

CERTIFICATE

This is to certify that the project entitled

“ATM Simulator”

Has been satisfactorily completed by

- 1) Vijay Balasaheb Jagtap
- 2) Utkarsh Anil Solanke
- 3) Anisha Bankat Umap
- 4) Dhiraj Shailesh warke
- 5) Bhavesh Sanjay Patil
- 6) Yash Pramod Talele

Under supervision and guidance for partial fulfilment of the
PG Diploma in Advanced Computing (DAC) (24 weeks Full Time) Course
of
Centre for Development of Advanced Computing (C-DAC), Pune.
at

Academy of Information Technology
(YCP) Nariman Point, Mumbai –400 021.

Faculty

Course Co-ordinator

Batch: PG DAC March 2024

Date: 19th Aug 2024.

PREFACE

In today's fast-paced financial environment, the need for seamless and secure banking experiences is more crucial than ever. The ATM Simulator Project is a pioneering initiative that redefines the traditional banking interface, bringing the future of financial transactions to your fingertips. With a focus on user convenience and technological innovation, our simulator offers a comprehensive suite of functionalities designed to make banking not just simple, but truly effortless.

Imagine a world where accessing your money is as easy as a few taps, whether you're using a card or going cardless. The ATM Simulator is your financial superhero, offering both card and cardless transactions. For cardless operations, our cutting-edge scanner technology ensures that your transactions are fast and secure. When using a card, the simulator provides versatile transfer options, allowing you to send money using either account numbers or UPI IDs. But that's not all—our ATM Simulator is equipped with a range of essential features to cater to your every banking need. From withdrawing cash and transferring amounts to accessing fast cash, checking balances, and obtaining mini statements, our system streamlines every aspect of banking, making it as easy as pie.

Join us on this journey as we transform the traditional ATM experience into a modern, user-friendly interface that makes financial management a breeze. Welcome to the ATM Simulator Project, where we're revolutionizing banking one transaction at a time. Welcome to the future of banking!

ACKNOWLEDGMENT

The successful completion of the ATM Simulator Project would not have been possible without the support and contributions of many individuals and organizations. We would like to express our deepest gratitude to all those who provided guidance, resources, and encouragement throughout this journey.

First and foremost, we extend our heartfelt thanks to our project supervisor, Siddhanath Bedake, whose invaluable insights, expertise, and unwavering support were instrumental in shaping this project. Your guidance and feedback have been essential in overcoming challenges and achieving our goals.

We also wish to acknowledge the contributions of our team members, whose dedication, creativity, and collaborative spirit made this project a reality. Each member's unique skills and tireless efforts have been crucial in developing and refining the various features of the ATM Simulator.

Special thanks to AIT-YCP Nariman Point for providing the necessary resources and a conducive environment for our work. Your support has been fundamental to the project's success.

To everyone who contributed, directly or indirectly, to the successful completion of this project, we say thank you. Your support has been the cornerstone of our success.

To everyone who has played a role, big or small, in making this project a reality – thank you from the bottom of our hearts.

Team Members;

- | | |
|---------------------------|--------------------------|
| 1) Bhavesh Sanjay Patil | 2) Utkarsh Anil Solanke |
| 3) Anisha Bankat Umap | 4) Dhiraj Shailesh warke |
| 5) Vijay Balasaheb Jagtap | 6) Yash Pramod Talele |

INDEX

SR.NO	TITLE	PAGE NO
1.	Introduction	2
2.	Software Requirement	3
3.	Hardware Requirement	3
4.	ER Diagram	4
5.	Data Flow Diagram	9
6.	Use Case Diagram	10
7.	Backend Details	11
8.	Frontend Details	12
9.	Interface Home Page Menu Page Withdraw Page Transfer Amount page Mini Statement page Fast Cash page Generate QR code Page QR code Page	11-16
10.	Coding	16-24
11.	Future Scope	32
12.	Conclusion	33
13.	Bibliography	34

INTRODUCTION

In today's technologically advanced society, the role of Automated Teller Machines (ATMs) has become integral to the daily operations of banking institutions and the financial lives of individuals. The ATM Simulator Project was conceived as a response to the growing need for a comprehensive, user-friendly platform that allows individuals to gain hands-on experience with various ATM functionalities without the associated risks of actual transactions.

The primary objective of this project is to develop a realistic and interactive ATM simulation that accurately mirrors the key functionalities of modern ATMs. This includes both card-based and cardless transactions, providing users with a versatile learning tool that caters to a wide range of banking operations. The project is designed to serve educational purposes, allowing users to familiarize themselves with ATM processes, enhance their financial literacy, and gain confidence in using ATM services.

The ATM Simulator encompasses a broad range of features that cover essential banking transactions. Users can perform cash withdrawals, balance inquiries, and fund transfers, with the option to transfer money using either account numbers or UPI IDs. Additionally, the simulator includes a cardless transaction mode that utilizes scanner technology to facilitate secure, contactless operations. Other key features include Fast Cash for quick withdrawals and the ability to generate mini statements, providing users with a summary of recent transactions.

The development of the ATM Simulator involved detailed research into the functionality and design of real-world ATMs, followed by the implementation of these features within a simulated environment. The project utilized a combination of software development tools and user interface design principles to create an intuitive and realistic user experience. Testing and validation were conducted to ensure that the simulator accurately reflects the operations of a real ATM.

Software Requirements:

- Eclipse IDE
- MySQL
- Visual Studio Code
- Browser
- Spring Boot Tool suite

Hardware Requirements:

- PC / Laptop
- i3 Processor minimum
- 4 GB Ram
- PC / Laptop

Tools Requirements :

- Postman
- Git
- GitHub

Use case Diagram

ATM Simulator Use Case Diagram

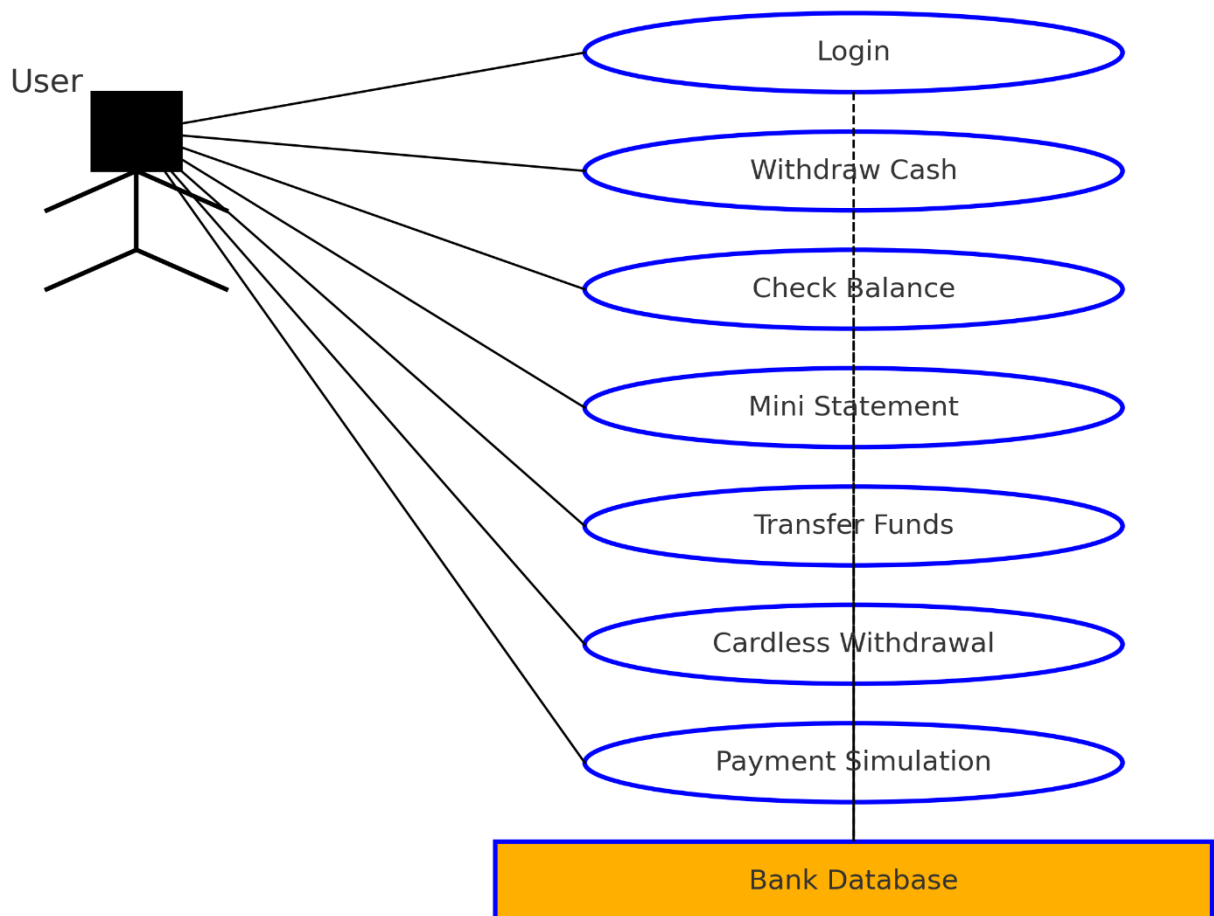


Diagram 1- Use Case Diagram

Data Flow Diagram

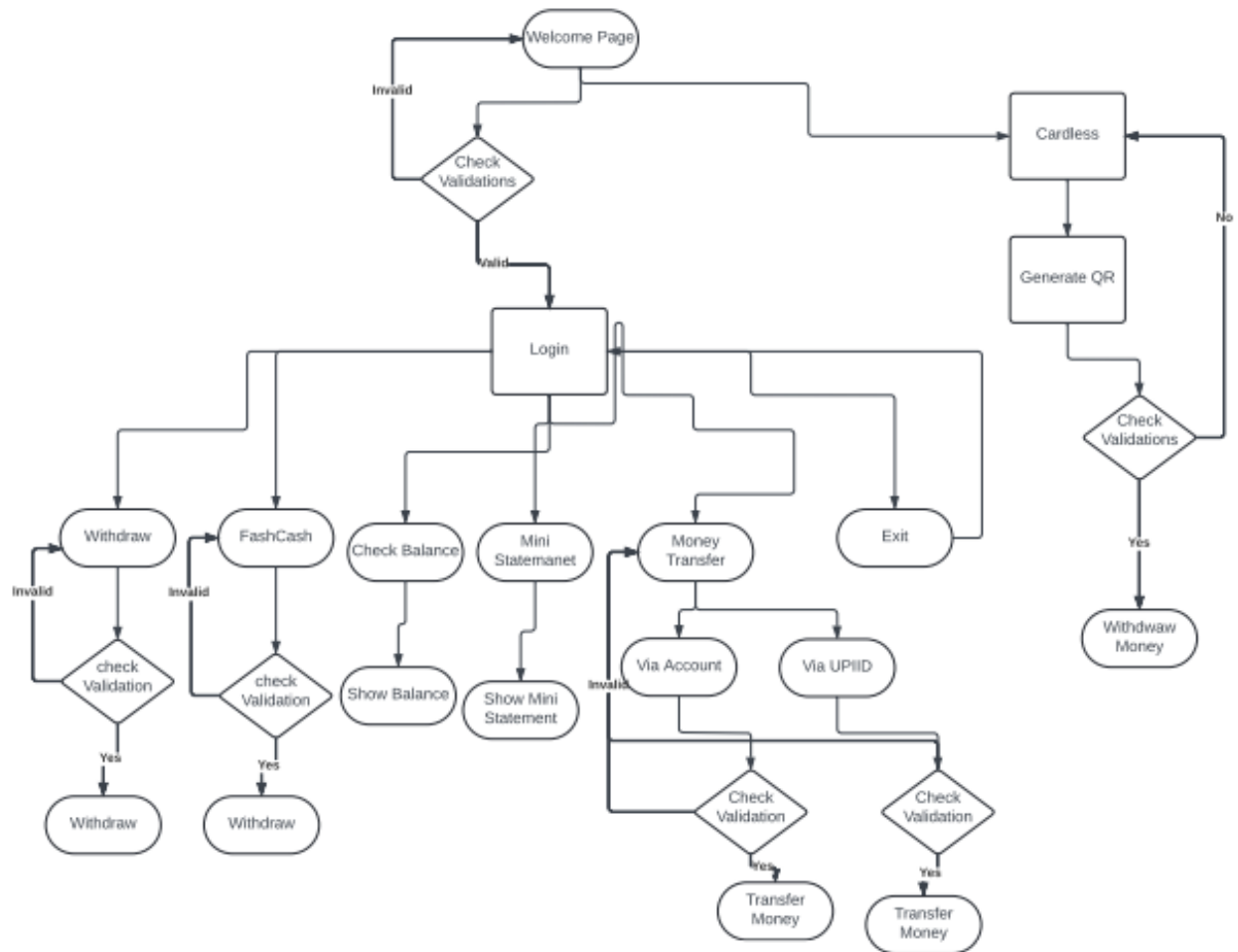


Diagram 2- Data Flow Diagram

ER Diagram

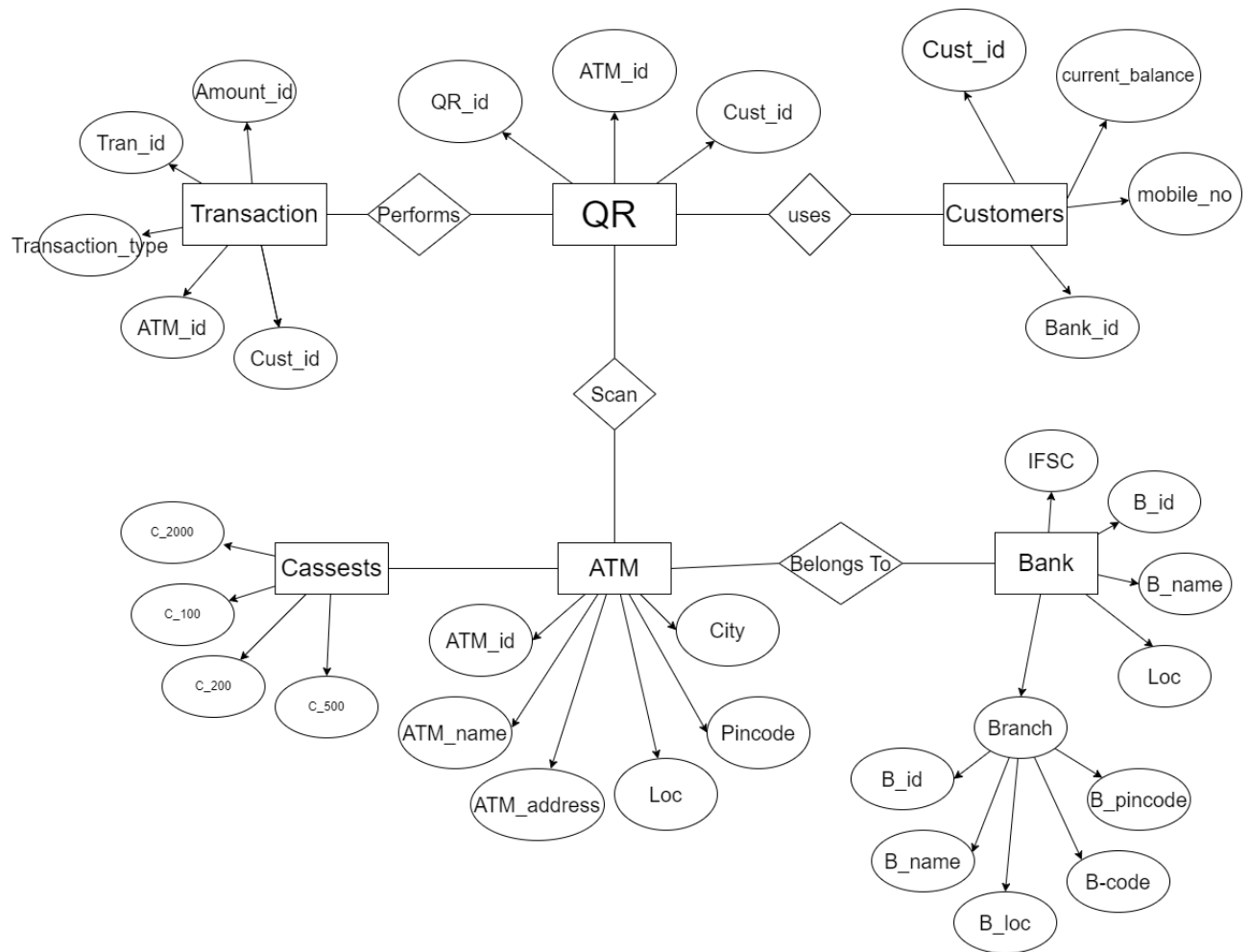


Diagram 3- ER Diagram

BACKEND DETAILS

What technology used?

- Spring Boot (Framework)
- MySQL (Database)

Dependencies:

- Spring web
- Spring-data-Jpa
- MySQL-connector
- Spring-security-core

Api End Points:

- User Login Functionality
- withdrawal functionality
- check balance functionality
- check balance functionality
- transfer amount by AC Number/ UPI ID functionality
- Cardless withdrawal functionality
- payment stimulator functionality

FRONTEND DETAILS

React.js, commonly referred to as React, is an open-source JavaScript library for building user interfaces (UIs) or user interface components. It was developed and is maintained by Facebook and a community of individual developers and companies. React is often used for building single-page applications and mobile applications.

What libraries are used in React?

- React-router-Dom
- Axios

User Login Functionality:

The Authenticate User process is responsible for verifying the identity of the user before allowing access to the ATM's features. Users must provide their credentials, such as a card number, PIN, or UPI ID, to log in.

Withdrawal Functionality:

The Withdraw Cash process allows users to withdraw cash from their accounts. Users specify the amount they wish to withdraw, and the system processes the transaction.

Check Balance Functionality:

The Check Balance process provides users with their current account balance.

Mini Statement Functionality:

The Mini Statement process generates a summary of recent transactions for the user.

Transfer Amount by Account Number/UPI ID Functionality:

The Transfer Funds process enables users to transfer money from their account to another account using either an account number or a UPI ID.

Cardless Withdrawal Functionality:

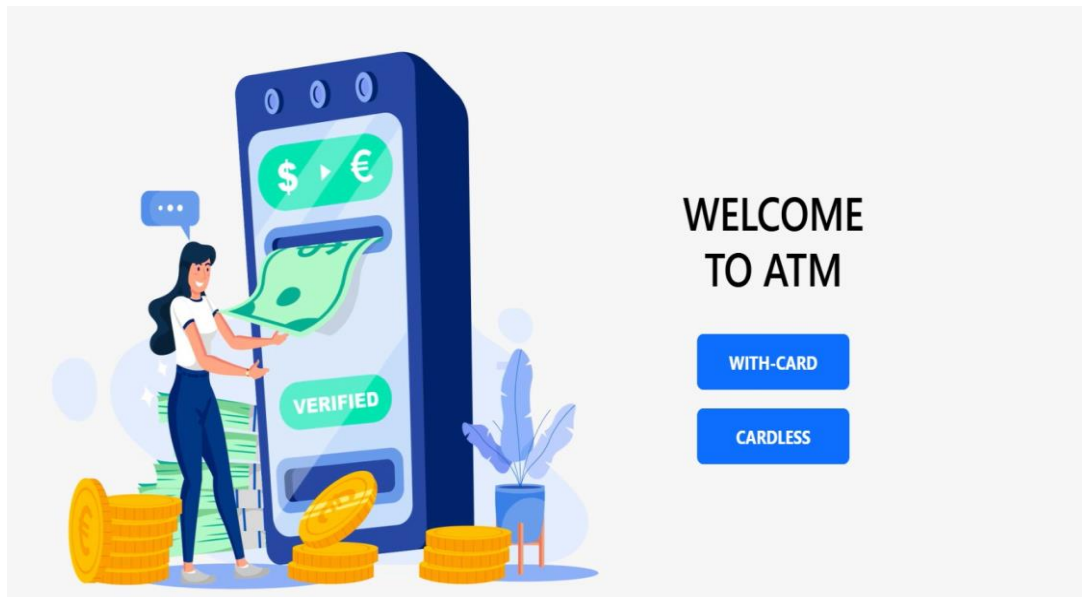
The Cardless Withdrawal process allows users to withdraw cash without using a physical card. Instead, the user might use a mobile device or a QR code.

Payment Simulator Functionality:

The Payment Simulation process allows users to simulate making payments, such as bill payments or online purchases, through the ATM interface.

Interface:

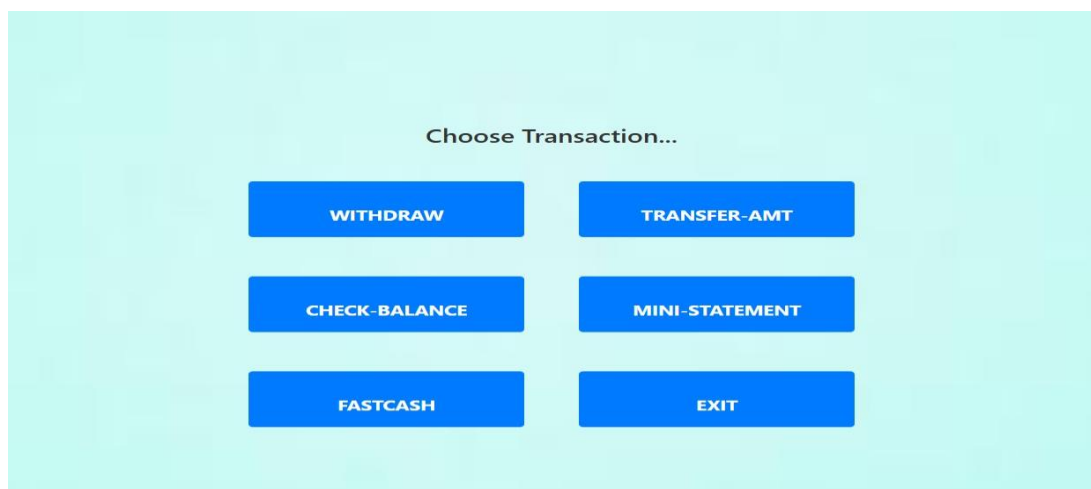
Home Page:



Screenshot-1

This is our Home Page you can see there is option for transaction customer can do the transaction with-card or without card.

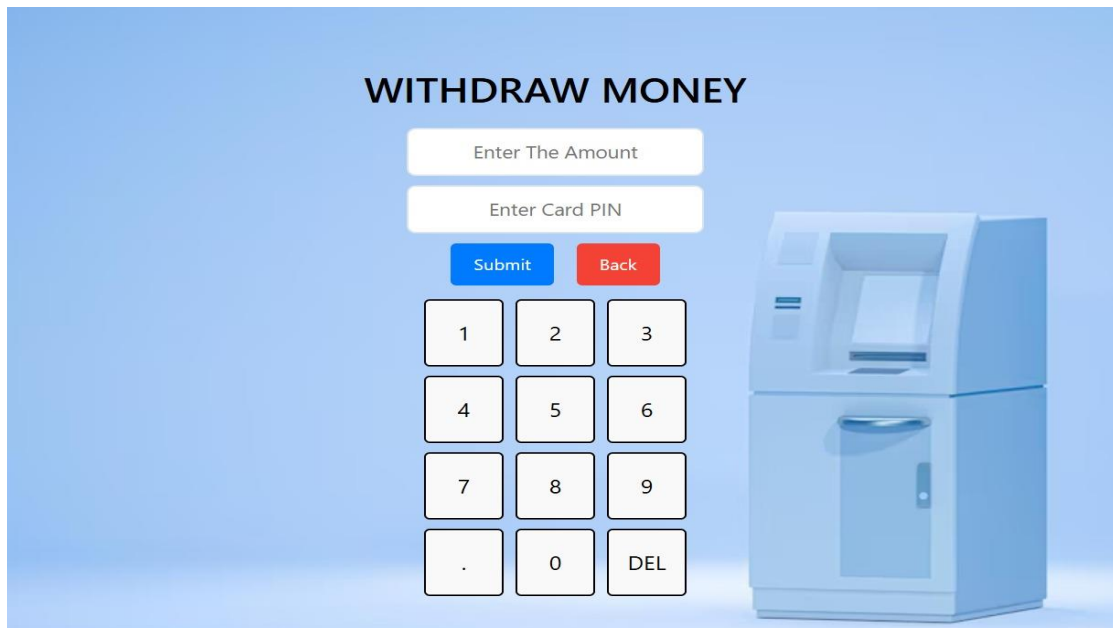
Menu Page:



Screenshot-2


This page having transaction option customer can select any of them and do the transaction.

Withdraw Page:

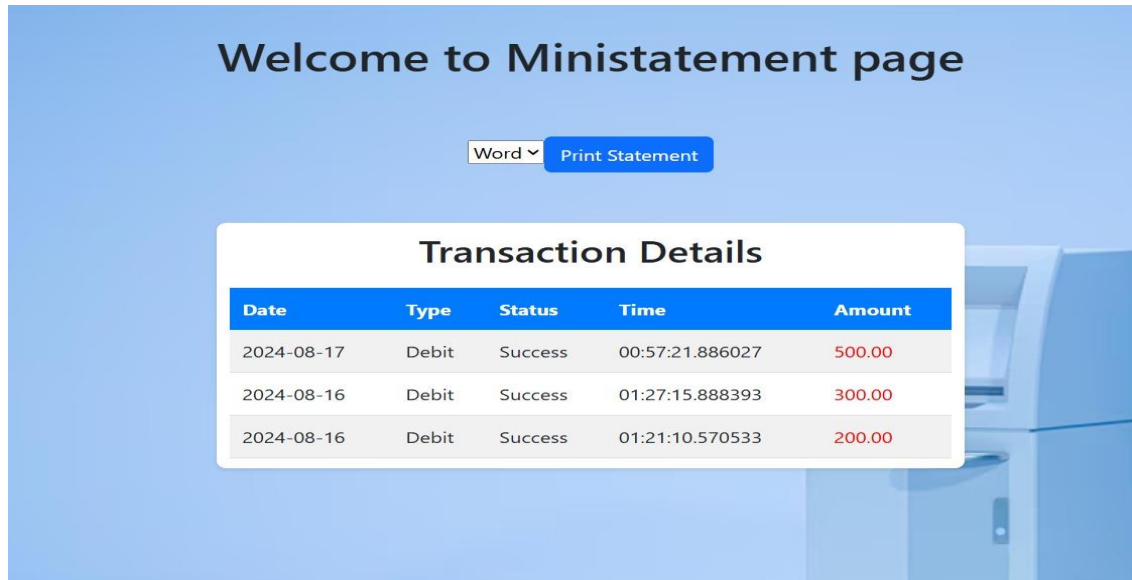
A screenshot of a mobile application's 'WITHDRAW MONEY' screen. The background is a solid light blue. On the right side, there is a 3D illustration of an ATM machine. The main content area on the left contains the title 'WITHDRAW MONEY' in bold black text. Below the title are two white input fields with rounded corners: the first is labeled 'Enter The Amount' and the second is labeled 'Enter Card PIN'. Below these fields are two buttons: a blue 'Submit' button and a red 'Back' button. Below the buttons is a numeric keypad with a 3x4 grid of buttons. The buttons are labeled with digits 1 through 9, a decimal point '.', a zero '0', and a 'DEL' key. The entire interface is clean and modern.

This is Withdraw page customer should enter the amount and correct pin.

Transfer Amount Page:

A screenshot of a mobile application's 'TRANSFER AMOUNT' screen. The background is a solid light blue. On the right side, there is a 3D illustration of an ATM machine. The main content area on the left contains the title 'TRANSFER AMOUNT' in bold white text. Below the title are three blue buttons with rounded corners: the first is labeled 'PROCEED WITH A/C NUMBER', the second is labeled 'PROCEED WITH UPI ID', and the third is a red button labeled 'BACK'. The entire interface is clean and modern.

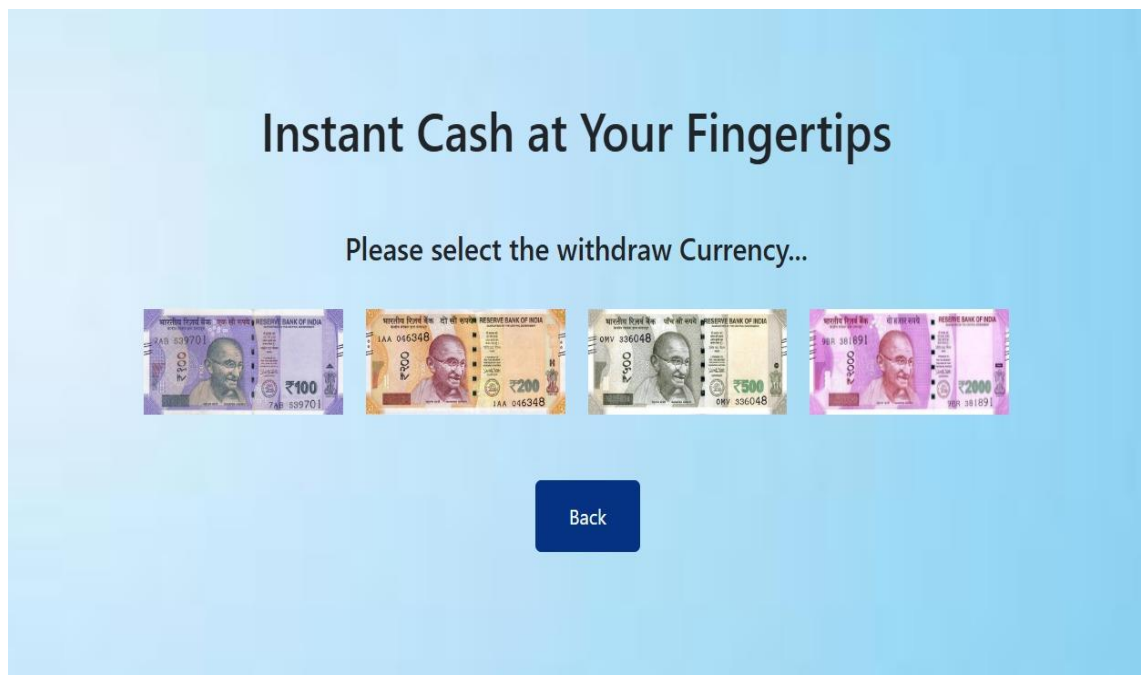
Mini Statement Page:



Screenshot-3

This is the page on which customer can see their last 3 transactions.

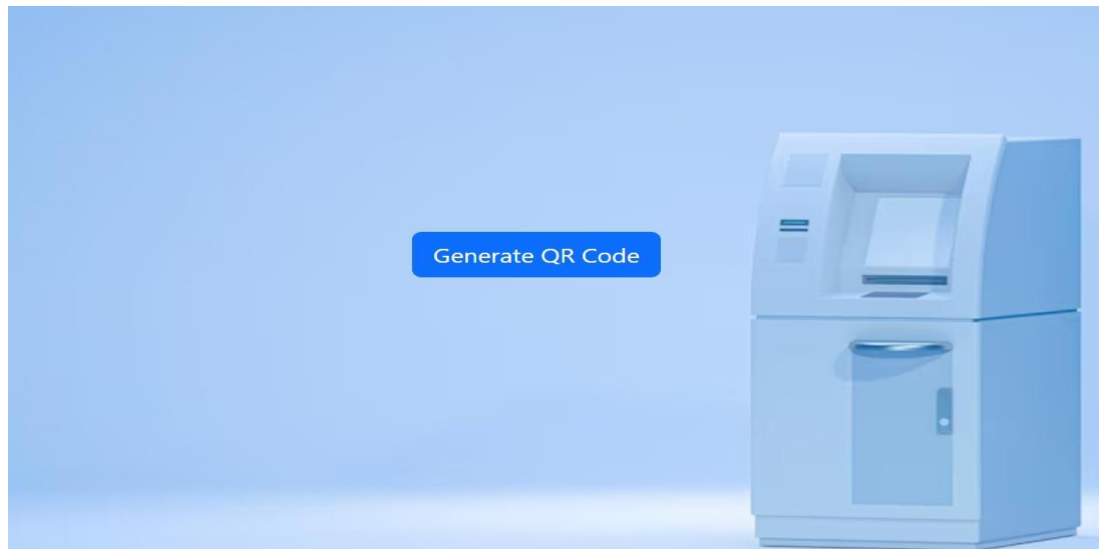
Fast Cash Page:



Screenshot-4

This is Fast Cash page in which customer can direct select which note he/she want and how much.

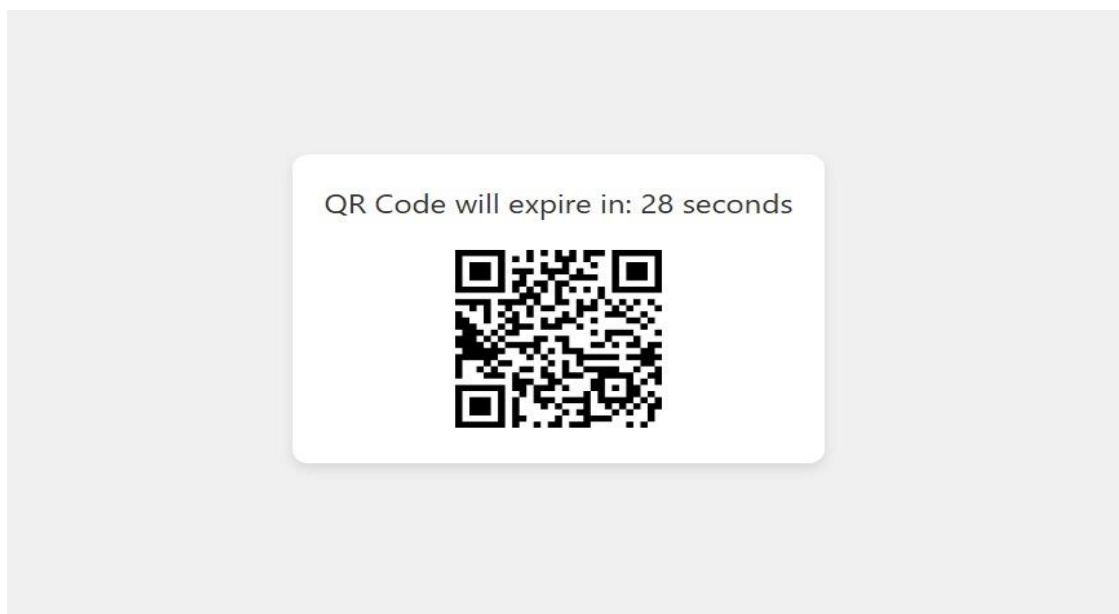
Generate QR Code Page:



Screenshot-5

This Page having Generate QR Code button customer can click on this button and can generate QR code

QR Code Page :



Screenshot-6

This Page show the QR Code customer can scan this code for payment.

Coding:

Functions:

The Atm Simulator is built on the Model-View-Controller (MVC) architectural pattern, which provides a clear separation of concerns and promotes modularity, maintainability, and scalability. The system architecture consists of three interconnected layers:

1. Model:

- The Model layer represents the application's data and business logic.
- The Model includes entities such as User, Transaction, Account, and Bank Database. These entities encapsulate data-related operations.
- Business logic related to User authentication and validation. Processing banking transactions such as withdrawals, transfers, balance inquiries, and mini statements.
- Processing banking transactions such as withdrawals, transfers, balance inquiries, and mini statements.

2. View:

- The View layer is responsible for presenting the user interface to the end-user.
- This layer includes the HTML templates, CSS stylesheets, and JavaScript (React JS) code for rendering the user interface.
- User interfaces such as the home page, Menu page, withdraw page, Transfer, Mini-Statement, Fast Cash, Check Balance are part of the View layer.

2. Controller:

- The Controller layer acts as an intermediary between the Model and View layers, handling user requests and orchestrating the flow of data.

- Receives incoming HTTP requests from the View layer, such as user login, withdrawal requests, or balance inquiries.
 - Invokes appropriate business logic in the Model layer, such as validating user credentials or processing a transaction.
 - Determines which view to render in response to a user request, such as showing the transaction result or displaying an error message.
 - Controllers manage the navigation logic, ensuring users are directed to the appropriate page based on their actions, such as moving from the login page to the dashboard after successful authentication.

Main Component of Backend:

Entry Point:

API Gateway:

Atm Controller :

```
package com.atm.controller;
```

```
import java.util.List;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.PathVariable;
```

```
import org.springframework.web.bind.annotation.PostMapping;
```

```
import org.springframework.web.bind.annotation.RequestBody;
```

```
import org.springframework.web.bind.annotation.RestController;
```

```

import com.atm.entities.Atm;
import com.atm.entities.Withdraw;
import com.atm.serviceinterface.AtmServiceInterface;

@RestController
public class AtmController
{
    @Autowired
    private AtmServiceInterface atmServiceRef;

    @GetMapping("/Getatm")
    public List<Atm> getAllAtm()
    {
        return atmServiceRef.atmDetails();
    }

    @PostMapping("/Insertatm/details")
    public void insertDetails(@RequestBody Atm details)
    {
        atmServiceRef.insertAtmDetails(details);
        System.out.println("Inserted Successful" + details);
    }

    @PostMapping("/Insertatm/atmDetails")
    public void insertArrayOfAtmDetails(@RequestBody List<Atm> atmDetails)
    {
        atmServiceRef.insertArrayOfAtmDetails(atmDetails);
        System.out.println("Inserted records successfully.");
    }

    @GetMapping("/Getatm/{id}")
    public Atm findAtmById(@PathVariable int id)

```

```

    {
        return atmServiceRef.findAtmById(id);
    }

    @GetMapping("/Getatm/{name}")
    public Atm findAtmByName(@PathVariable String name)
    {
        return atmServiceRef.findAtmByName(name);
    }

    @PostMapping("/Deleteatm/{id}")
    public void deleteAtmById(@PathVariable int id)
    {
        atmServiceRef.deleteAtmById(id);
    }

    @PostMapping("/Updateatm/atmDetails")
    public void updateAtm(@RequestBody Atm atmDetail)
    {
        atmServiceRef.updateAtm(atmDetail);
    }

    @PostMapping("/Updatearrayatm/listAtmDetails")
    public void updateArrayAtm(@RequestBody List<Atm> listAtmDetail)
    {
        atmServiceRef.updateArrayAtm(listAtmDetail);
    }

    // @PostMapping("/withdraw")
    // public void withdrawAmount(@RequestBody Withdraw details)
    // {
    //     atmServiceRef.withdrawAmount(details);
    // }

```

```
}
```

Customer Controller :

```
package com.atm.controller;
```

```
import java.util.List;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.web.bind.annotation.CrossOrigin;
```

```
import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.PathVariable;
```

```
import org.springframework.web.bind.annotation.PostMapping;
```

```
import org.springframework.web.bind.annotation.RequestBody;
```

```
import org.springframework.web.bind.annotation.RequestParam;
```

```
import org.springframework.web.bind.annotation.ResponseBody;
```

```
import org.springframework.web.bind.annotation.RestController;
```

```
import com.atm.entities.Card;
```

```
import com.atm.entities.Customer;
```

```
import com.atm.entities.Transfer;
```

```
import com.atm.serviceinterface.CustomerServiceInterface;
```

```
@RestController
```

```
@CrossOrigin(origins="https://localhost:3001")
```

```
public class CustomerController {
```

```
    @Autowired
```

```
    private CustomerServiceInterface CustomerServiceRef;
```

```
    @GetMapping("/Getcustomer")
```

```
    public List<Customer> getAllCustomer()
```

```
{  
    return CustomerServiceRef.CustomerDetails();  
}
```

```
@PostMapping("/InsertCustomer/details")  
public void insertDetails(@RequestBody Customer details)  
{  
    CustomerServiceRef.insertCustomerDetails(details);  
    System.out.println("Inserted Successful" + details);  
}
```

```
@PostMapping("/InsertCustomer/CustomerDetails")  
public void insertArrayOfCustomerDetails(@RequestBody List<Customer> CustomerDetails)  
{  
    CustomerServiceRef.insertArrayOfCustomerDetails(CustomerDetails);  
    System.out.println("Inserted records successfully.");  
}
```

```
@GetMapping("/findbyid/{id}")  
public Customer findCustomerById(@PathVariable int id)  
{  
    return CustomerServiceRef.findCustomerById(id);  
}
```

```
@GetMapping("/findbyname/{name}")  
public Customer findCustomerByName(@PathVariable String name)  
{  
    return CustomerServiceRef.findCustomerByName(name);  
}
```

```
@PostMapping("/DeleteCustomer/{id}")  
public void deleteCustomerById(@PathVariable int id)  
{
```

```

        CustomerServiceRef.deleteCustomerById(id);
    }

    @PostMapping("/UpdateCustomer/CustomerDetails")
    public void updateCustomer(@RequestBody Customer CustomerDetail)
    {
        CustomerServiceRef.updateCustomer(CustomerDetail);
    }

    @PostMapping("/UpdatearrayCustomer/listCustomerDetails")
    public void updateArrayCustomer(@RequestBody List<Customer> listCustomerDetail)
    {
        CustomerServiceRef.updateArrayCustomer(listCustomerDetail) ;
    }

    @PostMapping("/Login")
    public Customer checkAuthorization(@RequestBody Card details) {
        System.out.println(details);
        return CustomerServiceRef.checkCardNoAndPin(details);
    }

    @PostMapping("/moneytransfer/accounts/details")
    public String accTransfer(@RequestBody Transfer details) {
        System.out.println(details);
        return CustomerServiceRef.accTransfer(details);
    }

    @PostMapping("/moneytransfer/upi/details")
    public String upiTransfer(@RequestBody Transfer details) {
        return CustomerServiceRef.upiTransfer(details);
    }

    @GetMapping("/p/{custId}")
    public double balance(@PathVariable("custId")int custId)
    {

```

```

        System.out.println(custId);
        return CustomerServiceRef.balance(custId);
    }

}

```

Mini Statement Controller :

```

package com.atm.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;

import com.atm.entities.MiniStatement;
import com.atm.services.MinistatementService;

@RestController
@CrossOrigin(origins="http://localhost:3000")
public class MinistatementController {

    @Autowired
    MinistatementService ministatmentref;

    @GetMapping("/ministatement/{id}")
    public List<MiniStatement> ministatementRequest(@PathVariable int id) {
        return ministatmentref.sendMinistatementList(id);
    }
}

```


Fast Cash Controller :

```
package com.atm.controller;
```

```
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.CrossOrigin;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RestController;
```

```
import com.atm.entities.Withdraw;  
import com.atm.entities.WithdrawResponse;
```

```
import com.atm.services.FastCashService;  
import com.atm.services.WithdrawService;
```

```
@RestController
```

```
@CrossOrigin(origins="http://localhost:3000")
```

```
public class FastCashController {
```

```
    @Autowired
```

```
    FastCashService fastCash;
```

```
    @PostMapping("/fastcashwithdraw")
```

```
    public WithdrawResponse withdrawVerify(@RequestBody Withdraw obj) {
```

```
        System.out.println(obj);
```

```
        return fastCash.accessRequets(obj);
```

```
    }
```

```
}
```

QrGenApi Controller :

```
package com.atm.controller;
```

```
import java.util.UUID;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.http.ResponseEntity;
```

```
import org.springframework.web.bind.annotation.*;
```

```
import com.atm.entities.QrGenApi;
```

```
import com.atm.entities.QrGenApiDto;
```

```
import com.atm.entities.QrGenApiRequest;
```

```
import com.atm.services.QrGenApiService;
```

```
@RestController
```

```
@CrossOrigin(origins="http://localhost:3000")
```

```
public class QrGenApiController {
```

```
    @Autowired
```

```
    private QrGenApiService qrGenApiService;
```

```
    @PostMapping("/save")
```

```
    public ResponseEntity<QrGenApiDto> saveTransaction(@RequestBody QrGenApiRequest request) {
```

```
        String tranId = UUID.randomUUID().toString().substring(0, 8);
```

```
        qrGenApiService.saveTransaction(tranId, request.getAtmId(), request.getAmount());
```

```
        QrGenApiDto qrGenApiQ = qrGenApiService.getLatestTransaction();
```

```
        return ResponseEntity.ok(qrGenApiQ);
```

```
    }
```

```
@GetMapping("/latest")
public ResponseEntity<QrGenApiDto> getLatestTransaction() {
    QrGenApiDto qrGenApiDto = qrGenApiService.getLatestTransaction();
    if (qrGenApiDto != null) {
        return ResponseEntity.ok(qrGenApiDto);
    } else {
        return ResponseEntity.notFound().build();
    }
}
}
```

Transaction Controller :

```
package com.atm.controller;

import java.util.List;
import java.util.UUID;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import com.atm.entities.Atm;
import com.atm.entities.Check;
import com.atm.entities.QRRequest;
import com.atm.entities.Transaction;
import com.atm.services.TransactionService;

@RestController
@CrossOrigin(origins={"*", "https://localhost:3001"})
public class TransactionController {

    @Autowired
    private final TransactionService transactionService;

    public TransactionController(TransactionService transactionService) {
        this.transactionService = transactionService;
    }
}
```

```

public void insertTransactionDetails(@RequestBody Transaction details)
{
    transactionService.saveTransaction(details);
    System.out.println("Inserted Successful" + details);
}

```

```

@GetMapping("/api/Getalltranc")
public List<Transaction> getAlltranc()
{
    return transactionService.FindAlltranc();
}

```

```

@PostMapping("/api/checkstatus")
public String checkStatus(@RequestBody Check transactionId)
{
    System.out.println(transactionId);
    return transactionService.checkStatus(transactionId);
}

```

```

@PostMapping("/api/submit-transaction-cardless")
public ResponseEntity<String> submitTransaction(@RequestBody QRRequest qrRequest) {
    System.out.println("Received data: " + qrRequest);

    //Check if values are non-zero before saving
    if (qrRequest.getAmount() <= 0 || qrRequest.getAtmId() <= 0 || qrRequest.getCustomerId() <= 0 ) {
        System.out.println("Invalid values - Amount: " + qrRequest.getAmount() + ", ATM ID: " +
qrRequest.getAtmId() + ", Customer ID: " + qrRequest.getCustomerId());
    }
}

```

```

        return new ResponseEntity<>("Invalid data received", HttpStatus.BAD_REQUEST);
    }

    try {
        // Autogenerate transaction ID if not already set
        if (qrRequest.getTranId() == null || qrRequest.getTranId().isEmpty()) {
            qrRequest.setTranId(UUID.randomUUID().toString().substring(0, 8));
        }

        transactionService.saveCardlessTransaction(
            qrRequest.getTranId(),
            qrRequest.getAmount(),
            qrRequest.getCustomerId(),
            qrRequest.getAtmId(),
            "DEBIT", // Transaction type
            "Completed", // Transaction status
            "Success" // PSP status
        );

        transactionService.updatecardlesscustomerbankbalance(qrRequest.getTranId(),
            qrRequest.getAmount());

        String responseMessage = String.format(
            "Transaction saved successfully. Amount: %.2f, CustomerId: %d, AtmId: %d",
            qrRequest.getAmount(),
            qrRequest.getCustomerId(),
            qrRequest.getAtmId()
        );

        return new ResponseEntity<>(responseMessage, HttpStatus.OK);
    } catch (Exception e) {
        return new ResponseEntity<>("Error: " + e.getMessage(),
            HttpStatus.INTERNAL_SERVER_ERROR);
    }

```

```
    }  
}  
  
// @PostMapping("/api/submit-transaction-updateamount")  
// public void updateAmount(String transid ,double amount) {  
//     transactionService.updatecardlesscustomerbankbalance(transid , amount);  
// }  
  
}
```

FUTURE SCOPE

“The future of the ATM Simulator Project includes adding mobile app support, allowing users to make transactions on their phones. We could also introduce voice commands for easier use and enhance security with features like fingerprint or facial recognition. Expanding to handle multiple currencies and integrating with other financial services, like loans or bill payments, are other possibilities. Additionally, incorporating AI could help detect fraud and offer a more personalized user experience”.

CONCLUSION

“The ATM Simulator Project successfully demonstrates how modern banking operations can be simulated in a controlled, virtual environment. By integrating key functionalities like user authentication, cash withdrawal, balance inquiry, and fund transfer, the project offers a realistic and secure platform for users to perform transactions. The use of both card and cardless methods provides flexibility, catering to different user preferences.

The project’s foundation on the Model-View-Controller (MVC) architectural pattern ensures that it is modular, maintainable, and scalable, making it easier to expand and enhance in the future. As technology evolves, the ATM Simulator can be further developed to include advanced features such as mobile integration, AI-driven fraud detection, and multi-currency support, ensuring it remains relevant and useful. Overall, this project lays the groundwork for a robust and versatile ATM simulation tool that can be adapted to meet the changing needs of users and the financial industry”.

Bibliography:

We kept window open for further additional development.

- <https://react.dev/>
- <https://spring.io/projects/spring-boot>
- <https://www.w3schools.com/>