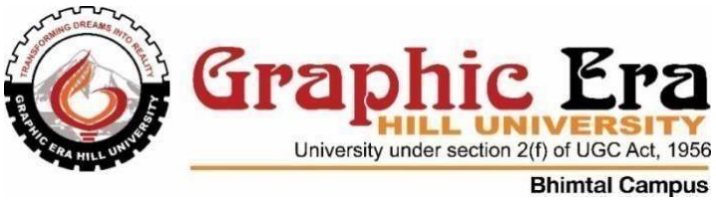


Software Engineering Lab (PCS-611)

Submitted inpartial fulfillment of the requirement for the VI semester

Bachelor of Technology

By

Arjun Pandey

University Roll No

2161094

Underthe Guidance of

Mr. Parthak Mehra

Lecturer

Deptt. of CSE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS

SATTAL ROAD, P.O. BHOWALI DISTRICT-

NAINITAL-263132

2023-2024

CERTIFICATE

The term work of **Software Engineering Lab (PCS-611),** being submitted by Arjun Pandey, Roll no 2161094 to Graphic Era Hill University Bhimtal Campus for the award of bona fide work carried out by her. She has worked under my guidance and supervision and fulfilled the requirement for the submission of this work report.

(……)

Mr. Parthak Mehra

ACKNOWLEDGEMENT

I take immense pleasure in thanking **Mr. Parthak Mehra** (GEHU Bhimtal Campus) for allowing us to carry out this project work under his excellent and optimistic supervision. This has all been possible due to her novel inspiration, able guidance and useful suggestions that have helped me in developing my subject concepts as a student.

I want to extend thanks to our President **"Prof. (Dr.) Kamal Ghanshala"** for providing us infrastructure and facilities to work in need without which this work would not be possible.

Arjun Pandey

STUDENT'S DECLARATION

I, **Arjun Pandey** hereby declare the work, which is being presented in the report, entitled **Term work of Software Engineering Lab (PCS-611)** in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (Computer Science)** in the session **2023-2024** for **semester VI**, is an authentic record of my own work carried out under the supervision of **Mr. Parthak Mehra** (GraphicEra Hill University, Bhimtal) The matter embodied in this project has not been submitted by me for the award of any other degree.

Date: …… …….

(Full signature of student)

Graphic Era HILL UNIVERSITY
University under section 2(f) of UGC Act, 1956
Bhimtal Campus

Computer Science and Engineering Department

Software Engineering Lab (PCS-611)

Index/List of Experiments

1.	Development of problem statement.
2.	Preparation of Software Requirement Specification (SRS) document .
3	Data Flow Diagram (DFD)
4.	ER diagram
5.	Preparation of Risk Management
6.	Use Case Diagram.
7.	Sequence Diagram

8.	Study and usage of any Design phase case tool (STARUML)
9.	Collaboration Diagram.
10.	Develop test cases for unit testing and integration testing.

Introduction

1.1 Purpose

The purpose of this SRS is to provide a detailed description of the Credit Card Fraud Detection System, including its functionalities, performance, and constraints. This document aims to guide the development and testing of the system.

1.2 Scope

The Credit Card Fraud Detection System is designed to identify fraudulent credit card transactions in realtime. It will analyze transaction patterns and flag suspicious activities for further investigation. The system will be used by financial institutions to reduce fraud losses and improve customer trust.

1.3 Definitions, Acronyms, and Abbreviations

- **Fraud**: Unauthorized or illegal transactions made using a credit card.
- **Real-Time Processing**: Immediate analysis and response to transactions as they occur.
- **False Positive**: A legitimate transaction incorrectly flagged as fraudulent.
- **False Negative**: A fraudulent transaction that is not flagged by the system.

2. Overall Description

2.1 Product Perspective

The Credit Card Fraud Detection System is an independent software module that integrates with existing transaction processing systems of financial institutions. It will interface with transaction databases and alert systems.

2.2 Product Functions

- **Transaction Analysis**: Analyze each transaction in real-time to detect potential fraud.
- **Pattern Recognition**: Identify unusual spending patterns and flag them.
- **Alert Generation**: Generate alerts for suspected fraudulent transactions.
- **Reporting**: Provide detailed reports of flagged transactions for review.
- **Learning Mechanism**: Continuously learn and adapt to new fraud patterns using machine learning techniques.

2.3 User Characteristics

- **Fraud Analysts**: Review flagged transactions and decide on further action.
- **System Administrators**: Manage system configurations and updates.
- **End Users**: Indirectly benefit from reduced fraud and increased security..

3. Specific Requirements

3.1 Functional Requirements

- **Transaction Input**: The system shall receive transaction data including amount, date, time, location, and card details.
- **Fraud Detection Algorithm**: The system shall implement machine learning algorithms to detect fraudulent transactions.

- **Alert System**: The system shall generate alerts for transactions deemed suspicious.
- **User Interface**: The system shall provide an interface for fraud analysts to review and manage alerts.
- **Reporting**: The system shall generate reports summarizing flagged transactions and detection performance.

3.2 Performance Requirements

- The system shall process transactions in less than 2 seconds.
- The system shall handle at least 1000 transactions per second during peak times.

3.3 Security Requirements

- The system shall encrypt all sensitive data in transit and at rest.
- The system shall implement access controls to restrict data access to authorized personnel only.

4. External Interface Requirements

4.1 User Interfaces

- **Dashboard**: For fraud analysts to view alerts and reports.
- **Configuration Panel**: For system administrators to manage settings.

4.2 Hardware Interfaces

- Integration with the financial institution's servers where transaction data is stored.

4.3 Software Interfaces

- API integration with existing transaction processing systems for data input and alert notifications.

5. Other Nonfunctional Requirements

5.1 Performance

The system shall ensure minimal impact on the transaction processing time of the existing systems.

5.2 Safety

- The system shall have mechanisms to fail gracefully without causing disruption to the transaction processing system.

5.3 Security

- Regular security audits and vulnerability assessments shall be conducted.

6. Appendices

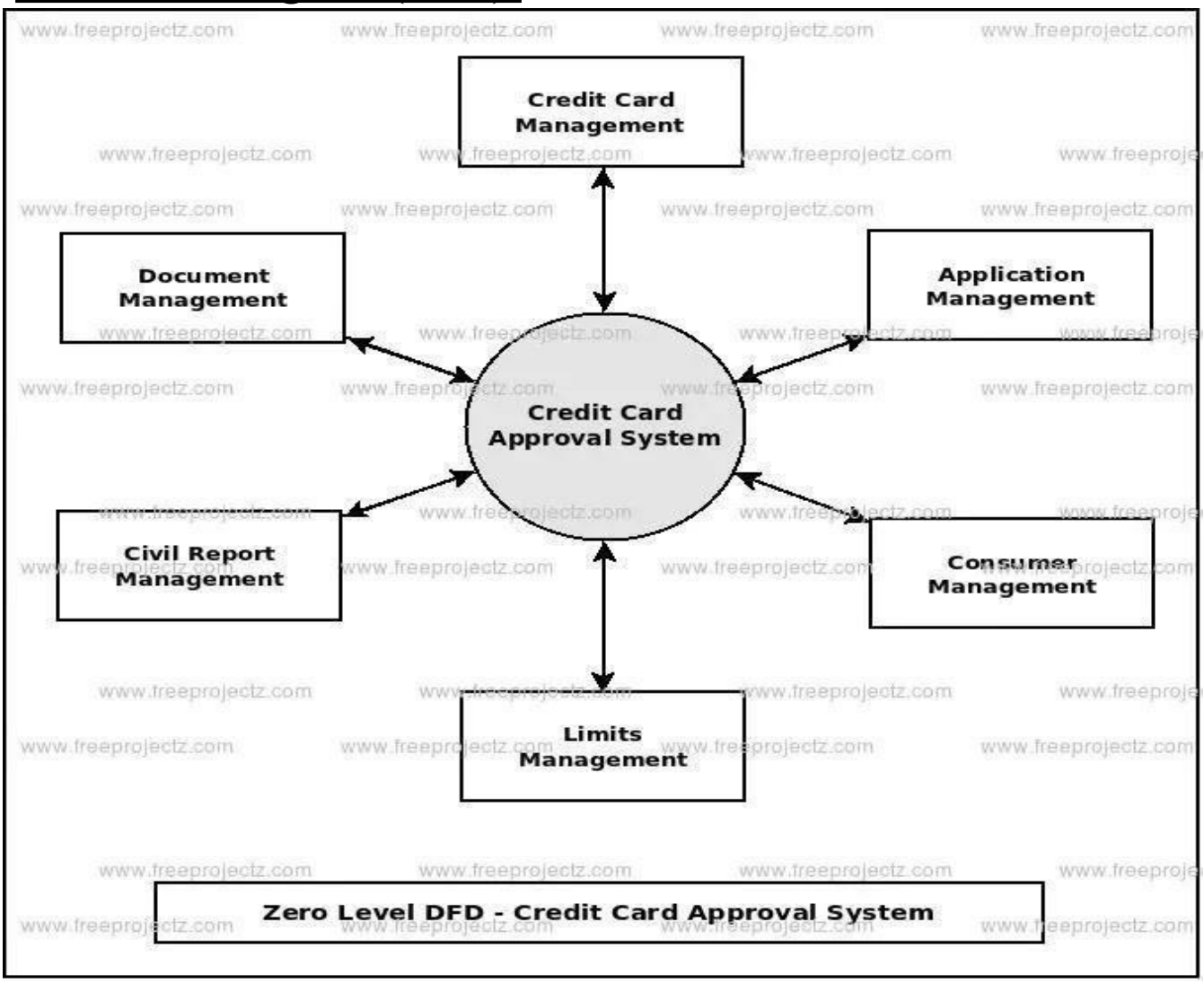
6.1 Glossary

- **Anomaly Detection**: Identifying unusual patterns that do not conform to expected behavior.
- **Machine Learning**: Algorithms that improve automatically through experience and data analysis.

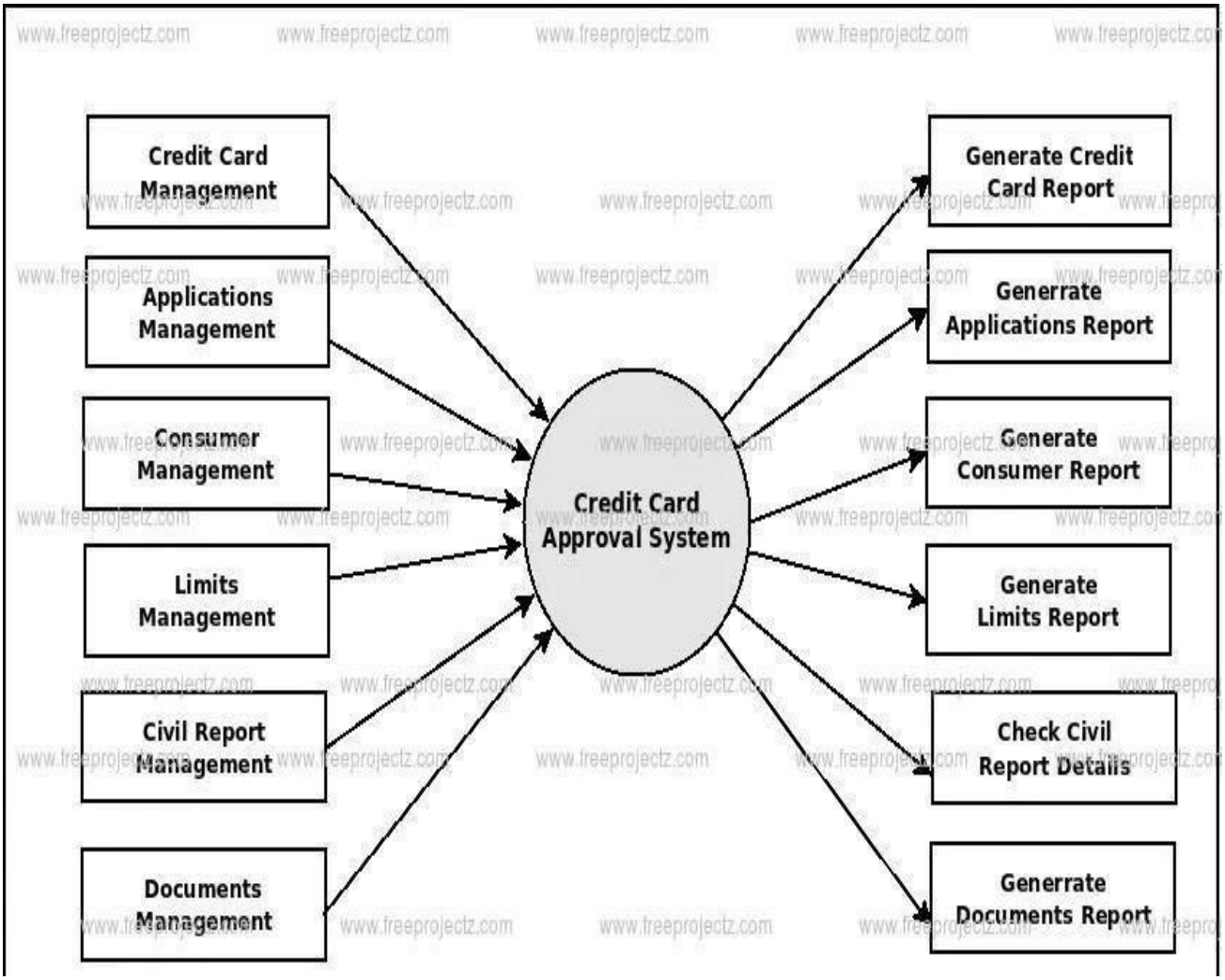
6.2 References

Detailed list of articles, guidelines, and papers referenced in the document.

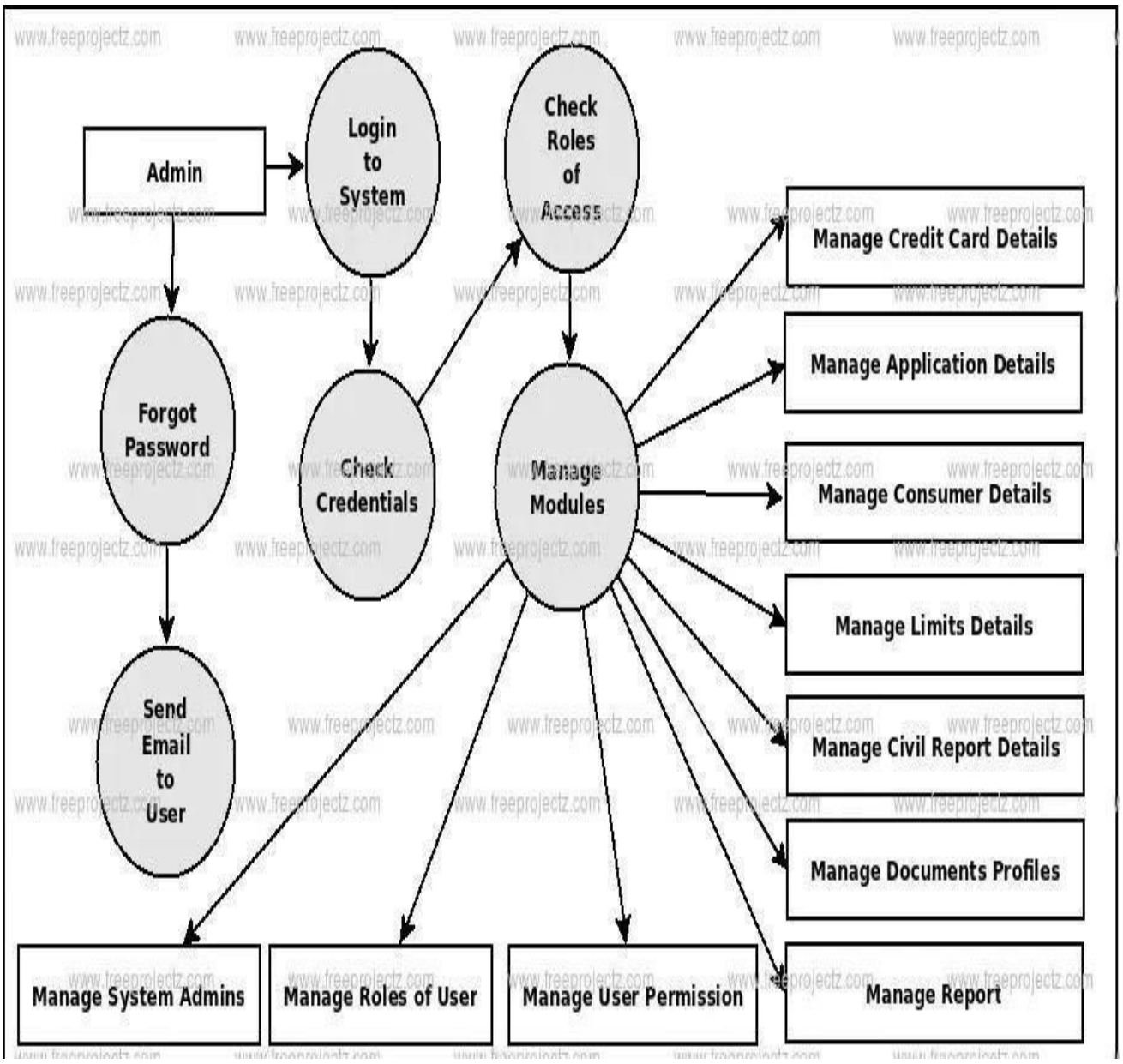
Data Flow Diagram (DFD):



Zero Level DFD

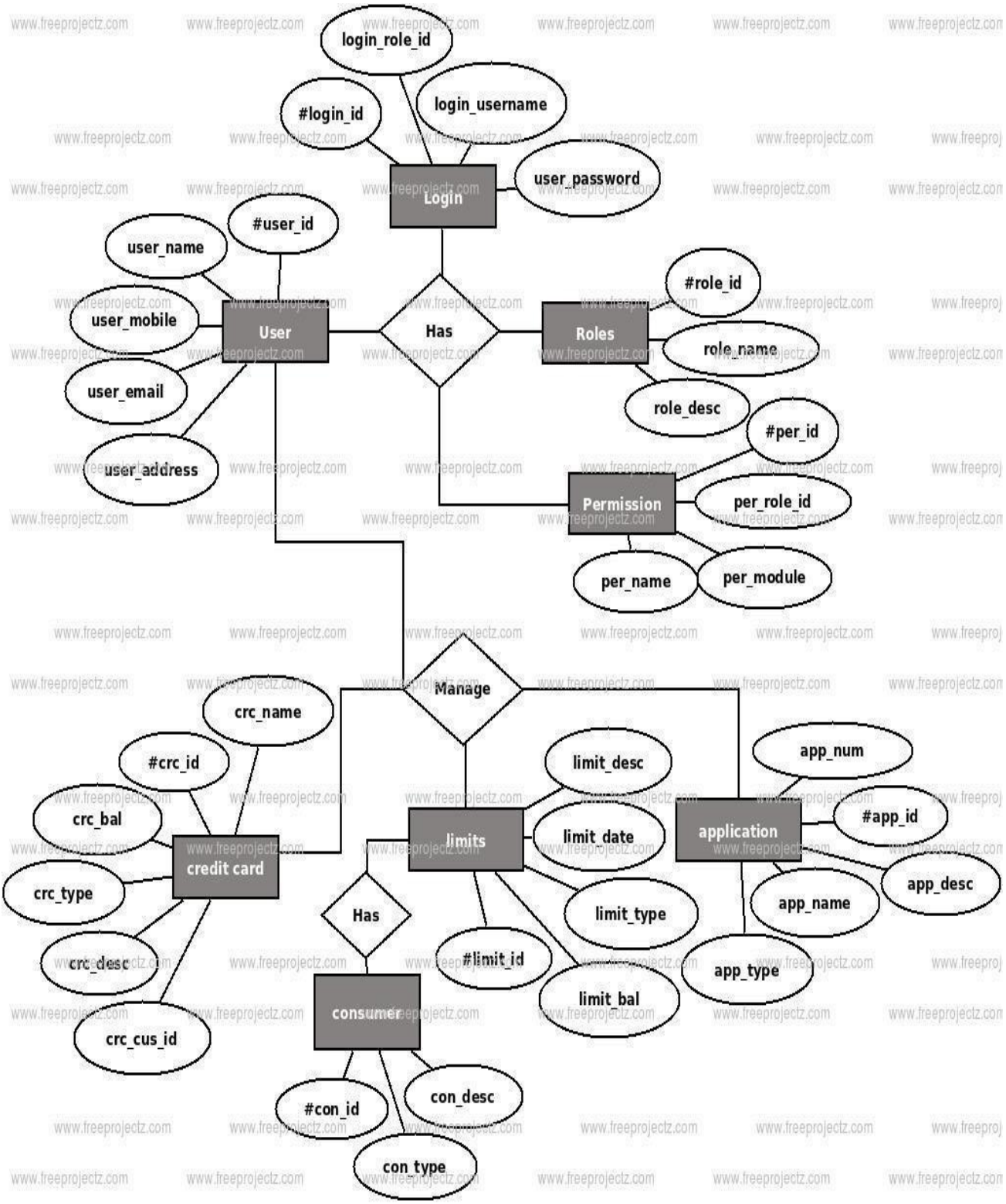


First Level DFD



Second Level DFD

ER diagram :



ER Diagram For Credit Card Approval System

Preparation of Risk Management :

Risk management for a Credit Card Fraud Detection system involves identifying potential risks, assessingtheir likelihood and impact, and developing strategies to mitigate or manage them-

1. Risk Identification

Identify potential risks that could impact the system's performance, security, compliance, and operational efficiency.

Examples of Risks:

Technical Risks:

- Algorithm inaccuracies (false positives and false negatives) ○ System integration issues ○ Software bugs and errors ○ Performance degradation **Operational Risks**: ○ System downtime ○ Scalability issues ○ Data loss or corruption
  Security Risks:
- Data breaches ○ Unauthorized access
- Internal threats from malicious insiders ○ Vulnerabilities in the system
  Compliance Risks:
- Non-compliance with data privacy laws (e.g., GDPR, CCPA)
- Non-compliance with financial regulations ○ Legal penalties and reputational damage

2. Risk Assessment

Evaluate the identified risks to determine their potential impact and likelihood.

Steps:

1. **Impact Assessment**:
   - Determine the severity of the consequences if the risk occurs.
   - Categories: High (major disruption or damage), Medium (moderate disruption), Low (minor disruption).
2. **Likelihood Assessment**:
   - Estimate the probability of the risk occurring. ○ Categories: High (likely to occur), Medium (possible but not certain), Low (unlikely to occur).

3. Risk Mitigation Strategies

Develop strategies to minimize or manage the risks based on their assessed impact and likelihood.

Technical Mitigation:

- Implement robust and up-to-date fraud detection algorithms.
- Regularly test and update the system to handle new fraud patterns.
- Employ redundancy and failover mechanisms to ensure continuous operation.
- Use machine learning models to improve detection accuracy and reduce false positives/negatives.

Operational Mitigation:

- Ensure high availability with redundant systems and disaster recovery plans.
- Perform regular performance monitoring and scalability testing. Implement comprehensive data backup procedures.

Security Mitigation:

- Use strong encryption for data in transit and at rest.
- Implement multi-factor authentication and access controls.
- Conduct regular security audits and vulnerability assessments. Establish protocols for incident response and management.

Compliance Mitigation:

- Stay updated with the latest regulations and compliance requirements.
- Conduct regular compliance audits and assessments.
- Maintain thorough documentation of all processes and procedures.
- Ensure transparent communication and reporting to regulatory bodies.

4. Risk Monitoring and Review

Continuously monitor and review risks to ensure they are being managed effectively and to identify any new risks that may arise.

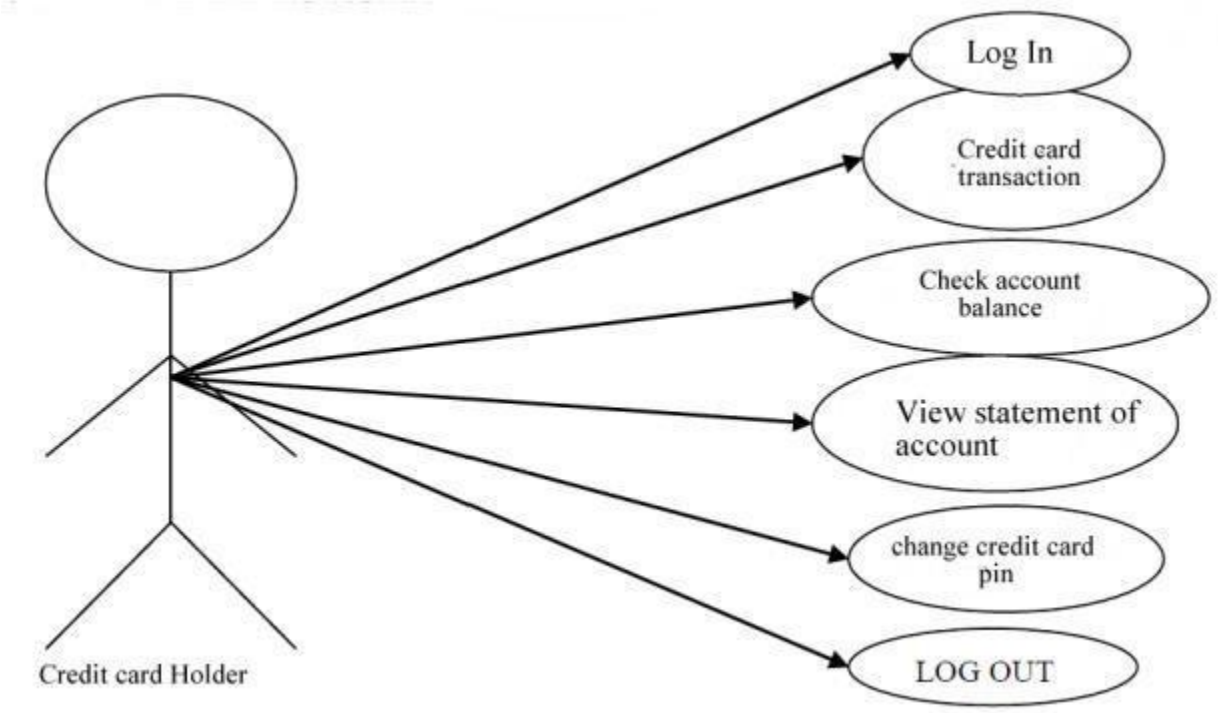
Monitoring:

- Use automated tools and dashboards to track risk indicators and system performance.
- Regularly review system logs, alerts, and reports for signs of potential issues.
- Establish a risk management team to oversee ongoing risk assessment and mitigation efforts.

Review:

- Periodically reassess risks and update the risk management plan accordingly.
- Conduct regular reviews of mitigation strategies to ensure they remain effective.
- Engage stakeholders in the review process to gather feedback and insights.
- Update training programs for employees to ensure they are aware of the latest risks and mitigation techniques.

Use Case Diagram :

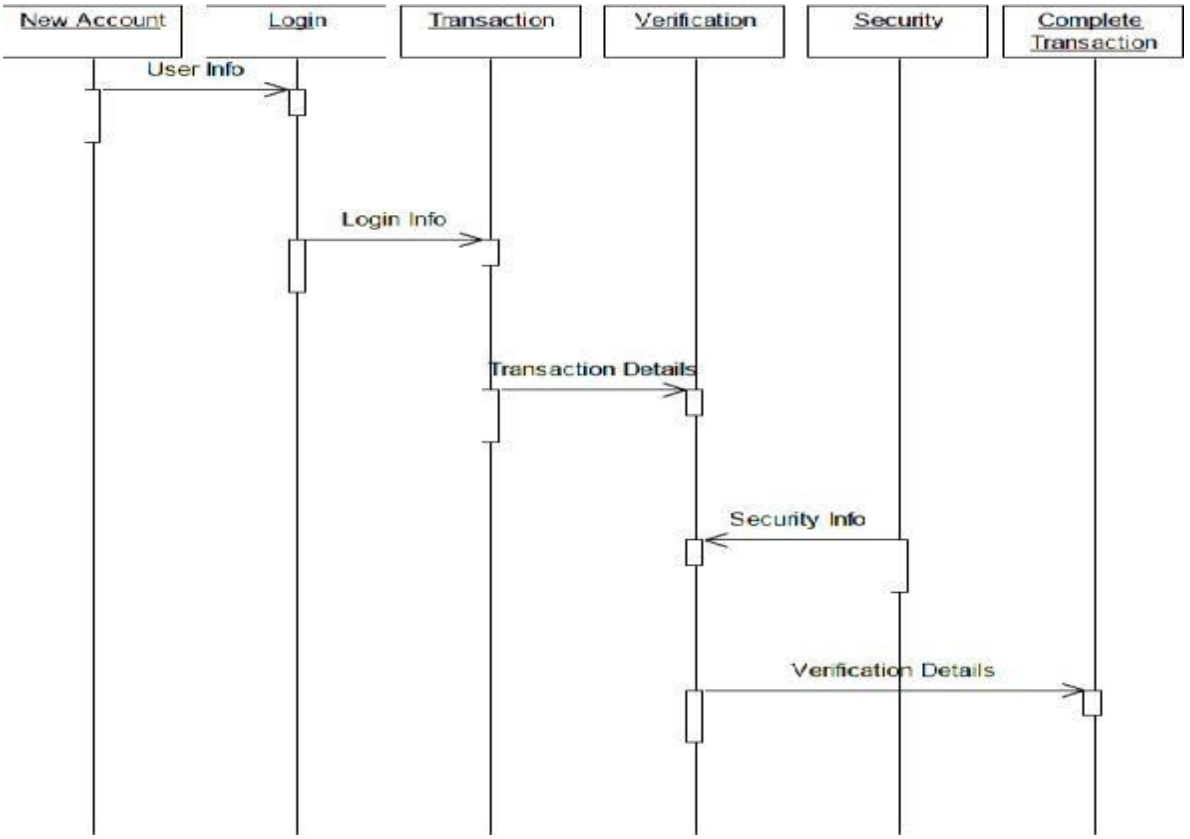


Use Case Diagram

Sequence Diagram:

A sequence diagram represents the sequence and interactions of a given USE-CASE or scenario. Sequence diagrams can capture most of the information about the system. Most object to object interactions and operations are considered events and events include signals, inputs, decisions, interrupts, transitions and actions to or from users or external devices.

An event also is considered to be any action by an object that sends information. The event line represents a message sent from one object to another, in which the “form” object is requesting an operation be performed by the “to” object. The “to” object performs the operation using a method that the class contains.



Study and usage of any Design phase case tool (STARUML)

Star UML is a UML (Unified Modeling Language) tool, introduce by MKLab. It is an open-source modeling tool that supports the UML framework for system and software modeling. Star UML is based on UML version 1.4, it provides 11 different types of diagram and it accepts UML 2.0 notation. Version 2.0 was released for beta testing under a property license.

Star UML is actively supporting the **MDA (Model Driven Architecture).** It approaches by supportingthe UML profile concept and allowing it to generate code for multiple languages. It also provides a number of bug fixes and improved compatibility with the modern versions of the Windows Operating System. Star UML is mostly used by the Agile and small development teams, professional persons andused by the educational institutes

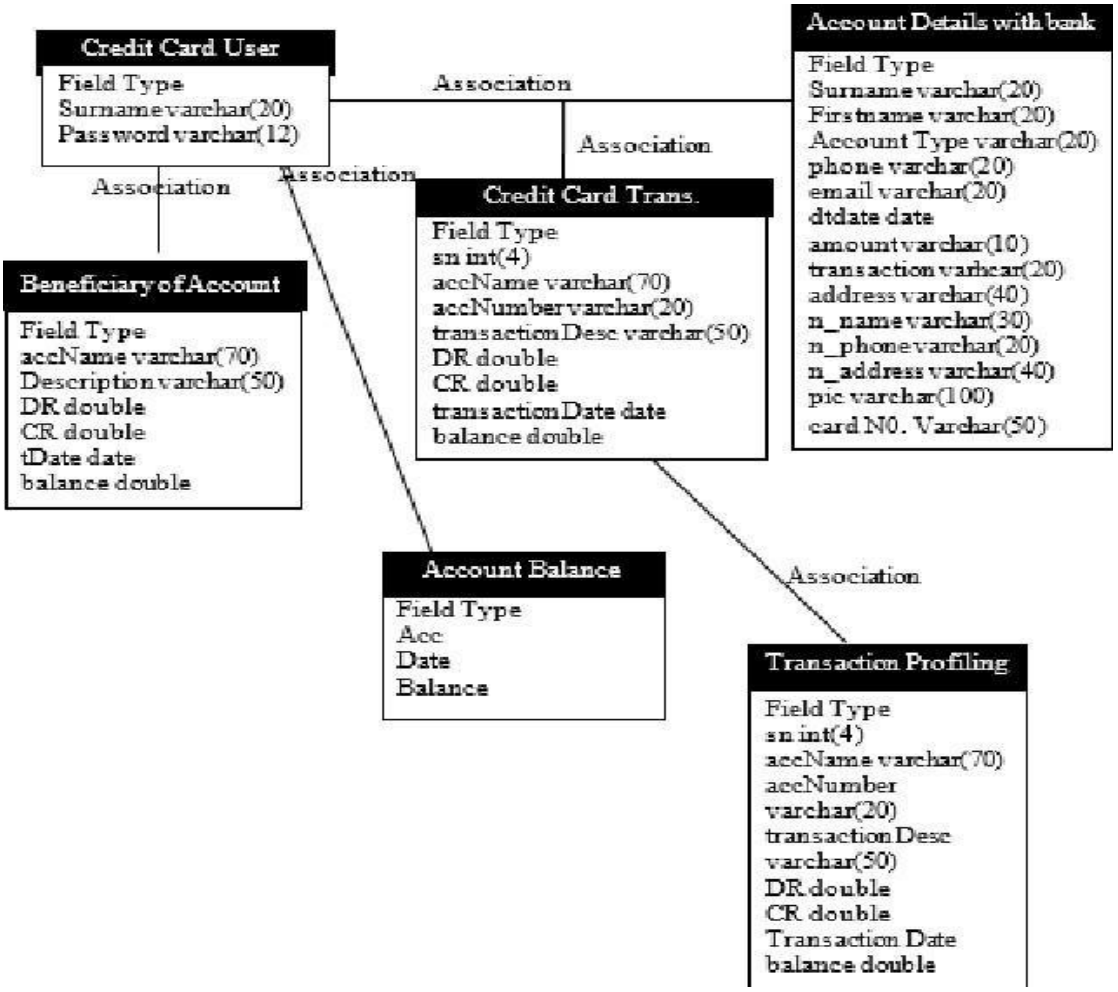
Class Diagram:

The class diagram, also referred to as object modeling is the main static analysis diagram. The main task of object modeling is to graphically show what each object will do in the problem domain. The problem domain describes the structure and the relationships among objects.

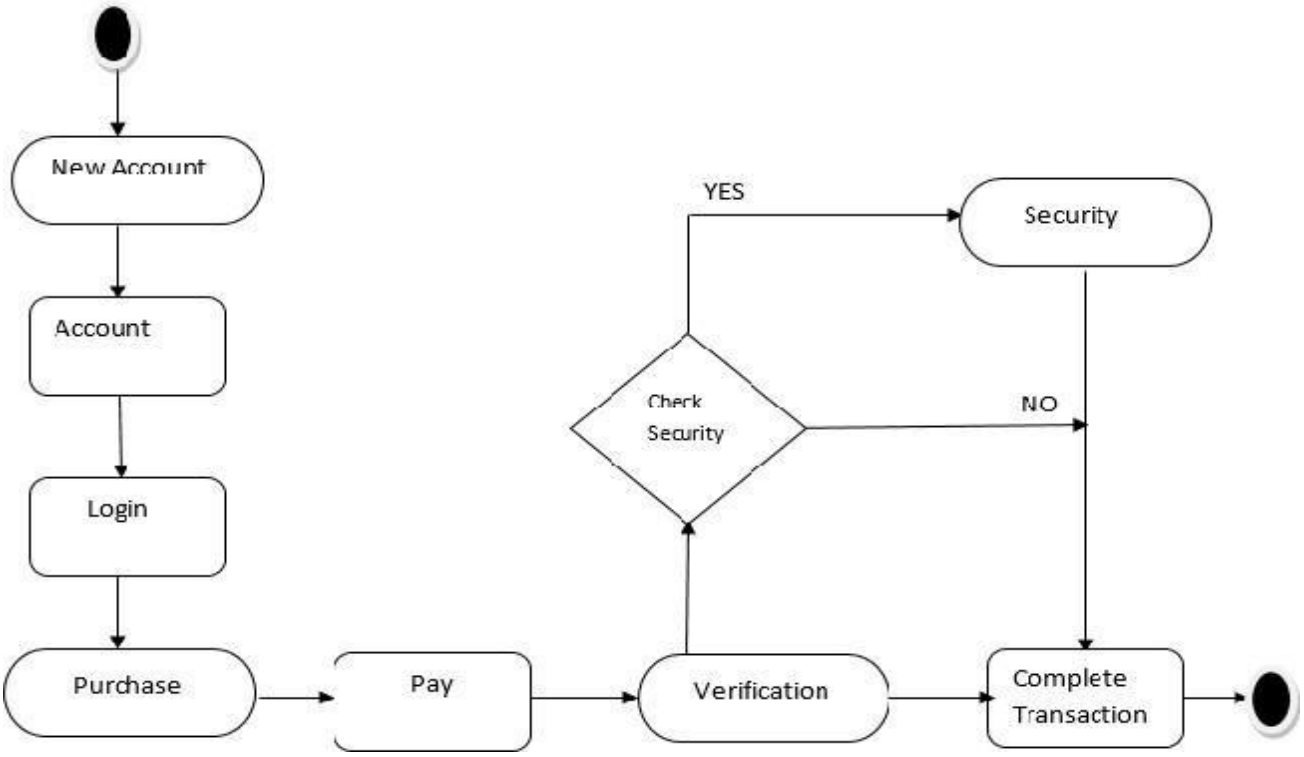
1. **Transaction** ○ Attributes:
    - Transaction Id: String
    - Card Number: String
    - amount: Double
    - date: Date
    - location: String ○ Methods:
    - validate(): Boolean
2. **User**
    - ○ Attributes:
        - User Id: String
        - Name: String
        - email: String
        - Contact Number: String ○ Methods:
        - Get Details(): User
3. **Card**
    - ○ Attributes:
        - Card Number: String
        - Card Type: String ▪ Expiry Date: Date
        - User Id: String ○ Methods:
        - Validate Card(): Boolean
4. **Fraud Detection Algorithm** ○ Attributes:
    - Algorithm Id: String
    - name: String
    - parameters: Map<String, Object> ○ Methods:
    - Detect Fraud(Transaction): Boolean
5. **Alert** ○ Attributes:
    - Alert Id: String
    - Transaction Id: String
    - Alert Date: Date
    - status: String ○ Methods:
    - Create Alert(Transaction): Alert
    - Update Status(String): Boolean
6. **Report**
    - ○ Attributes:
        - Report Id: String
        - Start Date: Date
        - End Date: Date
        - transactions: List<Transaction> ○ Methods:
        - Generate Report(start Date: Date, end Date: Date): Report
7. **Fraud Analyst**
    - ○ Attributes:
        - Analyst Id: String
        - name: String
        - email: String ○ Methods:

- Review Alert(Alert): Boolean

Class Diagram



Collaboration Diagram :



Here's how the objects might interact in a collaboration diagram for detecting and managing credit card fraud:

1. **User** initiates a transaction.
2. **Transaction** details are created and linked to **Card**.
3. **Fraud Detection Algorithm** analyses the **Transaction**.
4. If the transaction is flagged as fraudulent, an **Alert** is created.
5. **Fraud Analyst** reviews the **Alert**.
6. **Report** is generated based on **Transaction** and **Alert** details

Develop test cases for unit testing and integration testing

Unit Testing

Unit testing focuses on individual components to ensure they work correctly in isolation. Here are some sample test cases:

1. Transaction Class

Test Case: Validate Transaction

- **Description**: Ensure that the `validate` method correctly validates a transaction.
- **Inputs**: Transaction with valid details.
- **Expected Output**: `True`
- **Test Steps**:

1. Create a `Transaction` object with valid details.
2. Call the `validate` method.
3. Verify that the output is `True`.

Test Case: Invalid Transaction Amount

- **Description**: Ensure that a transaction with a negative amount is invalid.
- **Inputs**: Transaction with a negative amount.
- **Expected Output**: `False`
- **Test Steps**:
  1. Create a `Transaction` object with a negative amount.
  2. Call the `validate` method.
  3. Verify that the output is `False`.

2. Fraud Detection Algorithm Class Test Case: Detect Fraudulent Transaction

- **Description**: Ensure that the `detect Fraud` method correctly identifies a fraudulent transaction.
- **Inputs**: Suspicious transaction details.
- **Expected Output**: `True`
- **Test Steps**:
  1. Create a `Fraud Detection Algorithm` object.
  2. Provide a suspicious `Transaction` object.
  3. Call the `detect Fraud` method.
  4. Verify that the output is `True`.

Test Case: Detect Legitimate Transaction

- **Description**: Ensure that the `detect Fraud` method does not flag a legitimate transaction as fraudulent.
- **Inputs**: Legitimate transaction details.
- **Expected Output**: `False`
- **Test Steps**:
  1. Create a `Fraud Detection Algorithm` object.
  2. Provide a legitimate `Transaction` object.
  3. Call the `detect Fraud` method.
  4. Verify that the output is `False`.

3. Alert Class

Test Case: Create Alert

- **Description**: Ensure that the `create Alert` method correctly generates an alert for a suspicious transaction.
- **Inputs**: Suspicious transaction details.
- **Expected Output**: An `Alert` object with correct details.
- **Test Steps**:
  1. Create a `Transaction` object with suspicious details.
  2. Call the `create Alert` method with the transaction.
  3. Verify that an `Alert` object is created with the correct details.

Integration Testing

Integration testing focuses on the interactions between different components to ensure they work together correctly. Here are some sample test cases:

1. Transaction Processing and Fraud Detection Integration

Test Case: Process Transaction and Detect Fraud

- **Description**: Ensure that the system correctly processes a transaction and detects fraud.
- **Inputs**: Transaction details. - **Expected Output**: Fraud detection result (True/False). - **Test Steps**:
    1. Process a `Transaction` through the system.
    2. Verify that the `Fraud Detection Algorithm` analyzes the transaction.
    3. Check the fraud detection result.

2. Alert Generation and Review Integration

Test Case: Generate and Review Alert

- **Description**: Ensure that alerts are generated for suspicious transactions and can be reviewed by a fraud analyst.
- **Inputs**: Suspicious transaction details. - **Expected Output**: Alert status updated after review.
- **Test Steps**:
    1. Process a suspicious `Transaction` through the system.
    2. Verify that an `Alert` is generated.
    3. Simulate a fraud analyst reviewing the alert.
    4. Verify that the alert status is updated.

3. Report Generation Integration

Test Case: Generate Fraud Report

- **Description**: Ensure that the system correctly generates a report of fraudulent transactions.
- **Inputs**: Date range. - **Expected Output**: Report object with a list of fraudulent transactions. - **Test Steps**:
    1. Process multiple transactions, some of which are fraudulent.
    2. Call the `generate Report` method with a specific date range.
    3. Verify that the report includes all fraudulent transactions within the date range.