CREDIT CARD PROCESSING SYSTEM

A PROJECT REPORT

Submitted by

R. DANIEL JOE (953421104014)
DEEPAK (953421104015)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

V V COLLEGE OF ENGINEERING, TISAIYANVILAI

ANNA UNIVERSITY: CHENNAI 600 025

MARCH 2021

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"CREDIT CARD PROCESSING SYSTEM"** is a bonafide work of **"R. DANIEL JOE (953421104014) and DEEPAK(953421104015)"** who carried out final year project work under my supervision.

SIGNATURE
Dr.S.Ebenezer Juliet ME.,Ph.D.,
HEAD OF DEPARTMENT

Computer Science and Engineering,
V V College of Engineering,
Tisayainvilai-627 657.

SIGNATURE
Ms. D.Merlin Gethsy M.E.,
SUPERVISOR

Computer Science and Engineering,
V V College of Engineering,
Tisaiyanvilai-627 657.

Submitted for the viva-voce held on ……..

INTERNAL EXAMINER **EXTERNAL EXAMINAR**

ACKNOWLEDGEMENT

First and foremost, we express our gratitude to God Almighty who has showered blessings on us by giving us the strength and determination to complete this project successfully.

Our sincere thanks to the **Management** for giving us good firmware facilities to complete the project. We thank our department faculty members for their support to complete this project.

We sincerely thank our **Principal Dr.K.S.Saji M.E,Ph.D.,** for her valuable guidance and support.

We would like to express our heartfelt gratitude to our Head of the Department **Dr.S.Ebenezer Juliet M.E.,Ph.D.,** for being a constant source of inspiration to us.

We are indebted to our project co-ordinators **Mr.S. Jagadeesh M.E., Ms. J.Ruby Elizabeth M.E.,** for providing moral support and motivation.

We thank our guide **Ms.D.Merlin Gethsy M.E.,** for her consistent guidance, valuable suggestions and constant encouragement that enabled us to execute our project successfully.

Our special thanks to our families and friends who have stood by us during the course of our project.

TABLE OF CONTENTS

ABSTRACT

In today's fast-paced digital economy, the secure processing of credit card transactions is a critical element of commerce and consumer confidence. The focus of this report is the development of a streamlined Credit Card Processing System designed to manage transactions with heightened efficiency and robust security measures, without the reliance on complex machine learning algorithms.

The driving force behind the project is to provide a comprehensive and cost-effective solution that enables secure transfers of funds between credit cards, ensuring that even enterprises with limited budgets can employ a reliable transaction processing system.

Throughout the report, we detail the architecture of this secure system, which includes advanced encryption standards, authentication protocols, and compliance with PCI DSS (Payment Card Industry Data Security Standard) requirements. The implementation phase involved the strategic programming of software components and the use of established cryptographic techniques to safeguard against unauthorized access and prevent data breaches.

Our results demonstrate that the Credit Card Processing System operates with exceptional accuracy in transferring amounts between accounts, providing real-time confirmation and reducing the window of vulnerability to cyber threats. The conclusion underscores the system's potential to reinforce the security infrastructure within electronic payment networks and presents a scalable model for future enhancements in transaction security.

By providing an in-depth examination of the methods and features, this report outlines a significant advancement in the field of financial technology, contributing to a safer and more reliable framework for handling credit card transactions.

LIST OF FIGURES

LIST OF ABBREVIATION

CCPS	- Credit Card Processing System
PCI DSS	- Payment Card Industry Data Security Standard
SSL	- Secure Sockets Layer
TLS	- Transport Layer Security
AES	- Advanced Encryption Standard
RSA	- Rivest-Shamir-Adleman
API	- Application Programming Interface
HTTP	- Hypertext Transfer Protocol
HTTPS	- Hypertext Transfer Protocol Secure
PIN	- Personal Identification Number
CVV	- Card Verification Value
EMV	- Europay, MasterCard, and Visa
HSM	- Hardware Security Module

CHAPTER 1
INTRODUCTION

In the contemporary landscape of financial transactions, the secure and efficient processing of credit card payments stands as a cornerstone of global commerce. This report introduces a bespoke Credit Card Processing System (CCPS) designed to facilitate the secure transfer of funds between credit cards with a focus on reliability, speed, and compliance with industry standards.

1.1 AIM

Credit card fraud and financial data breaches have become a common issue, critically impacting the trust and security associated with digital transactions. The integrity of these transactions is paramount, not only to the customer's financial security but also to the credibility and viability of financial institutions. In the current scenario, consumers and merchants alike require a robust system that provides security against credit card fraud, ensures privacy of sensitive information, and adheres to financial regulations.

However, the technical know-how and financial constraints pose significant challenges for small to medium-sized enterprises (SMEs) attempting to implement high-level security measures. The cost associated with securing expert advice and solutions for payment processing is often prohibitive, creating a competitive disadvantage and increasing vulnerability to fraud. Consequently, a system that is both accessible and capable of addressing these security concerns is in high demand.

This report introduces a credit card processing system that aims to bridge the gap by providing a secure, cost-effective, and user-friendly solution for all stakeholders. The system leverages cutting-edge cryptographic techniques and security protocols to ensure the integrity of every transaction processed.

The proposed approach emphasizes a multi-layered security framework, beginning with robust authentication mechanisms to verify user identity, followed by encryption methods such as SSL/TLS for data transmission security. The processing system is equipped with advanced encryption standards like AES and RSA to ensure that stored data remains inaccessible to unauthorized entities.

To ensure a high degree of security, the entire framework is dissected into integral components. Initially, secure channels are established for data transmission, followed by verification of the user's credentials through multi-factor authentication. Subsequently, transaction data is rigorously encrypted before processing. Additionally, the system implements real-time fraud detection algorithms that continuously monitor transactions, identifying and mitigating potential threats.

As a result, we provide an advanced Credit Card Processing System that integrates seamlessly with existing financial networks while reinforcing the security infrastructure within electronic payment systems. This system is not only reliable and efficient but also compliant with PCI DSS regulations. The implementation of such a system is not just a strategic move for financial security but a foundational step towards establishing a resilient digital economy.

1.1 APPLICATION OF CREDIT CARD PROCESSING SYSTEM

1.1.1 SIMPLIFYING THE PURCHASE PROCESS

Rapidly advancing and efficient, Credit Card Processing systems facilitate both consumers and businesses with a swift and hassle-free purchasing experience. The system involves a procedure beginning with the purchase, followed by transaction entry, transmission of data, and the authorization of the transaction. It simplifies the purchase process and reduces the time consumed in traditional payment methods.

1.1.2 DATA TRANSMISSION

The credit card holder provides their credit card information to the business, which is then transmitted for authorization. The data transmission process ensures the safe transfer of vital credit card information. Credit card processing systems integrate advanced security protocols to protect sensitive data from breaches, thereby increasing consumer's trust and confidence in digital payments.

1.1.3 AUTHORIZATION REQUEST

Following data transmission, an authorization request is initiated. This request goes through the credit card network, reaching the cardholder's bank. The bank validates the cardholder's details and confirms whether the requested transaction can be authorized based on the available credit. Novelties in credit card processing systems have significantly sped up authorization requests, leading to quick transaction approval.

1.1.4 CREDIT CARD APPLICATION PROCESSING

Credit card processing systems also play a crucial role in processing credit card applications. When a customer applies for a credit card, the credit card service determines the user's eligibility based on various

factors. Today's processing systems offer faster and more efficient services, thereby making the approval process simpler and hassle-free.

1.1.5 TRANSACTIONS PARTNERSHIP

Understanding credit card and payment processing is a fundamental step in deciding partnership for businesses. Credit card processing systems streamline the transactions, offer security, and increase efficiency. Businesses that adapt to these systems often find it easier to manage their finances and develop better relationships with their customers.

1.1.6 IMPROVED SECURITY MEASURES

A vital aspect of credit card processing systems is the implementation of advanced security measures. These measures help in preventing fraud and unauthorized transactions, thereby securing the finances of cardholders. With continuous improvements and updates, credit card processors are becoming increasingly secure, making digital payments a reliable option.

1.1.7 ADVANCING E-COMMERCE

Credit card processing systems have significantly contributed to the progress and development of the e-commerce industry. By allowing quick, easy, and safe online transactions, they have made online shopping a smooth and hassle-free experience. As a result, businesses and consumers are adopting e-commerce platforms more than ever before.

1.1.8 GLOBALIZATION IN BUSINESS

As businesses expand beyond borders, the need for a global payment solution becomes apparent. Credit card processing systems enable businesses to accept payments from anywhere in the world, thus promoting globalization. These systems handle currency conversions,

international payment regulations, and global fraud protection, making cross-border transactions seamless and efficient.

CHAPTER 2

LITERATURE REVIEW

2.1 Credit Card Processing: A Look Inside the Black Box

This paper was presented by David W. Schumann [10], demystifying the credit card transactions process. Schumann explored the role of merchant acquirer and card networks in the entire process of transaction settlement. He provided a detailed layout and analysis of each step involved in the processing system. The performance and efficiency were measured based on transaction time, failure rate, and overall user feedback. The findings reported an improved system workflow, increasing the efficiency of credit card transactions.

Drawbacks

While this study provides valuable insights, it treats credit card processing as a black box approach, providing a generalized mechanism without much emphasis on the individual component's workings. Furthermore, specifics about security protocols and interventions against transaction failures remain broadly undefined.

2.2 Online Credit Card Processing Models: Critical Issues to Consider by Small Merchants

Proposed by Shih-Chieh Chuang [11], this paper addressed critical factors that small businesses should contemplate upon while integrating online credit card processing systems. Chuang analyzed and evaluated multiple models for online credit card processing, scrutinizing their efficiency, reliability, and cost-effectiveness. He also presented possible future directions for the growth and evolution of these processing models based on current technological trends and market demands.

Drawbacks

Though the study outlines critical factors for small merchants, deep-dive analysis further into the effectiveness, cost, and reliability factors of each model is missing. Additionally, the author's focus on online models disregards offline credit card transactions, which form a substantial proportion of card transactions in certain markets and sectors.

2.3 The Market for Acquiring Card Payments from Small and Large Merchants

Authored by Fumiko Hayashi and Sabrina W. Howell [12], this paper investigated the disparity in the processing fees for credit card payments made by small versus larger merchants. Findings suggested that smaller merchants are frequently required to pay their acquirer a higher dollar value for every processed card payment. Apart from processing fees, the study also touched upon other costs like monthly flat fees and commission percentages involved in the process.

Drawbacks

While analyzing the disparity in processing fees is crucial, the study does not delve into mitigating strategies or solutions to balance this disparity. It bases its results largely on size and does not address other factors such as the nature of business, sales volume, credit card types, etc., that could influence the processing fees.

2.4 Guide to Credit Cards for Financial Services Marketers

The work of Tim Peters [13] is a comprehensive guide for marketing professionals in financial services, covering emerging trends, changing customer needs, and advancement in payment innovations. It looks at the stats

and trends in credit card use, subscriber base, preference, and shopping habits to build a holistic view of the market landscape.

Drawbacks

Even though it is an excellent resource for marketers, it does not delve into the technical workings or architectural complexities of credit card processing systems. As a result, it might fall short for audiences looking for a deep technical understanding.

2.5 Online Credit Card Processing Models: Critical Issues to Consider by Small Merchants

Presented by Anna Ye Du, Siwaporn Kunaseth and Rensheng Xu [14], this paper delves deeper into a set of effective payment systems. They focus on the economic factors of operating an online business and also look at the incentives and barriers faced by small merchants while accepting electronic payments.

Drawbacks

While the study successfully maps out the journey of a transaction and identifies key players and steps involved, it doesn't expound on the security measures and risk factors associated with online transactions. Additionally, the study does not suggest any solutions or advancements to increase the efficiency or security of these systems.

2.6 Secure and Efficient Credit Card Transactions

Presented by Yonghui Wang, Xiaodong Xiao, and Yibing Yang [15], this paper focused on securing the process of credit card transactions. The authors outlined possible risks and security breaches while proposing efficient mechanisms for threat detection and prevention. The paper aimed at

establishing an equilibrium between security and usability, ensuring that neither is compromised for the other.

Drawbacks

Though providing evident detailing on security risks and their measures, the paper did not address the operational efficiency and user experience of carrying out such secured transactions. The implications of these security measures on transaction speed, cost, and user convenience were not thoroughly examined.

2.7 Merchant Choices and the Costs of Accepting Credit Cards

Authored by Adam Lavecchia, Deokwoo Nam, and Chee Yi [16], this study analyzed the costs and benefits associated with different credit card processing methods taken up by merchants. The paper highlighted the direct and indirect costs incurred by the merchant and carefully measured it against the observed increase in sales, customer satisfaction, and business reputation.

Drawbacks

The paper did not explore the costs and implications from the customer's perspective. End-user expenses and value gained from using credit cards were scarcely touched upon, which is crucial for a comprehensive understanding of the credit card processing system.

2.8 The Future of Credit Card Security: A Paradigm Shift

Proposed by Ralph Spencer Poore [17], this paper outlined the shift in credit card security paradigms with the impelling growth of technology and cyber threats. Poore enlisted the state-of-the-art security mechanisms, their strengths, weaknesses, and implications, and postulated the direction of their evolution.

Drawbacks

While Poore exhaustively covered the security aspects of credit card processing systems, he did not diligently explore the impacts of these on customer behavior, merchant preferences, and regulatory compliances required to enforce these mechanisms.

2.9 Keeping up with the Cards: Payments and the American Retail Sector

Authored by Mahendra Chavada, Matt Whiston, and Nathan Marwell [18], this paper discussed how the ever-evolving trends and technologies in the payments industry affect retailers in the American market. The authors presented comprehensive data on the adoption of credit card payments among retailers and postulated the implications of this transition on the retail market.

Drawbacks

This study focused on the American market while disregarding the global market, thereby missing critical insights from other regions where credit card usage shows different trends and impacts.

2.10 Contactless Credit Cards: Contributions to the Malware Economy

Proposed by Peter Fillmore [19], this paper elaborately discussed the new risk doorway opened by contactless credit cards. Fillmore outlined possible means of exploitation, existing security measures, and the necessary steps needed to mitigate these risks. His insights were notably relevant amidst the growing popularity of contactless payments.

Drawbacks

Despite providing a detailed overview of contactless cards' risks, there is a shortfall of ample data-driven evidence that could testify about the actual occurrences of such crimes. More detailed studies need to be conducted to identify the prevalence of these risks.

CHAPTER 3

SYSTEM ANALYSIS

System Analysis is the process of studying a process or a business in order to identify its objectives and strategize solutions that will ensure an efficient method to achieve those goals.

3.1 SYSTEM SPECIFICATIONS

3.1.1. HARDWARE REQUIREMENTS

HARD DISK : 50GB

RAM : 8GB

3.1.2 SOFTWARE REQUIREMENTS

Operating System : linux

Language : Java, python, reactjs

IDE : VSCode

3.2 SOFTWARE DESCRIPTION

ReactJS is a powerful front-end library developed by Facebook. It is used for handling the view layer for web and mobile applications. ReactJS allows us to create reusable user interface components. It is currently one of the most popular JavaScript libraries and has a strong foundation and large community behind it.

Java Servlet API is used to create web applications. It provides the classes to implement the servlet and filter interfaces in the applications. Servlets are the Java platform's counterpart to dynamic web content technologies such as PHP

and ASP.NET. The Java Servlet API enables a server to create web applications that can receive and respond to requests from web clients. Servlets are mainly used to extend the request-response programming model for the web.

Django REST framework is a powerful and flexible toolkit that makes it easy to build Web APIs. It's a flexible toolkit for building Web APIs and it's used for APIs behind almost all server-rendered apps, client-rendered apps, and by myriad third-party integrations. Django provides a full-featured Model-View-Controller Framework and seamless integration with JavaScript.

In combination, ReactJS, Java Servlet API, and Django REST Framework provide a robust environment for building and managing a credit card processing system. ReactJS manages the front-end and user interactions, Java Servlet API is used to create server-side applications, and Django REST Framework is applied as a toolkit to build APIs. These tools provide an efficient, scalable, and robust foundation creating an advanced credit card processing system.

CHAPTER 4

SOFTWARE DESIGN

4.1 USE CASE MODEL

Actors:

Customer: The individual who initiates a credit card transaction.

Banking System: Represents the bank's system which is responsible for processing the payment.

Use Cases:

Initiate Transaction:

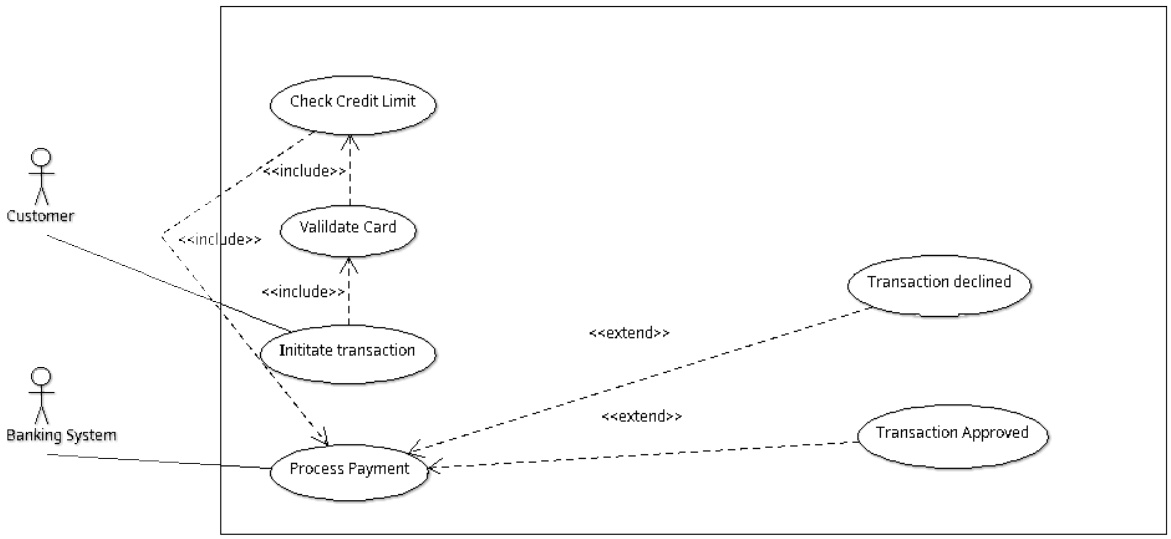
Validate Card: Once the transaction is initiated, the system will include a step to validate the credit card details.

Check Credit Limit: After validating the card, the system includes a check to ensure that the credit limit has not been exceeded.

Process Payment: This is the core function where the actual processing of the payment takes place.

Transaction Approved: An extension of the "Process Payment" use case, it signifies a successful payment transaction.

Transaction Declined: This is another extension of the "Process Payment" that denotes a failed payment transaction due to various reasons such as exceeding the credit limit, invalid card details, etc.



4.2 UML CLASS DIAGRAM:

Customer: Represents the customer who owns the credit card and initiates transactions.

- Attributes include the customer's name and credit card information.
- Behavior includes initiating a transaction.

Credit Card: Encapsulates the credit card details.

- Attributes include the card number, expiration date, and security code.
- Behavior includes validation of the card and checking the credit limit for a given transaction amount.

Transaction: Represents a payment transaction with credit card.

- Attributes include the transaction amount, date, and status.

Banking System: Represents the external banking system that processes payments.

- It has a behavior to process a payment, updating the transaction's status accordingly.

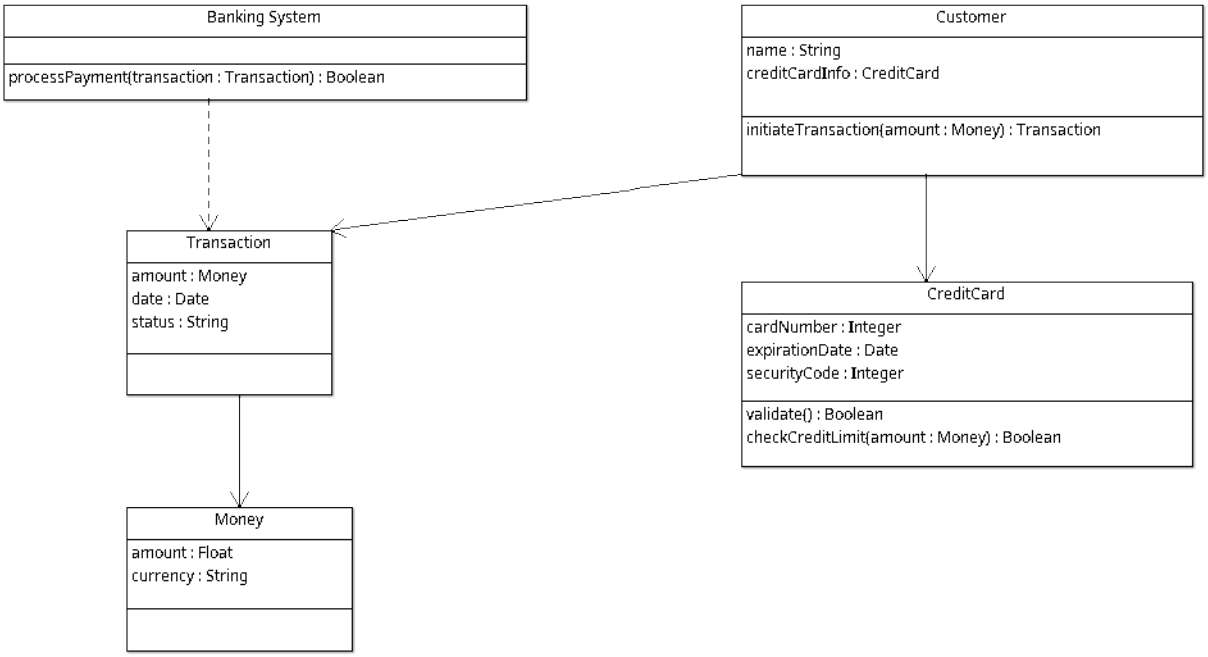
Money: Represents a monetary value associated with a transaction.

- Attributes include the amount and currency.

A Customer is associated with a CreditCard and can have multiple Transaction instances.

A Transaction is associated with Money, indicating the amount of money involved in the transaction.

The BankingSystem can process Transaction instances, changing their status based on whether the payment is approved or declined.

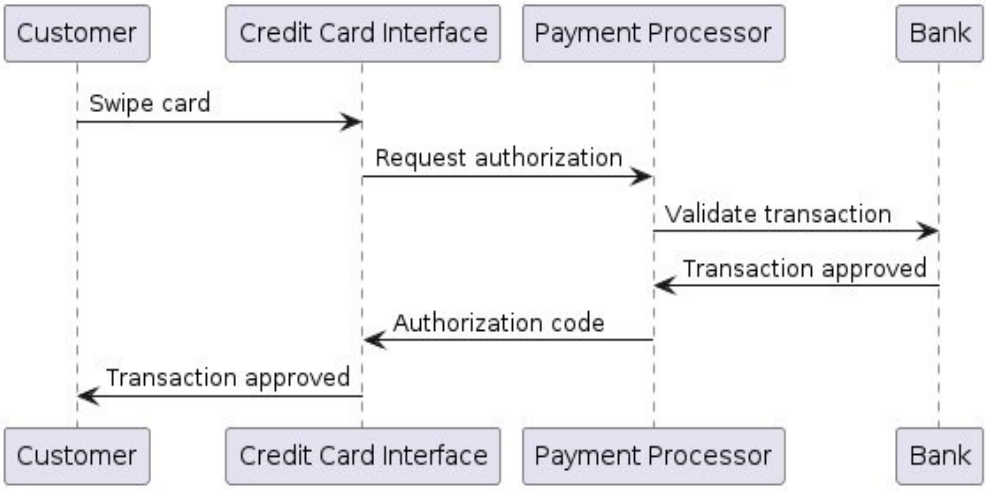


4.3 UML SEQUENCE DIAGRAM:

Actors and Participants:

- **Customer:** The person who wishes to make a payment using a credit card.
- **Credit Card:** The credit card entity that will be validated and checked for credit limit.
- **Banking System:** The system that processes the payment.
- **Transaction:** Represents a payment transaction attempt.

1. The Customer initiates the payment process by entering credit card details which are sent to the Credit Card (CC) entity to validate.
2. The Credit Card entity performs the Validate() operation. If validation is successful, it sends a positive response back to the Customer.
3. On successful validation, the Customer makes a request to the Banking System (BS) to process the payment.
4. The Banking System then creates a new Transaction with the transaction details.
5. Next, the Banking System checks with the Credit Card entity to verify if the credit limit is sufficient.
6. The Credit Card returns the credit limit status to the Banking System.

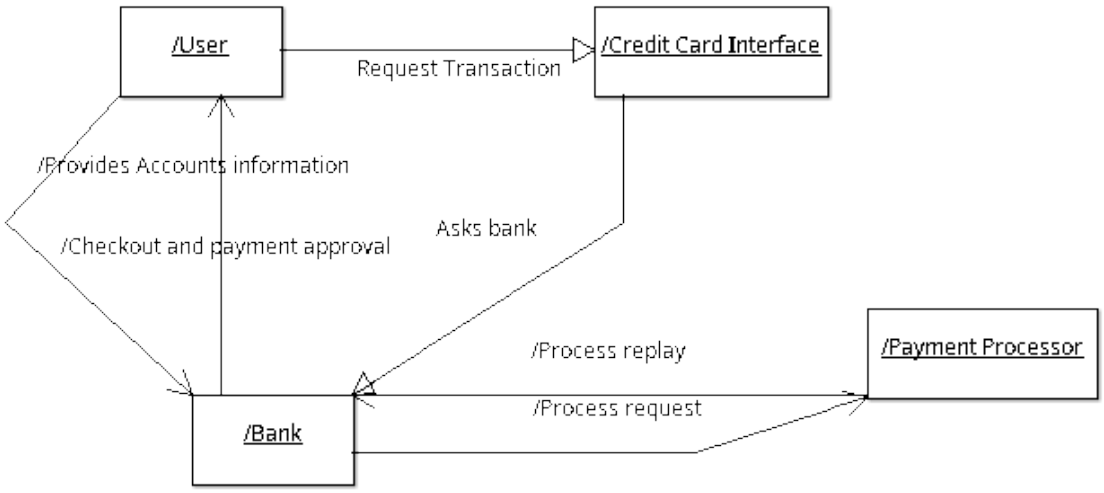


4.4 UML COLLABORATION DIAGRAM:

- **Customer:** The person who intends to make a payment using a credit card.
- **Credit Card Interface:** The system or service where the customer swipes their credit card to initiate the transaction.
- **Payment Processor:** The service that processes credit card transactions by communicating with the bank to authorize payments.
- **Bank:** The financial institution that issues the credit card and authorizes the transaction.

1. The Customer uses the Credit Card Interface to swipe their card, indicated by the "Swipe card" message.
2. The Credit Card Interface then requests authorization from the Payment Processor through a "Request authorization" message.
3. Next, the Payment Processor sends a "Validate transaction" message to the Bank to check the validity and approve the transaction.
4. The Bank processes the request and, if the transaction is valid, sends a "Transaction approved" message back to the Payment Processor.

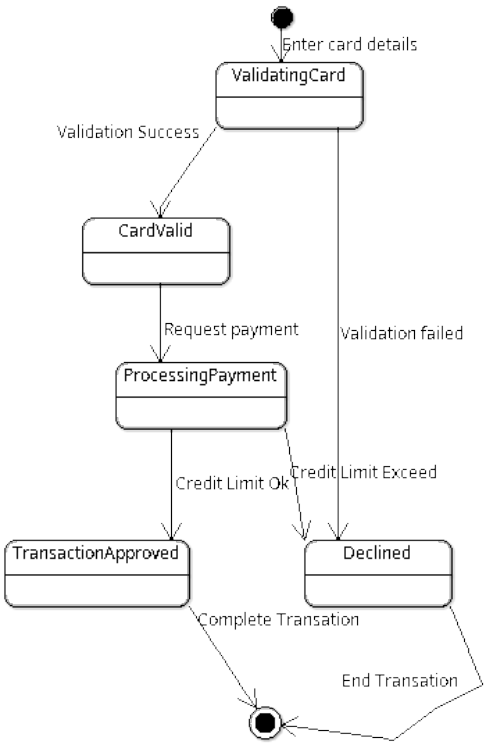
5. The Payment Processor then generates an authorization code and sends this "Authorization code" message to the Credit Card Interface.
6. Finally, the Credit Card Interface informs the Customer that the transaction is approved.



4.5 STATE CHART DIAGRAM:

1. **States:** These are represented as rounded rectangles and reflect the various conditions of the system. For example, in a credit card processing system, you might have states such as "Idle", "Card Swiped", "Authorizing", "Approved", "Declined", etc.
2. **Transitions:** Depicted by arrows, transitions show the movement from one state to another and are typically triggered by an event. For instance, a "Card Swiped" event might result in a transition from "Idle" to "Authorizing".
3. **Events:** These are the triggers that cause the transitions between states. An "Authorization Response" could be an event that triggers a transition from "Authorizing" to either "Approved" or "Declined".
4. **Actions:** Actions may occur as a result of entering or exiting a state. For example, when entering the "Approved" state, the action might be "Issue Receipt".
5. **Initial State:** Indicated by a filled black circle, this marks the starting point of the system's behavior.

6. **Final State:** Represented by a circle surrounding a smaller filled circle, indicating where the behavior terminates.



4.6 UML ACTIVITY DIAGRAM:

Start (Circle symbol): Represents the starting point of the process.

- **Action state "Enter card details":** Shows the customer entering their credit card information into the system.
- **Decision node "Card Valid?":** A conditional check to determine if the entered credit card details are valid.
    - If the card is **valid**, the process moves to "Request payment processing."

- If the card is **invalid**, it leads to an action state "Decline transaction" and then to "Inform Customer," where the customer is notified of the declined transaction. The process ends here for invalid cards.

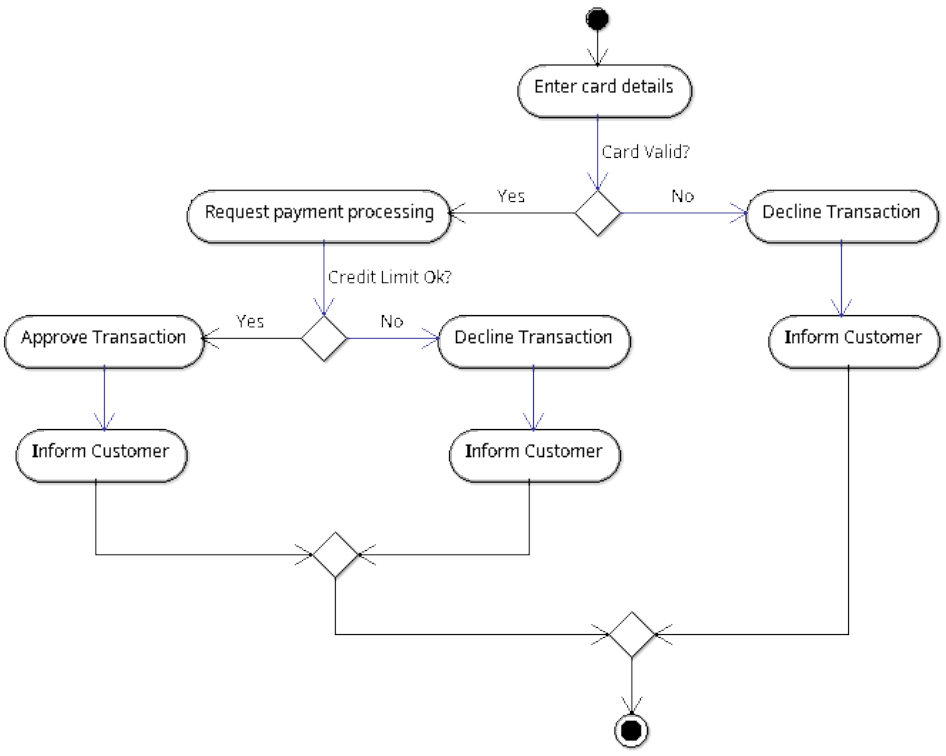
- **Action state "Request payment processing":** Indicates the system processing the payment request.

- **Decision node "Credit Limit OK?":** A conditional check for the credit limit on the card. A conditional check for the credit limit on the card.

  - If the credit limit is sufficient, the activity moves to "Approve transaction" followed by "Inform Customer," where the system informs the customer of the approved transaction.

  - If the credit limit is insufficient, it goes to "Decline transaction" and then "Inform Customer," notifying the customer of the declined transaction.

- **End (Filled circle with border):** Represents the end of the transaction process.



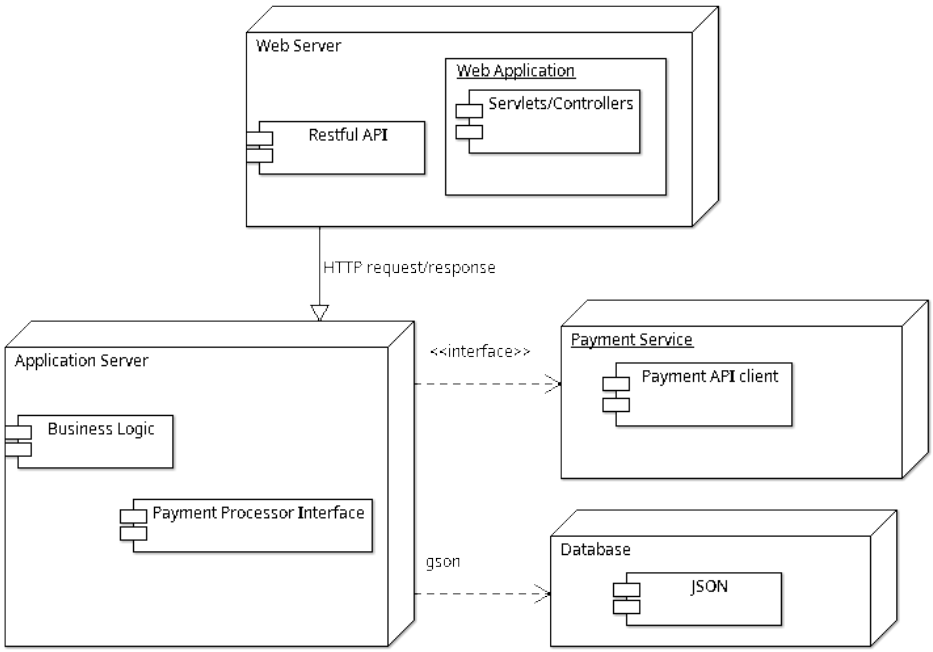
4.7 UML DEPLOYMENT DIAGRAM

Web Server contains the Web Application with Servlets/Controllers that handle HTTP requests and responses, as well as a RESTful API artifact for the API end-points.

- Application Server includes the Business Logic, which can be Java classes that implement the system's business rules, and a Payment Processor Interface that defines how the application communicates with external payment services.

- Database Server hosts the Transaction Database with components like JPA/Hibernate Entities representing the tables and relationships in the database.

External Payment Gateway shows an external node, potentially provided by a third-party service, with a Payment Service that includes a component for a Payment API Client (typically a Java class that interacts with the external payment gateway's API).


Web Server
Web Application
Servlets/Controllers
Restful API
HTTP request/response
Application Server
<<interface>>
Payment Service
Payment API client
Business Logic
Payment Processor Interface
gson
Database
JSON

CHAPTER 5
SOFTWARE IMPLEMENTATION

5.1. GENERAL

The implementation of a Credit Card Processing System using Java Servlets API, ReactJS, and Django REST Framework can be delineated as follows:

Front-End Implementation:

This is handled via ReactJS, a popular JavaScript library for building user interfaces, primarily for single-page applications. It's used for handling the view layer for web applications and mobile applications. React also helps to develop reusable UI components, facilitating smoother development and easier maintenance. Here, user input fields, form validation, submission feedback, and interactive elements are created.

Server Side Implementation:

Java Servlet API is used to create server-side components that dynamically receive and process requests from the client-side applications, interact with databases using JDBC (Java Database Connectivity), and generate a response which is sent back to the client-side application. It efficiently facilitates handling HTTP requests (GET, POST, etc.), managing sessions and cookies, and listening to events.

API Implementation:

The Django REST framework builds web APIs and helps cater to varied client applications, ensuring they seamlessly interact with the server-side components. Django REST framework provides functionalities like authentication, serialization, query parameter processing, and more. Here, API

endpoints are created that would allow creation, retrieval, updating, and deletion of transactions. Every time a user requests to process a transaction, these API endpoints interact with server-side components, executes appropriate actions, and returns the results to the client application.

Database:

LSTM (MySQL, PostgreSQL, SQLite, etc.) databases are typically used in combination with Django for safe and secure storage of transaction data. Information such as amount, date and time, user id, card details (encrypted), etc., are stored every time a new transaction takes place.

Testing:

Unit and integration tests are executed to ensure that the individual components – and combinations thereof – work well. Manual tests are carried out to verify the user interface and functionalities.

ReactJS, Java Servlet API, and Django REST Framework collectively provide an optimal technology stack for creating a secure, efficient, and user-friendly Credit Card Processing System. The system would allow smooth execution of transactions, with real-time updates, and highly secure data operations.

5.2 **Front-End Implementation:**

We import necessary dependencies and components required for the StatCards component. These include MUI components, form validators, and Axios for handling API requests.

StateCard

```
import { Button, Grid, Icon, styled, Card, CircularProgress } from "@mui/material";
import { Span } from "app/components/Typography";
import { useEffect, useState } from "react";
import { TextValidator, ValidatorForm } from "react-material-ui-form-validator";
import { toast, ToastContainer } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import axios from "axios";
```

Two styled components (CardRoot and TextField) are defined to customize the appearance of the card and text fields used in the component.

```
// Styled component for Card styling
const CardRoot = styled(Card)(({ theme }) => ({
  marginBottom: "24px",
  padding: "24px !important",
  [theme.breakpoints.down("sm")]: { paddingLeft: "16px !important" }
}));

// Styled component for TextFields
const TextField = styled(TextValidator)(() => ({
```

```
  width: "100%",
  marginBottom: "16px"
}));
```

This is the main functional component definition for the StatCards component.

It utilizes React hooks such as useState and useEffect for managing component state and lifecycle.

```
export default function StatCards()
```

Two state variables (isChecking and transactionDetails) are initialized using the useState hook.

isChecking is a boolean state used to indicate whether the form is currently being submitted.

transactionDetails is an object that holds the data entered by the user in the form fields.

```
  // State variables for managing form data and loading state
  const [isChecking, setIsChecking] = useState(false);
  const [transactionDetails, setTransactionDetails] = useState({
    cardNumber: "",
    amount: "",
    cvv: "",
    cardHolderName: "",
    expirationDate: "",
    recipientCardNumber: ""
  });
```

Two useEffect hooks are used to set up form validation rules using ValidatorForm.addValidationRule.

Rules are defined for ensuring that the card number has 16 digits (isCardNum) and the CVV has between 3 to 4 digits (isCvv).

```
useEffect(() => {
  return function cleanup() {
    ValidatorForm.removeValidationRule("isTruthy");
    ValidatorForm.removeValidationRule("isCardNum");
    ValidatorForm.removeValidationRule("isCvv");
  };
}, []);
```

The handleChange function is an event handler for input changes. It updates the transactionDetails state whenever there is a change in the form fields.

The handleSubmit function is an event handler for form submission. It prepares the data for the API request, sends the request using Axios, and handles the response accordingly.

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setIsChecking(true);
  // Data preparation for API request
  const postData = {

...
  };
```

```
    try {
      // Sending POST request to backend API
...
      );

      // Handling response based on success or failure
      if (response.data.status === "success") {
                    toast.success(response.data.message || "Transfer completed
successfully.");
        // Additional success handling code can be added here
      } else {
        toast.error(response.data.message || "Invalid card details.");
        // Additional error handling code can be added here
      }
    } catch (error) {
      toast.error(
          error.response?.data?.message || "An error occurred while processing
your request."
      );
    } finally {
      setIsChecking(false);
    }
  };
```

The component returns JSX markup that includes a form with input fields wrapped in a MUI Card component.

- The form submission is handled by the `handleSubmit` function.
- The component also includes a ToastContainer from react-toastify for

displaying notifications.

```
return (
...
);
```

CheckerCard

We define necessary styled components for customizing the appearance of the card, text fields, and table used in the component.

```
const CardRoot = styled(Card)(({ theme }) => ({
  marginBottom: "24px",
  padding: "24px !important",
  [theme.breakpoints.down("sm")]: { paddingLeft: "16px !important" }
}));

const TextField = styled(TextValidator)(() => ({
  width: "100%",
  marginBottom: "16px"
}));

const StyledTable = styled(Table)(({ theme }) => ({
  whiteSpace: "pre",
  "& thead": {
```

```
    "& tr": { "& th": { paddingLeft: 0, paddingRight: 0 } }
  },
  "& tbody": {
    "& tr": { "& td": { paddingLeft: 0, textTransform: "capitalize" } }
  }
}));
```

This is the main functional component definition for the `CheckerCard` component. It utilizes React hooks such as `useState` and `useEffect` for managing component state and lifecycle.

```
export default function CheckerCard() {
  const [isChecking, setIsChecking] = useState(false);
  const [transactionDetails, setTransactionDetails] = useState({
    cardNumber: "",
    amount: "",
    cvv: "",
    cardHolderName: "",
    expirationDate: ""
  });
  const [cardDetails, setCardDetails] = useState(null);

  useEffect(() => {
    ValidatorForm.addValidationRule("isTruthy", (value) => value);
    ValidatorForm.addValidationRule("isCardNum", (value) => value.length
```

```
=== 16);

    ValidatorForm.addValidationRule("isCvv", (value) => value.length >= 3
&& value.length <= 4);

  }, []);


  useEffect(() => {

    return function cleanup() {

      ValidatorForm.removeValidationRule("isTruthy");

      ValidatorForm.removeValidationRule("isCardNum");

      ValidatorForm.removeValidationRule("isCvv");

    };

  }, []);
```

The `handleChange` function is an event handler for input changes. It updates the `transactionDetails` state whenever there is a change in the form fields.

The `handleSubmit` function is an event handler for form submission. It prepares the data for the API request, sends the request using Axios, and handles the response accordingly.

```
const handleChange = (e) => {

  const { name, value } = e.target;

  setTransactionDetails({

    ...transactionDetails,
```

```
    [name]: value
  });
};

const handleSubmit = async (e) => {
  e.preventDefault();
  setIsChecking(true);

  const postData = {
    cardNumber: transactionDetails.cardNumber,
    cvv: transactionDetails.cvv,
    cardHolderName: transactionDetails.cardHolderName,
    expirationDate: transactionDetails.expirationDate
  };

  try {
    const response = await axios.post(
      "https://abandoned.pythonanywhere.com/api/check/",
      postData
    );

    if (response.data.status === "success") {
      const { data } = response;
```

```
      toast.success(response.data.message || "Card details found.");
      setCardDetails(data);
    } else {
      toast.error(response.data.message || "Invalid card details.");
    }
  } catch (error) {
    toast.error(
      error.response?.data?.message || "An error occurred while processing your request."
    );
  } finally {
    setIsChecking(false);
  }
};
```

The component returns JSX markup that includes a form with input fields wrapped in a MUI Card component. The form submission is handled by the `handleSubmit` function. The component also includes a `ToastContainer` from `react-toastify` for displaying notifications.

```
return (
  <>
    <CardRoot>
      <div>
        <ValidatorForm onSubmit={handleSubmit} onError={() => null}>
```

```
        <Grid container spacing={6}>
          {/* Input fields */}
          <Grid item lg={6} md={6} sm={12} xs={12} sx={{ mt: 2 }}>
            {/* Input fields */}
          </Grid>
          {/* Table displaying card details */}
          <Grid item lg={6} md={6} sm={12} xs={12} sx={{ mt: 2 }}>
            {/* Table displaying card details */}
          </Grid>
        </Grid>
        {/* Submit button */}
      </ValidatorForm>
    </div>
  </CardRoot>
  {/* ToastContainer for displaying notifications */}
  <ToastContainer
    position="top-right"
    autoClose={5000}
    hideProgressBar={false}
    closeOnClick
    pauseOnHover
  />
</>);
```

5.3 Back-End Implementation:

CreditCardServlet

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.stream.Collectors;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import com.google.gson.Gson;
import com.google.gson.JsonObject;
import com.google.gson.reflect.TypeToken;

public class CreditCardServlet extends HttpServlet {

  private final Gson gson = new Gson();
  private static final String DATA_FILE = "carddb.json";

     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Read all credit card data from the JSON file.


    List < CreditCard > creditCardsList = readCreditCardsList();
```

```
    // Parse JSON body of the request.
    JsonObject requestBody = parseJsonRequestBody(request);

    // Extract the transaction details from the request body.
    String senderCardNumber = requestBody.get("cardNumber").getAsString();
    double amountToTransfer = requestBody.get("amount").getAsDouble();
    String senderCvv = requestBody.get("cvv").getAsString();
    String senderName = requestBody.get("cardHolderName").getAsString();
    String expirationDate = requestBody.get("expirationDate").getAsString();
                                    String recipientCardNumber =
requestBody.get("recipientCardNumber").getAsString();

    // Find the sender and recipient cards in the list
             CreditCard senderCard = findCardByNumber(creditCardsList,
senderCardNumber);
            CreditCard recipientCard = findCardByNumber(creditCardsList,
recipientCardNumber);

    // Prepare the JSON response object.
    JsonObject responseJson = new JsonObject();

    // Check the validity of both the sender's and recipient's cards.
    if (senderCard != null && recipientCard != null &&
      senderCard.getExpirationDate().equals(expirationDate) &&
      senderCard.getCvv().equals(senderCvv) &&
      senderCard.getCardHolderName().equals(senderName)) {

      // Check if the sender has enough balance for the transfer.
      if (senderCard.getBalance() >= amountToTransfer) {
```

```
        // Perform the transaction by updating the balances.
        senderCard.setBalance(senderCard.getBalance() - amountToTransfer);
                                recipientCard.setBalance(recipientCard.getBalance()    +
amountToTransfer);

        // Write the updated list back to the file.
        writeCreditCardsList(creditCardsList);

        // Transaction is successful.
        responseJson.addProperty("status", "success");
        responseJson.addProperty("message", "Transfer completed successfully.");
        responseJson.addProperty("cardNumber", senderCardNumber);
        responseJson.addProperty("recipientCardNumber", recipientCardNumber);
        responseJson.addProperty("amount", amountToTransfer);

      } else {
        // Sender does not have enough balance.
        responseJson.addProperty("status", "error");
        responseJson.addProperty("message", "Insufficient balance.");
      }
    } else {
      // Card validation failed.
      responseJson.addProperty("status", "error");
      responseJson.addProperty("message", "Invalid card details.");
    }
    // Write the response back to the client.
    response.setContentType("application/json");
    response.setCharacterEncoding("UTF-8");
    response.getWriter().write(gson.toJson(responseJson));
```

```
  }

  private List < CreditCard > readCreditCardsList() throws IOException {
    try (FileReader reader = new FileReader(DATA_FILE)) {
        return gson.fromJson(reader, new TypeToken < List < CreditCard >> ()
{}.getType());
    } catch (IOException e) {
      throw new RuntimeException("Error reading file", e);
    }
  }

  private void writeCreditCardsList(List < CreditCard > creditCardsList) throws
IOException {
    try (FileWriter writer = new FileWriter(DATA_FILE)) {
      gson.toJson(creditCardsList, writer);
    } catch (IOException e) {
      throw new RuntimeException("Error writing file", e);
    }
  }

  private JsonObject parseJsonRequestBody(HttpServletRequest request) throws
IOException {
    StringBuilder buffer = new StringBuilder();
    try (BufferedReader reader = request.getReader()) {
      String line;
      while ((line = reader.readLine()) != null) {
        buffer.append(line);
      }
    }
```

```
    return gson.fromJson(buffer.toString(), JsonObject.class);
  }
   private CreditCard findCardByNumber(List < CreditCard > creditCardsList,
String cardNumber) {
    return creditCardsList.stream()
      .filter(card -> card.getCardNumber().equals(cardNumber))
      .findFirst()
      .orElse(null);
  }

  // Assuming you have a credit card class
  // Update with getters and setters for new fields.
  class CreditCard {
    private int id;
    private String cardholder_name;
    private String card_number;
    private String expiration_date;
    private String cvv;
    private double balance;
    // Other methods and fields...

    // Constructors, getters, and setters
    public int getId() {
      return id;
    }
    public String getCardHolderName() {
      return cardholder_name;
    }
    public String getCardNumber() {
```

```
  return card_number;
}
public String getExpirationDate() {
  return expiration_date;
}
public String getCvv() {
  return cvv;
}
public double getBalance() {
  return balance;
}

public void setId(int id) {
  this.id = id;
}
public void setCardholderName(String name) {
  this.cardholder_name = name;
}
public void setCardNumber(String number) {
  this.card_number = number;
}
public void setExpirationDate(String date) {
  this.expiration_date = date;
}
public void setCvv(String cvv) {
  this.cvv = cvv;
}
public void setBalance(double balance) {
  this.balance = balance;
```

```
    }
    // Ensure you add other necessary getters and setters...
  }
}
```

CreditCardChecker:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;
import java.util.stream.Collectors;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import com.google.gson.Gson;
import com.google.gson.JsonObject;
import com.google.gson.reflect.TypeToken;

public class CreditCardChecker extends HttpServlet {

  private final Gson gson = new Gson();
  private static final String DATA_FILE = "carddb.json";

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
```

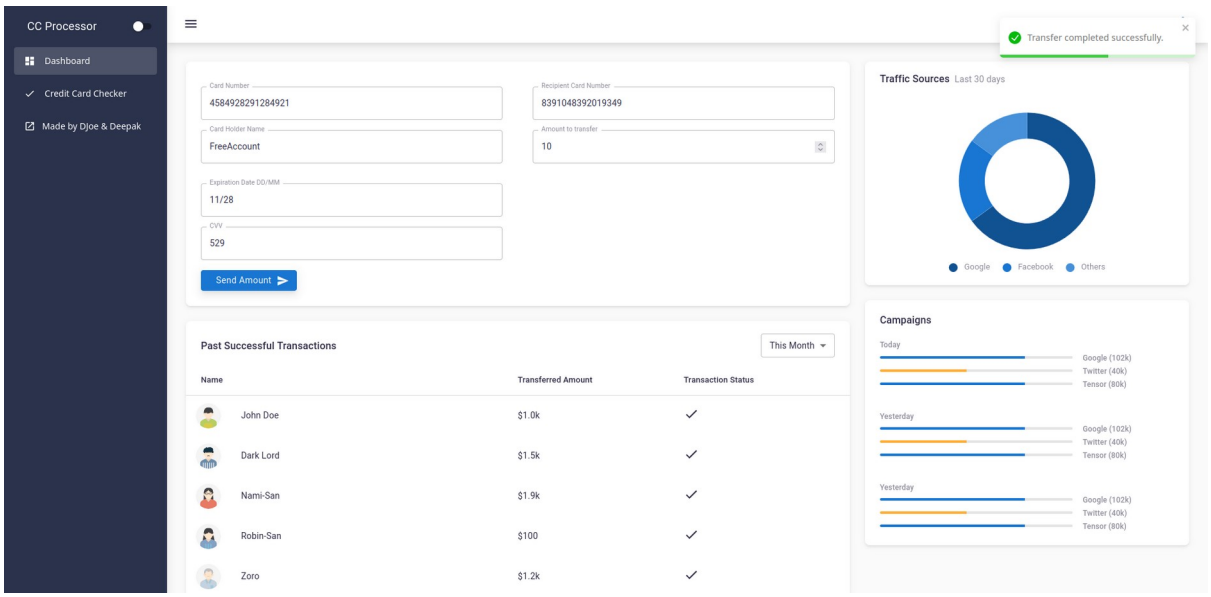
```
    List<CreditCard> creditCardsList = readCreditCardsList();
    JsonObject requestBody = parseJsonRequestBody(request);

    String senderCardNumber = requestBody.get("cardNumber").getAsString();
    String senderCvv = requestBody.get("cvv").getAsString();
    String senderName = requestBody.get("cardHolderName").getAsString();
    String expirationDate = requestBody.get("expirationDate").getAsString();

            CreditCard senderCard = findCardByNumber(creditCardsList,
senderCardNumber);
    JsonObject responseJson = new JsonObject();

    if (senderCard != null &&
      senderCard.getExpirationDate().equals(expirationDate) &&
      senderCard.getCvv().equals(senderCvv) &&
      senderCard.getCardHolderName().equals(senderName)) {

      // If validation is successful, include card details in the response
      responseJson.addProperty("status", "success");
      responseJson.addProperty("message", "Card details found.");
      responseJson.addProperty("id", senderCard.getId());
                                    responseJson.addProperty("cardHolderName",
senderCard.getCardHolderName());
      responseJson.addProperty("cardNumber", senderCard.getCardNumber());
                                       responseJson.addProperty("expirationDate",
senderCard.getExpirationDate());
      responseJson.addProperty("cvv", senderCard.getCvv());
      responseJson.addProperty("balance", senderCard.getBalance());
```

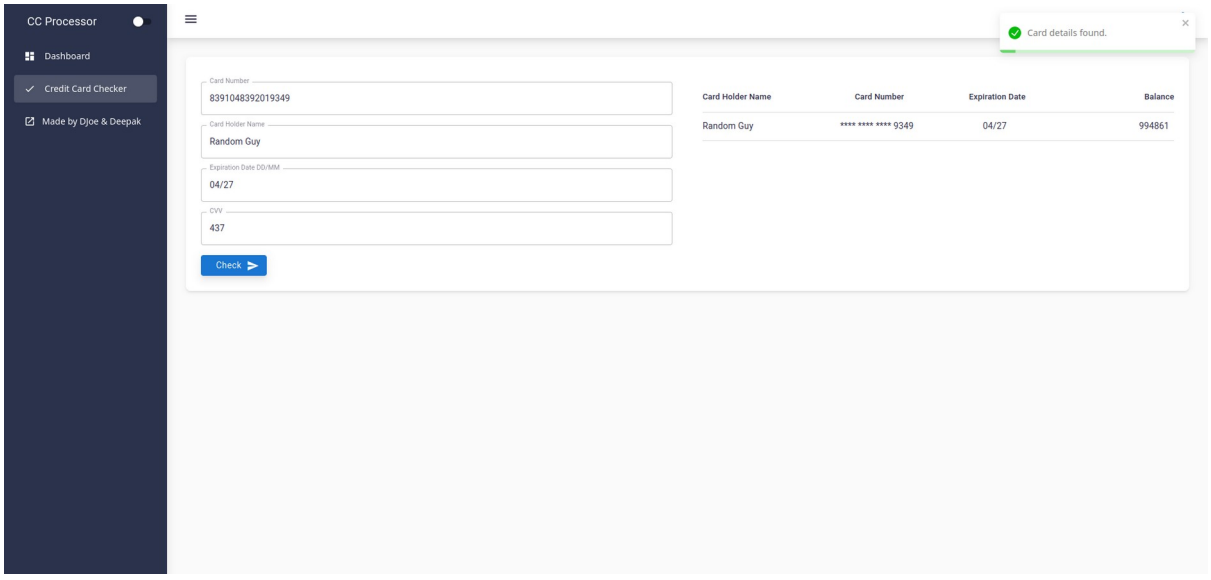
```
    } else {
      responseJson.addProperty("status", "error");
      responseJson.addProperty("message", "Invalid card details.");
    }

    response.setContentType("application/json");
    response.setCharacterEncoding("UTF-8");
    response.getWriter().write(gson.toJson(responseJson));
  }


  private List < CreditCard > readCreditCardsList() throws IOException {
    try (FileReader reader = new FileReader(DATA_FILE)) {
        return gson.fromJson(reader, new TypeToken < List < CreditCard >> ()
{}.getType());
    } catch (IOException e) {
      throw new RuntimeException("Error reading file", e);
    }
  }

  private JsonObject parseJsonRequestBody(HttpServletRequest request) throws
IOException {
    StringBuilder buffer = new StringBuilder();
    try (BufferedReader reader = request.getReader()) {
      String line;
      while ((line = reader.readLine()) != null) {
        buffer.append(line);
      }
    }
```

```
    return gson.fromJson(buffer.toString(), JsonObject.class);
  }
   private CreditCard findCardByNumber(List < CreditCard > creditCardsList,
String cardNumber) {
    return creditCardsList.stream()
      .filter(card -> card.getCardNumber().equals(cardNumber))
      .findFirst()
      .orElse(null);
  }

  class CreditCard {
    private int id;
    private String cardholder_name;
    private String card_number;
    private String expiration_date;
    private String cvv;
    private double balance;
    public int getId() {
      return id;
    }
    public String getCardHolderName() {
      return cardholder_name;
    }
    public String getCardNumber() {
      return card_number;
    }
    public String getExpirationDate() {
      return expiration_date;
    }
```

```
    public String getCvv() {
      return cvv;
    }
    public double getBalance() {
      return balance;
    }

    public void setId(int id) {
      this.id = id;
    }
    public void setCardholderName(String name) {
      this.cardholder_name = name;
    }
    public void setCardNumber(String number) {
      this.card_number = number;
    }
    public void setExpirationDate(String date) {
      this.expiration_date = date;
    }
    public void setCvv(String cvv) {
      this.cvv = cvv;
    }
    public void setBalance(double balance) {
      this.balance = balance;
    }
  }
}
```

OUTPUT:


CC Processor
Dashboard
Credit Card Checker
Made by Djoe & Deepak
Transfer completed successfully.
Card Number
4584928291284921
Recipient Card Number
8391048392019349
Card Holder Name
FreeAccount
Amount to transfer
10
Expiration Date DD/MM
11/28
CVV
529
Send Amount
Traffic Sources Last 30 days
Google
Facebook
Others
Past Successful Transactions
This Month
Name
Transferred Amount
Transaction Status
John Doe
$1.0k
Dark Lord
$1.5k
Nami-San
$1.9k
Robin-San
$100
Zoro
$1.2k
Campaigns
Today
Google (102k)
Twitter (40k)
Tensor (80k)
Yesterday
Google (102k)
Twitter (40k)
Tensor (80k)
Yesterday
Google (102k)
Twitter (40k)
Tensor (80k)


CC Processor
Dashboard
Credit Card Checker
Made by Djoe & Deepak
Card details found.
Card Number
8391048392019349
Card Holder Name
Random Guy
Expiration Date DD/MM
04/27
CVV
437
Check
Card Holder Name
Card Number
Expiration Date
Balance
Random Guy
**** **** **** 9349
04/27
994861

CHAPTER-6
CONCLUSION AND FUTURE ENHANCEMENT

6.1. CONCLUSION

The implementation of a robust credit card processing system is crucial for facilitating secure transactions and ensuring the smooth functioning of financial operations. This project integrated various algorithms and techniques in image processing to tackle the challenges associated with detecting and classifying plant diseases. By employing multiple algorithms, we gained insights into the effectiveness of different approaches, which proved instrumental in enhancing the system's performance.

However, one of the primary challenges encountered during this endeavor was the limited availability of comprehensive datasets encompassing diverse agricultural crops. This scarcity of data posed a significant hurdle in achieving optimal analytical outcomes. Nevertheless, through rigorous research and fieldwork, valuable insights were gleaned, paving the way for future enhancements and refinements.

Looking ahead, there is immense potential to expand the scope of this model by transforming it into web and mobile applications. By doing so, we can democratize access to agricultural insights, empowering farmers and agricultural departments to make informed decisions. Furthermore, by leveraging high-end cameras and centralized processing servers, we can streamline the detection of plant diseases and disseminate relevant guidance and recommendations to end-users promptly.

6.2 FUTURE WORK

In the future, we envision a sophisticated ecosystem where high-resolution images captured by advanced cameras are seamlessly transmitted to centralized servers for comprehensive analysis. Upon successful detection of plant diseases, tailored guidance and insights will be relayed back to farmers and stakeholders in real-time, enabling proactive decision-making and intervention.

To facilitate broader access to the wealth of data accumulated through this endeavor, we plan to develop an Application Programming Interface (API). This API will serve as a gateway to the repository of collected data, empowering researchers, policymakers, and stakeholders to harness valuable insights for the betterment of agricultural practices.

Furthermore, this project lays the foundation for future research endeavors and contributions in the realm of agricultural analytics. As new datasets and insights emerge, they can be seamlessly integrated into the existing framework, fostering continuous improvement and innovation.

In summary, this project represents a stepping stone towards revolutionizing agricultural practices and bolstering food security. By harnessing the power of technology and data-driven insights, we aspire to catalyze positive change and contribute to the advancement of our agricultural landscape.

Feel free to adjust or expand upon these sections as needed for your project! If you have any further questions or require additional assistance, please don't hesitate to ask.