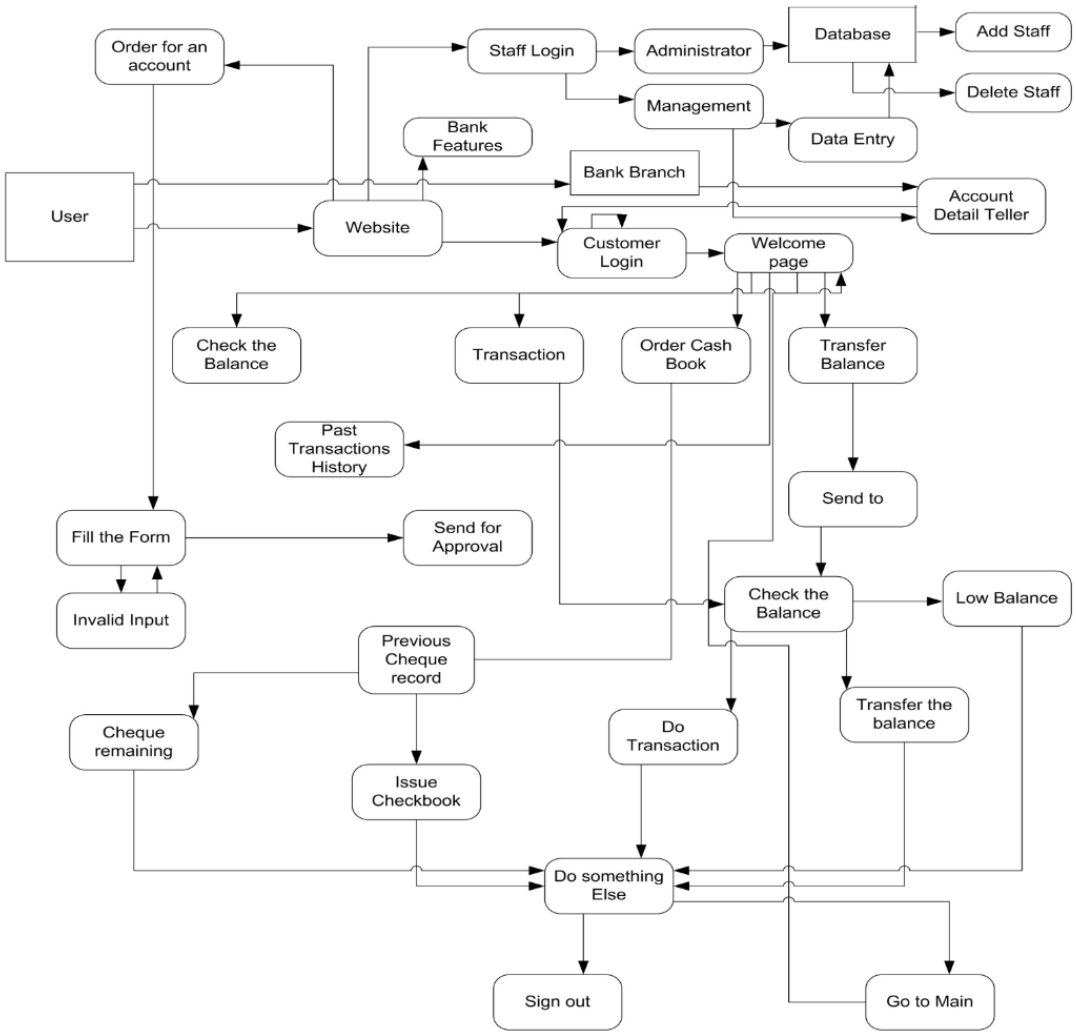


# 3. Data Modelling

## 3.4.1 Context Diagram (Logical)

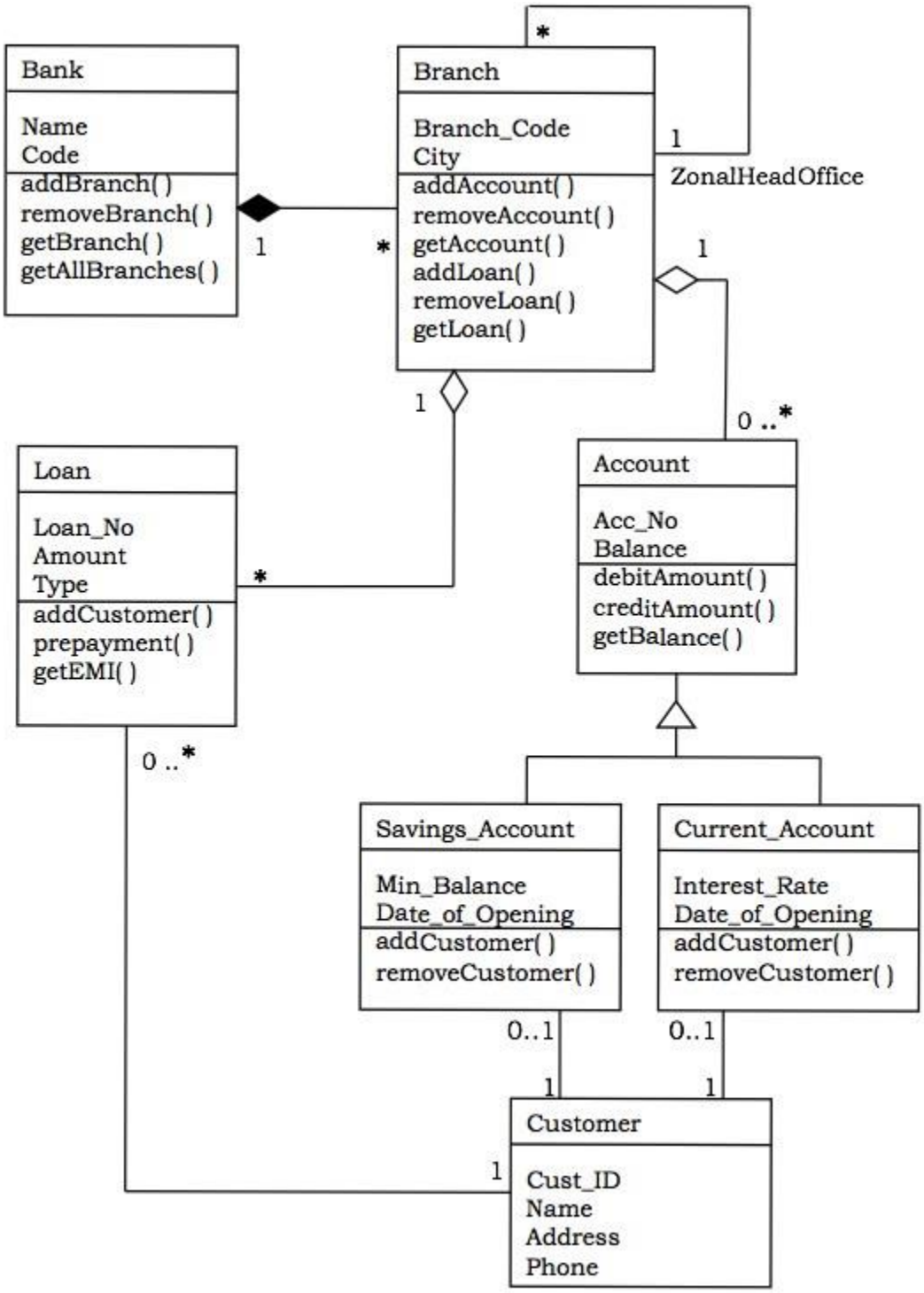



## 3.4.2 Data Flow Diagram (Physical)



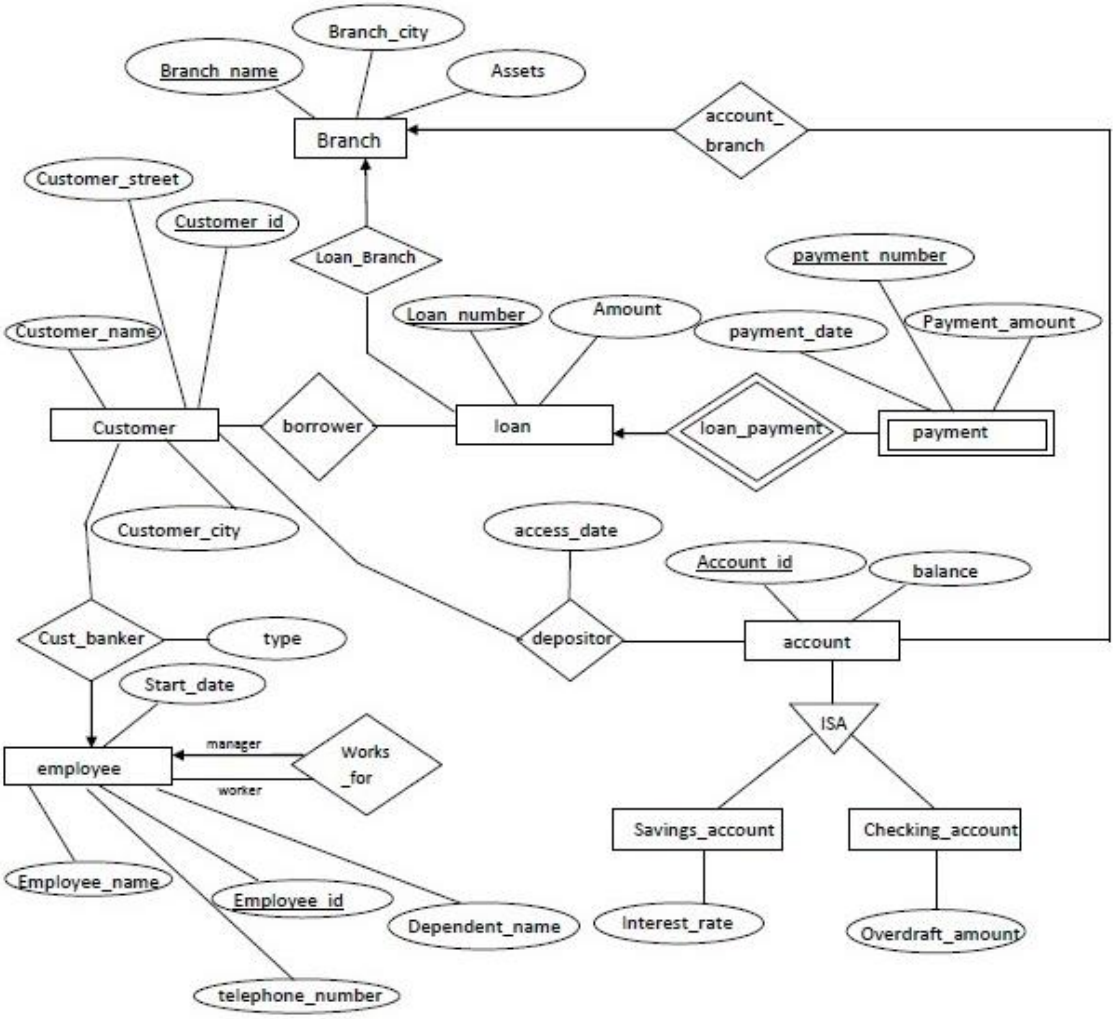
## 3. Data Modelling

3.5 Class Diagram



# 4. Development – Database Structure

## 4.1 E-R Diagram




## 4.2 Table Design

| | Table | Action | Rows | Type | Collation | Size | Overhead |
|---|---|---|---|---|---|---|---|
| ☐ | accountmaster | Browse Structure Search Insert Empty Drop | 4 | InnoDB | latin1_swedish_ci | 16 KiB | - |
| ☐ | accounts | Browse Structure Search Insert Empty Drop | 3 | InnoDB | latin1_swedish_ci | 16 KiB | - |
| ☐ | branch | Browse Structure Search Insert Empty Drop | 2 | InnoDB | latin1_swedish_ci | 16 KiB | - |
| ☐ | customers | Browse Structure Search Insert Empty Drop | 3 | InnoDB | latin1_swedish_ci | 16 KiB | - |
| ☐ | employees | Browse Structure Search Insert Empty Drop | 6 | InnoDB | latin1_swedish_ci | 16 KiB | - |
| ☐ | loan | Browse Structure Search Insert Empty Drop | 0 | InnoDB | latin1_swedish_ci | 16 KiB | - |
| ☐ | loanpayment | Browse Structure Search Insert Empty Drop | 1 | InnoDB | latin1_swedish_ci | 16 KiB | - |
| ☐ | loantype | Browse Structure Search Insert Empty Drop | 4 | InnoDB | latin1_swedish_ci | 16 KiB | - |
| ☐ | registered_payee | Browse Structure Search Insert Empty Drop | 2 | InnoDB | latin1_swedish_ci | 16 KiB | - |
| ☐ | transactions | Browse Structure Search Insert Empty Drop | 3 | InnoDB | utf8_general_ci | 16 KiB | - |
| | 10 tables | Sum | 28 | InnoDB | utf8_unicode_ci | 160 KiB | 0 B |

## 4. Development – Coding

a. Database Connection

```
const CONFIG = require("@config/config");
const mongoose = require("mongoose").set(
  "debug",
  CONFIG.current_env === "development" ? true : false
);

mongoose.connect(`${CONFIG.mongodb_uri}`, {
  useNewUrlParser: true,
  useCreateIndex: true,
  useFindAndModify: false,
  useUnifiedTopology: true,
});

mongoose.connection.on("error", (err) => {
  console.error(err);
  console.log("MongoDB connection error. Please make sure MongoDB is
running.");
  process.exit();
});
```

b. Authentication (ExpressJS)

```
const CONFIG = require('@config/config');
const bcrypt = require('bcryptjs');
const crypto = require('crypto');
const jwt = require('jsonwebtoken');
const User = require('@models/user');
const { throwError, passError, handleValidationErrors } =
require('@util/errors');

exports.register = async (req, res, next) => {
   const user = new User(req.body);
   try {
      await user.save();
      if (!user) {
         throwError('Problems creating a user', 422);
      }
      res.status(201).json({ message: 'User has been created' });
   } catch (err) {
      passError(err, next);
```

```
    }
};

exports.login = async (req, res, next) => {
    try {
        const { email, password } = req.body;
        let user = await User.findOne({ email }).lean();
        if (!user) {
            throwError('A user with this email could not be found', 422);
        }
        let isPassOk = await bcrypt.compare(password, user.password);
        if (!isPassOk) {
            throwError('Wrong password!', 401);
        }
        const { _id } = user;
        const token = jwt.sign({ id: _id, email }, CONFIG.jwt_secret_key, {
            expiresIn: CONFIG.jwt_expiration
        });
        res.status(200).json(token);
    } catch (err) {
        passError(err, next);
    }
};
```

c. Connecting Front-End with Back-End (ReactJS)

```
import axios from 'axios';

const BASE_URL = 'http://localhost:3001';

let config = {
    authToken: ''
};

export const getAPIConfig = () => ({ ...config });

export const updateAPIConfig = newConfig => {
    config = {
        ...config,
        ...newConfig
    };
};

export const callAPI = (endpoint, method = 'get', data) => {
    return new Promise((resolve, reject) => {
```

```
      axios({
         method,
         headers: {
            Authorization: `Bearer ${getAPIConfig().authToken}`
         },
         url: `${BASE_URL}${endpoint}`,
         data
      })
         .then(res => resolve(res.data))
         .catch(err => {
            reject({
               status: (err.response && err.response.status) || '',
               message: err.message || ''
            });
         });
   });
};
```

d. Getting transaction Data from Back-End (ReactJS)

```
import { callAPI } from './base';

export const getMyTransfers = (params = '') =>
callAPI(`/transfers/my/${params}`);

export const getSingleTransfer = id => callAPI(`/transfers/${id}`);

export const createTransfer = data => callAPI(`/transfers`, 'post', data);
```

e. Home Page (ReactJS)

```
import React from "react";
import { connect } from "react-redux";

import IncomeStats from "components/Widgets/IncomeStats";

const PanelHome = ({ user }) => {
  return (
    <div className="row panel-content">
      <div className="col-md-12">
        <h1>
          Welcome {user.firstName} {user.lastName}
        </h1>
      </div>
```

```
      <div className="col-md-8">
        <IncomeStats />
      </div>
    </div>
  );
};

const mapStateToProps = (state) => {
  return {
    user: state.profile.data,
  };
};

export default connect(mapStateToProps)(PanelHome);
```

f. Card Details Code (Express JS)

```
const Card = require('@models/card');

const { throwError, passError, handleValidationErrors } =
require('@util/errors');

exports.getMyCards = async (req, res, next) => {
   try {
      let cards = await Card.find({ owner: req.user._id }).lean();
      if (!cards) {
         throwError('No cards found', 422);
      }
      res.status(200).json(cards);
   } catch (err) {
      passError(err, next);
   }
};

exports.getSingle = async (req, res, next) => {
   try {
      let card = await Card.findOne({ _id: req.params.id, owner: req.user._id
}).lean();
      if (!card) {
         throwError('No card found', 422);
      }
      res.status(200).json(card);
   } catch (err) {
      passError(err, next);
   }
};
```

```
exports.changePin = async (req, res, next) => {
   try {
      const card = await Card.findOne({ _id: req.params.id, owner:
req.user._id });
      if (!card) {
         throwError("Card not found or it doesn't belong to you", 422);
      }
      card.pin = req.body.pin;
      const saved = await card.save();
      if (!saved) {
         throwError('Pin could not be changed', 422);
      }
      res.status(200).json({ status: 'Card pin changed' });
   } catch (err) {
      passError(err, next);
   }
};

exports.changeLimits = async (req, res, next) => {
   try {
      const { dailyOnlineLimit, dailyWithdrawalLimit } = req.body;
      const card = await Card.findOne({ _id: req.params.id, owner:
req.user._id });
      if (!card) {
         throwError("Card not found or it doesn't belong to you", 422);
      }
      card.dailyOnlineLimit = dailyOnlineLimit;
      card.dailyWithdrawalLimit = dailyWithdrawalLimit;
      const saved = await card.save();
      if (!saved) {
         throwError('Limits could not be changed', 422);
      }
      res.status(200).json({ status: 'Card limits changed' });
   } catch (err) {
      passError(err, next);
   }
};
```

g. Account Schema (Mongo DB)

```
const mongoose = require('mongoose');

const accountSchema = new mongoose.Schema(
   {
      type: {
         type: String,
         required: true
```

```
      },
      owner: {
         type: mongoose.Schema.Types.ObjectId,
         required: true,
         ref: 'User'
      },
      isActive: {
         type: Boolean,
         default: true
      },
      ifsccode: {
         type: Number,
         required: true,
         minlength: 6,
         maxlength: 6
      },
      number: {
         type: Number,
         required: true,
         minlength: 8,
         maxlength: 8
      },
      currency: {
         type: String,
         required: true
      },
      balance: {
         type: Number,
         required: true
      }
   },
   {
      timestamps: true
   }
);

const Account = mongoose.model('Account', accountSchema);

module.exports = Account;
```

## 5. Software Testing

### 5.1 Test Plan

| S.NO | TEST CASE ID | TEST CASE DESCRIPTION | PRECONDITION | TEST DATA | EXPECTED RESULT | POST CONDITION | ACTUAL RESULT | STATUS | COMMENTS |
|---|---|---|---|---|---|---|---|---|---|
| 1 | UT001 | User Interface | Figma Design | Dimensions of different Components | UI Looking as Intended | Good Looking UI | UI Looking as Intended | Pass | UI is Interpreted as in Design |
| 2 | UT002 | Responsive Web Page | Basic Frontend Implementation | Dimensions of Different Devices | Webpage Adjusting for device | Responsive Web Page on all Devices | Webpage Adjusting for device | Pass | Looking good on all Devices |
| 3 | IT001 | Connecting to Cloud DB(AWS) | AWS Setup | DB Credentials | Successful Connection to DB | DB functioning properly with backend | Successful Connection to DB | Pass | DB is integrated well |
| 4 | ST001 | Authentication and Session Handling | Database Implementation | Login Credentials | Should Logout After 5 min Inactivity | Webpage Logout | Auto Logout After 5 min Inactivity | Pass | Session Handling is implemented properly |
| 5 | NFT001 | DB Security Testing | Database is hosted in private subnet | AWS ACL (DB Access Control List) | DB should not be accessed easily | DB is Secure | DB cannot be accessed easily | Pass | Website is secure from DB attacks |
| 6 | AT001 | End-User Testing | Website URL | Login Credentials | Successful use of basic Banking Features | App should be ready for Production | Basic Banking Features are working as intended | Pass | Website is working as Intended |