Major Project Report

on

Security Information and Event Management-Driven Financial Fraud Detection: Enhancing Security and Risk Mitigation

Submitted by

Akshay K R (20221094)
Nandulal Krishna (20221097)
Pravaal B Nath (20921038)
Rahul R (20221098)

In partial fulfilment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering.

DIVISION OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF ENGINEERING
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

March 2025

DIVISION OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF ENGINEERING
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CERTIFICATE

Certified that this is a Main Project Report titled

SIEM-Driven Financial Fraud Detection: Enhancing Security and Risk Mitigation

Submitted by

Akshay K R (20221094)
Nandulal Krishna (20221097)
Pravaal B Nath (20921038)
Rahul R (20221098)

of VIII Semester, Computer Science and Engineering in the year 2025 in partial fulfillment requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering of Cochin University of Science and Technology.

Mr.Pramod Pavithran	Ms. Preetha S Ms. Thasnim K M	Mr. V. Damodaran
Head of Division	Project Coordinators	Project Guide

Declaration

We, Akshay KR, Nandulal Krishna, Pravaal B Nath and Rahul R, hereby declare that the project titled **"SIEM-Driven Financial Fraud Detection: Enhancing Security and Risk Mitigation"** is a bonafide work done by us during the year 2025 under the guidance of Mr. V. Damodaran, Associate Professor at School of Engineering, CUSAT and that no part has formed the basis for the award of any degree, diploma, associate ship, fellowship or any other similar title or recognition in any other university.

Acknowledgement

We would like to express our deepest gratitude to the Almighty for providing us with the guidance and wisdom throughout this venture. We extend our heartfelt thanks to Mr. V. Damodaran, our project guide, whose unwavering support, insightful guidance, and constant encouragement have been invaluable throughout this journey.

We also wish to express our sincere appreciation to Ms. Preetha S and Ms. Thasnim K M, our project coordinators, for their unwavering support and constructive feedback, which have been instrumental in the smooth progress of our project. Additionally, We are deeply grateful to Dr. Pramod Pavithran, Head of the Division of Computer Science and Engineering, for his continuous support and guidance throughout this experience. His leadership has been key to overcoming challenges during the Project design phase, and we appreciate his dedication to fostering the success of students within the department.

Akshay K R (20221094)
Nandulal Krishna (20221097)
Pravaal B Nath (20921038)
Rahul R (20221098)

Abstract

In the digital economy, the surge in financial transactions has led to a corresponding rise in sophisticated fraud schemes. This project focuses on the development of a Security Information and Event Management (SIEM) system designed for real-time fraud detection across banking systems, payment gateways, and enterprise financial networks. The system consolidates extensive datasets using advanced event correlation techniques to identify suspicious activities. Leveraging hybrid AI modeling, it combines rule-based engines with machine learning models such as unsupervised clustering, anomaly detection, and deep learning-based fraud detection to identify deviations from typical transaction behaviors and anomalies in network traffic.

To enhance detection capabilities, the SIEM integrates external threat intelligence feeds, including Indicators of Compromise (IoCs), malware signatures, and dark-web monitoring insights. These are cross-referenced with real-time logs to deliver contextual analysis, improving alert precision and reducing false positives. Behavioral analytics and risk-based scoring further ensure that high-risk transactions trigger immediate alerts, enabling proactive fraud mitigation.

Designed for scalability and efficiency, the system is deployed in a cloud-native environment with a microservices-based architecture, ensuring seamless integration with existing financial infrastructures. It leverages distributed processing frameworks such as Apache Kafka and Flink to handle millions of events per second, enabling real-time threat detection in high-volume environments. Security is reinforced through encrypted communication protocols and zero-trust policies, ensuring data integrity and compliance with financial regulations.

Real-time alerts are directed to Security Operations Centers (SOCs) for rapid response, while automated incident response mechanisms—such as transaction blocking and account freezing—help contain fraudulent activities. This project aims to provide a robust, AI-powered fraud detection solution tailored

for financial institutions and regulatory agencies, addressing the growing challenge of financial fraud in an increasingly digital landscape.

Contents

List of Figures

Chapter 1

Introduction

In today's digital economy, financial institutions face increasing threats from fraudulent activities. As the volume of transactions increases, so does the complexity and sophistication of the fraud attempts. This project aims to address these challenges by designing and implementing a Security Information and Event Management (SIEM) system specifically tailored to detect financial fraud in real-time. The system leverages large-scale data aggregation from various financial systems, processing and correlating events to flag suspicious activities. By providing immediate insights into potentially fraudulent transactions, the system plays a critical role in maintaining the integrity of financial operations.
To enhance its detection capabilities, the SIEM solution integrates external threat intelligence feeds and advanced machine learning algorithms, focusing on anomaly detection and real-time alerts. These features ensure quick responses to threats while maintaining scalability and high performance to manage the vast number of transactions processed in modern financial ecosystems. In addition, the system is built with compliance in mind, helping organizations comply with financial regulations while reducing the risk of fraud. This project promises to deliver a secure and efficient fraud detection mechanism that will benefit banking institutions and regulatory bodies, providing them with the tools needed to combat the ever-evolving landscape of financial fraud.

1.1 Objective

1. Design and implement a Security Information and Event Management (SIEM) solution tailored to detect financial fraud in real-time.

2. Integrate advanced machine learning algorithms and threat intelligence for improved detection accuracy.
3. To ensure compliance with financial regulations and reduce the number of fraud incidents within financial institutions.

1.2 Methodology

1. Project Planning:
   The project aims to design a robust SIEM system capable of aggregating and correlating data from multiple sources, including financial institutions and external threat intelligence feeds, to identify fraudulent activities effectively. It will incorporate machine learning for anomaly detection, enhancing its ability to recognize suspicious patterns and potential threats. Additionally, the system will provide real-time alerts and responses to counteract detected fraudulent transactions, ensuring swift mitigation. Scalability optimization is a key focus to accommodate growing data volumes and evolving security challenges. Furthermore, the project emphasizes on-time delivery, regular maintenance, and self-training capabilities to ensure the system remains future-proof and continuously improves its effectiveness.
2. System Design:
   The system is designed with multiple modules working in tandem to provide a fast and responsive fraud detection mechanism. It processes transaction data provided by client institutions, ensuring accurate and real-time monitoring. A cloud-based storage system is integrated to facilitate easy access and seamless integration with various platforms. Industry-standard tools are employed to implement robust machine learning algorithms for anomaly detection, enhancing the accuracy and efficiency of fraud identification. Additionally, the system features a responsive front-end interface for both users and administrators, enabling intuitive access and management. An API is also provided for integration with various networks and endpoints, ensuring flexibility and interoperability across different systems.
3. Technology Setup:

The development process begins with setting up the environment for React.js and AWS S3 to ensure a smooth workflow. This involves installing the necessary libraries and dependencies for React.js, including React Router for efficient navigation and Axios for handling API calls. Additionally, the backend implementation requires configuring an AWS S3 bucket, Azure Databricks, and Tableau. This setup includes the installation and configuration of ODBC drivers to facilitate seamless data connectivity and integration, ensuring efficient data processing and visualization.

4. Frontend Development:
   The user interface will be developed using React.js components, following the design guidelines. Separate interfaces will be created for users and administrators.

5. Backend Development:
   The AWS S3 bucket will be set up for dynamic data access in Azure Databricks to facilitate preprocessing and machine learning implementation. Backend functionalities will be implemented, including training and deploying machine learning models and setting up the API.

6. Integration and Testing:
   The frontend and backend components will be integrated to ensure smooth communication and data synchronization. Unit testing will be performed for individual components, followed by integration testing to verify the functionality of the entire system. Additionally, the application will be tested on multiple devices and browsers to ensure cross-device compatibility and responsiveness.

1.3 Features

1.3.1 Real-time Transaction Monitoring

The SIEM system monitors financial transactions in real-time to detect potential fraud. It aggregates data from various sources, including financial systems.

1.3.2 Hybrid AI Model for Anomaly Detection

The SIEM system incorporates a hybrid AI approach, combining machine learning (ML) and artificial neural networks (ANNs) to enhance fraud detection. ML algorithms identify suspicious transaction patterns, while ANNs analyze complex relationships in financial data to detect subtle anomalies. The model continuously evolves by learning from emerging fraud patterns, ensuring adaptive and proactive fraud detection.

1.3.3 Integration with External Threat Intelligence

The system integrates external threat intelligence feeds to enhance its detection capabilities. It uses these feeds to identify potential threats and enrich the detection process, ensuring up-to-date fraud indicators.

1.3.4 Fraud Alerts and Response

When suspicious activities are detected, the system generates real-time fraud alerts. These alerts are sent to compliance officers and system administrators, who can then take corrective actions, such as blocking transactions or investigating further.

1.3.5 User and Admin Interfaces

Both end-users (customers) and administrators have access to a responsive front-end interface. Users can submit transactions, while administrators can monitor and review fraud alerts, train machine learning models, and take corrective actions.

1.3.6 System Scalability and Performance

The solution is designed to handle large volumes of transactions and maintain high performance, ensuring that it can scale according to the needs of different financial institutions.

Chapter 2

Literature Review

2.1 Enhancing Transaction Fraud Detection with a Hybrid Machine Learning Model. Zhao, Xin & Zhang, Qiong & Zhang, Chang. (2024).

This paper addresses the growing complexity of financial transaction fraud detection and introduces a hybrid model combining LightGBM and Keras neural networks, enhanced by Focal Loss to handle imbalanced datasets. The proposed system improves both precision and adaptability, particularly in real-time scenarios where fraudulent transactions are rare but highly impactful. The ensemble approach leverages LightGBM's gradient boosting and the deep learning capabilities of Keras to outperform traditional models in terms of accuracy and robustness. The research also highlights the need for continuous evolution in fraud detection strategies to counter emerging fraud tactics effectively.

2.2 Security Information Event Management Data Acquisition and Analysis Methods with Machine Learning Principles.Tendikov, Noyan & Leila, Rzayeva & Saoud, Bilal & Shayea, Ibraheem & Bin Azmi, Marwan & Myrzatay, Ali & Alnakhli, Mohammad. (2024).

This paper explores the integration of artificial intelligence (AI) and machine learning (ML) into SIEM systems, focusing on enhancing network intrusion detection and cybersecurity strategies. Various machine learning techniques, such as classification, clustering, and text vectorization, are employed to detect and mitigate cybersecurity threats. The study also demonstrates the

effectiveness of models like Random Forest classifiers in identifying security anomalies with high accuracy, showcasing the synergy between ML algorithms and SIEM systems for proactive threat detection.

2.3 Adaptive Fraud Detection Systems: Real-Time Learning from Credit Card Transaction Data.Ahmad Amjad Mir.(2024)

This paper examines adaptive fraud detection systems that use machine learning to process credit card transaction data in real-time. It emphasizes the inadequacies of static, rule-based systems, proposing dynamic approaches using anomaly detection and reinforcement learning. The adaptive models adjust to changing fraud patterns, allowing financial institutions to detect fraudulent transactions more quickly and accurately, reducing financial losses and enhancing customer trust. The system effectively handles the concept of drift in fraud patterns by continuously learning from incoming data .

2.4 Using the SIEM Software Vulnerability Detection Model Proposed. In-seok Jeon, Keun-hee Han, Dong-won Kim, Jin-yung Choi

The paper proposes a model for software vulnerability detection using SIEM (Security Information and Event Management) systems. It describes how the integration of vulnerability databases like CVE, CPE, CCE, and CVSS enhances the detection process. The model effectively correlates data from various assets, analyzing software vulnerabilities and responding to potential threats by leveraging big data analytics. The proposed solution is tested and found to improve the detection rate of vulnerabilities compared to traditional methods, reducing false positives while ensuring rapid response to software threats.

2.5 Why SIEM is Irreplaceable in a Secure IT Environment? O. Podzins and A. Romanovs

This paper discusses the critical role of SIEM (Security Information and Event Management) in modern cybersecurity, emphasizing its ability to de-

tect anomalies and cyberattacks across an organization's IT infrastructure. It highlights SIEM's role in integrating and analyzing security logs from firewalls, intrusion detection systems, and Active Directory logs to provide real-time threat intelligence and automated remediation. The study underscores that while SIEM enhances security monitoring and compliance, its effectiveness depends on proper optimization, log management, and integration with a Security Operations Center (SOC). Challenges such as high costs, false positives, and resource-intensive maintenance are noted, but the paper concludes that SIEM remains an indispensable cybersecurity tool for enterprises seeking proactive threat detection and response.

2.6 Identifying Fraudulent Credit Card Transactions Using AI. H. Cheddy and R. K. Sungkur

The paper Identifying Fraudulent Credit Card Transactions Using AI examines machine learning techniques for fraud detection, tackling challenges like data imbalance and high verification costs. Using a synthetic dataset, it evaluates KNN, Decision Tree, Random Forest, and XGBoost, with XGBoost and Decision Tree emerging as the most accurate and reliable models. Techniques like StratifiedKFold Cross-Validation and SMOTE improved performance, while hyperparameter tuning optimized the XGBoost model, which was then deployed via a Flask-based web application. The study underscores the need for real-world validation to enhance model robustness and generalizability.

2.7 A Survey on Detection of Fraudulent Credit Card Transactions Using Machine Learning Algorithms. A. N. Ahmed and R. Saini

This paper reviews various machine learning techniques for detecting fraudulent credit card transactions, emphasizing the challenge of data imbalance in fraud detection. It analyzes methods used in multiple studies, comparing the effectiveness of algorithms such as Random Forest, Logistic Regression, SVM, Naïve Bayes, XGBoost, and KNN. The findings suggest that ensemble methods like XGBoost and Random Forest outperform other models in fraud classification, delivering high accuracy and recall. Additionally, resampling

techniques like SMOTE and undersampling are highlighted as crucial for improving model performance on imbalanced datasets. The survey concludes that a combination of advanced machine learning algorithms and effective data preprocessing significantly enhances fraud detection accuracy, with further improvements possible through neural networks and deep learning models.

2.8 AI-Powered Fraud Detection in the Financial Services Sector: A Machine Learning Approach. M. Marripudugala

This paper examines the use of machine learning techniques for fraud detection in financial transactions, focusing on Logistic Regression, Decision Tree, and Multi-Layer Perceptron (MLP) models. Using a synthetic dataset of mobile money transactions, the study evaluates the effectiveness of these models in identifying fraudulent activities. The results indicate that Logistic Regression struggled with imbalanced data, MLP had high precision but low recall, and Decision Tree provided a more balanced performance. The findings highlight the importance of fine-tuning models and selecting appropriate techniques to enhance fraud detection accuracy. The study also discusses behavioral analytics, anomaly detection, and AI-driven strategies to improve financial security.

2.9 Comparative Performance of Random Forest versus Gradient Boosting Machines in Detecting Financial Fraud. R. Sharma and D. Minhas

This study evaluates the effectiveness of Random Forest (RF) and Gradient Boosting Machines (GBM) in detecting financial fraud, addressing the complexity and scale of fraudulent transactions. Using a comprehensive dataset of financial transactions, both models demonstrated strong performance, with GBM achieving higher accuracy and an AUC-ROC of 0.95, compared to 0.92 for RF. The study highlights GBM's advantage in precision and adaptability, though it comes at a higher computational cost. While RF provides faster predictions and easier scalability, GBM excels in handling complex fraud patterns through sequential error correction. The findings suggest that ensemble

models like RF and GBM offer versatile and efficient solutions for fraud detection, though financial institutions must balance accuracy and computational demand based on operational requirements.

Chapter 3

System Analysis

3.1 Existing System

Blockchain technology has been increasingly adopted in financial systems for its ability to provide secure, transparent, and decentralized transaction tracking. Many financial institutions and cryptocurrency platforms have integrated blockchain-based solutions to enhance fraud detection mechanisms. By using the immutable ledger feature, blockchain ensures that every transaction is traceable and cannot be altered, providing a robust audit trail. This transparency makes it harder for fraudsters to manipulate transaction records without detection. Additionally, smart contracts, which automate the execution of transactions when predefined conditions are met, reduce the possibility of manual errors and fraudulent interventions.

However, despite these advancements, blockchain implementations still face several limitations when it comes to detecting and preventing fraud. One of the key inadequacies is the inability of blockchain systems to offer real-time fraud detection at the scale needed for large financial systems. Most blockchain networks lack sophisticated machine learning algorithms that can dynamically identify evolving fraud patterns. This gap in adaptive fraud detection leaves them vulnerable to new and unforeseen attack vectors. Moreover, while blockchain provides transparency, it does not inherently integrate external threat intelligence, making it difficult to preemptively identify and respond to external threats or vulnerabilities discovered outside of the blockchain network.

Our proposed SIEM tool addresses the limitations of blockchain by integrating real-time monitoring, machine learning-based anomaly detection, and ex-

ternal threat intelligence. While blockchain ensures data integrity, the SIEM system aggregates and correlates data from multiple networks, providing a more comprehensive view of transaction activity. By using machine learning, the tool detects subtle fraud patterns that blockchain alone might miss. Additionally, it offers real-time alerts and automated responses, allowing institutions to take immediate action against fraud. The tool also ensures regulatory compliance, making it a more scalable and responsive solution for financial and cryptocurrency fraud prevention.

3.2 Proposed System

The proposed system introduces a cutting-edge approach to financial and cryptocurrency fraud detection, offering a comprehensive, technology-driven platform that enhances security monitoring and threat detection. Built on the principles of real-time response, machine learning insights, and integration with external threat intelligence, the system is designed to strengthen fraud detection capabilities across financial institutions and cryptocurrency networks.

At its core, the system leverages a new hybrid artificial intelligence model that combines the robustness of machine learning algorithms like XGBoost with the adaptability and accuracy of artificial neural networks, such as a feedforward neural network. By integrating a machine learning model with an artificial neural network, several drawbacks and constraints of applying SIEM ideology to fraud detection can be overcome. While ML models provide excellent feature-based classification, ANNs excel at capturing nonlinear patterns and subtle interactions within the dataset. The neural network enhances adaptability to new and unseen fraud trends—truly embodying the essence of SIEM ideology.

The advanced hybrid model is further enhanced by the Updater feature of XGBoost models, which enables incremental learning by updating the model with new data without requiring a complete retraining from scratch. This capability is particularly advantageous as it significantly reduces both the training time and computational resources needed. Initial model training can be highly resource-intensive, especially when dealing with large datasets and complex feature sets. By allowing the model to adapt to evolving trends in real-time, the Updater feature ensures that the model remains accurate

and relevant while optimizing operational efficiency.
The platform is built on a robust technology stack, utilizing Flask and Django for API development, ensuring scalability, security, and efficiency. The system's objectives are to improve fraud detection accuracy, ensure regulatory compliance, and provide a responsive, real-time fraud prevention solution for financial institutions and cryptocurrency platforms.

Chapter 4

System Study

4.1 Software Requirements Specification

The Software Requirements Specification (SRS) is a detailed document that defines both the functional and non-functional requirements of the placement website. It serves as a roadmap for the development team, providing a clear understanding of the system's desired features and guiding the implementation process.

4.1.1 Purpose

The purpose of this project is to develop a real-time fraud detection system that integrates advanced hybrid AI models and external threat intelligence to monitor financial and cryptocurrency transactions. This system aims to enhance fraud detection accuracy, ensure regulatory compliance, and provide timely alerts for proactive fraud prevention.

4.1.2 Scope

The scope of this project involves the design, development, and deployment of a Security Information and Event Management (SIEM) system for real-time detection of financial and cryptocurrency fraud. The system will integrate hybrid intelligence algorithms for anomaly detection, external threat intelligence feeds, and real-time alert mechanisms. It will support multiple financial institutions and analysis platforms, providing scalable, secure, and responsive fraud monitoring. Additionally, the system will include automated compliance reporting features to ensure adherence to financial regulations. The

project will focus on creating a centralized hub for monitoring, fraud detection, and compliance management, ensuring adaptability across diverse financial infrastructures.

4.1.3 Project Perspective

From a product perspective, the proposed fraud detection system represents a groundbreaking platform that revolutionizes financial and cryptocurrency fraud prevention. By integrating an advanced in-house developed artificial intelligence algorithm and external threat intelligence, it offers a comprehensive solution for financial institutions, cryptocurrency exchanges, and regulatory bodies. The system's user-friendly interface ensures seamless interaction, providing real-time alerts, compliance tools, and in-depth data analysis for proactive fraud management. This platform not only enhances the accuracy and efficiency of fraud detection but also fosters a secure and regulated financial environment, setting a new standard for technology-driven solutions in fraud prevention and compliance management.

4.1.4 Use case Diagrams

The following diagrams 4.1 and 4.2, illustrate the use cases for different user roles within our tool. They provide a comprehensive view of how administrators and end users engage with the system to implement SIEM for financial fraud detection and analysis. Refer to both diagrams to understand the interactions and responsibilities of administrators and users in leveraging the tool's capabilities for fraud detection.

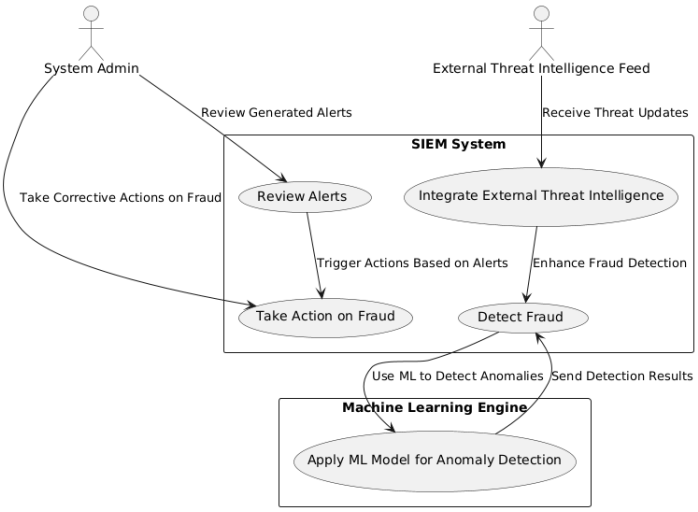

Figure 4.1: Use Case Diagram: Admin POV

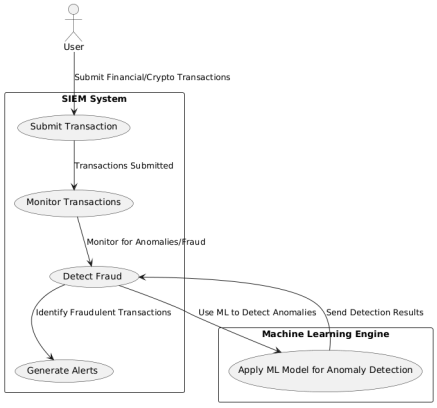

Figure 4.2: Use Case Diagram: User POV

4.2 Functional Requirements

1. **Real-time Fraud Detection:** The system shall monitor financial and cryptocurrency transactions in real-time and identify suspicious activities using machine learning algorithms.
2. **Data Collection:** The system shall collect transaction data from various sources such as blockchain networks, financial systems, and external threat intelligence feeds.
3. **Anomaly Detection:** The system shall apply machine learning models to detect anomalies and flag potential fraudulent transactions.
4. **Real-time Alerts:** The system shall generate real-time alerts for detected fraud, which will be sent to compliance officers and system administrators.
5. **Data Aggregation and Correlation:** The system shall aggregate data from multiple sources and correlate events to identify suspicious patterns across networks.
6. **Compliance Reporting:** The system shall generate automated compliance reports to meet financial regulatory requirements.
7. **User and Admin Access:** The system shall provide a responsive front-end interface for users and administrators to access and monitor transactions, view alerts, and interact with the system.
8. **External Threat Intelligence Integration:** The system shall integrate external threat intelligence feeds to enhance fraud detection capabilities.
9. **Data Visualization:** The system shall provide visual analytics and insights, including historical data on transactions, fraud patterns, and performance.
10. **Scalability and Performance:** The system shall handle high volumes of transactions and scale to support multiple financial institutions and cryptocurrency exchanges.

4.3 Non Functional Requirements

1. **Performance:** The system shall process and analyze large volumes of transactions in real-time without noticeable latency, ensuring minimal delay in fraud detection and alerts.
2. **Scalability:** The system shall be scalable to accommodate increasing transaction volumes and support multiple financial institutions and cryptocurrency exchanges.
3. **Security:** The system shall provide robust security measures, including data encryption, secure communication channels, and access control mechanisms to protect sensitive transaction data.
4. **Reliability:** The system shall maintain high availability, ensuring continuous operation with minimal downtime, even in the event of hardware or software failures.
5. **Compliance:** The system shall comply with financial and regulatory standards, ensuring that all data handling and reporting adhere to legal requirements.
6. **Usability:** The user interface shall be intuitive and easy to use for both administrators and users, requiring minimal training for effective operation.
7. **Maintainability:** The system shall be designed with modular components to facilitate ease of maintenance and updates, ensuring minimal disruption during system upgrades.
8. **Interoperability:** The system shall be capable of integrating with various external systems, including threat intelligence platforms, blockchain networks, and financial databases.
9. **Auditability:** The system shall log all transactions, alerts, and actions, providing a complete audit trail for investigation and compliance purposes.
10. **Extensibility:** The system shall be designed to allow future enhancements and the addition of new features without requiring significant changes to the existing architecture.

Chapter 5

System Requirements

5.1 Hardware Requirements

- **Processor:** Multi-core processors (Intel Core i7 or higher, AMD Ryzen 7 or higher) to handle large volumes of data processing and real-time transaction monitoring.
- **RAM:** Minimum 16 GB (32 GB recommended) to support high-performance machine learning models and data processing.
- **Storage:**
  - SSD with at least 500 GB for local processing and faster read/write operations.
  - Additional cloud storage (e.g., Amazon S3) for handling large datasets and logs.
- **Network:** High-speed internet connectivity for real-time data ingestion, threat intelligence updates, and cloud integration.
- **Graphics Card:** Optional but recommended for machine learning tasks requiring GPU acceleration (NVIDIA GTX 1660 or higher).

5.2 Software Requirements

5.2.1 Operating System

- Linux-based OS (Ubuntu, CentOS) for production servers.
- Windows 10/11 or macOS for local development environments.

5.2.2 Programming Languages

- **Python** (for machine learning, data processing, API development).
- **JavaScript** (for front-end development with React JS).

5.2.3 Frameworks and Libraries

- **Flask** and **Django** for building the web API and handling backend logic.
- **React JS** for the front-end, providing a responsive user interface.
- **Apache Kafka** for real-time data streaming and ingestion.
- **Apache Spark** for large-scale data processing.
- **TensorFlow** or **PyTorch** for machine learning model implementation.

5.2.4 Database

- **PostgreSQL** for structured data storage.
- **Elasticsearch** for real-time data querying and analysis.

5.2.5 Cloud Services

- **Amazon S3** for cloud-based storage and retrieval of large datasets.
- **AWS CloudTrail** for logging and monitoring of system activity.
- **AWS RDS** for hosting our PostgreSQL database online.

5.2.6 Visualization Tools

- **Chart.js** for data visualization and real-time monitoring dashboards.

5.2.7 Security Tools

- **IAM (Identity and Access Management)** for access control and security policies.

5.2.8 Version Control

- **Git** for version control and code collaboration.

Chapter 6

System Design

6.1 Introduction

Throughout this chapter, the design rationale and underlying ideology behind the proposed system have been discussed in detail. Implementing a Security Information and Event Management (SIEM) system for fraud detection is a complex task that requires meticulous integration of real-time data streams with advanced analytical frameworks. To effectively incorporate SIEM into fraud detection, live transaction data must be continuously ingested and processed in a serial manner before being analyzed by the hybrid machine learning model. This model is designed to generate accurate, actionable insights regarding both incoming and under-review transactions. The system architecture accounts for multiple user perspectives, with distinct administrative roles tailored to specific operational needs. A Network Administrator role is proposed for financial institution administrators, enabling them to oversee transaction integrity, monitor financial activity, and assess transaction characteristics in real time. Additionally, an API Administrator role is designated for overseeing the framework's operational stability, ensuring seamless system functionality, and maintaining overall system health. These structured roles enhance the system's reliability, enabling efficient fraud detection and mitigation while aligning with SIEM best practices.

6.2 System Architecture

The system architecture is designed with a layered approach to ensure scalability and efficiency. Data is collected from various financial institutions like American Express, JP Morgan & Chase as well as popular dataset archives

like kaggle, ingested via Kafka, and processed using Apache Spark, with storage in Amazon S3 and PostgreSQL. A robust hybrid artificial intelligence model is executed in Spark to process and provide seamless real time fraud detection. The service layer, utilizing Flask and Django, manages business logic and APIs, while the web application layer, built with React JS and Flask, provides a responsive user interface. This modular design enables real-time processing and seamless data flow, optimized for fraud detection in financial and cryptocurrency networks.

The figure 6.1 below illustrates the modular data flow through the various stages of our tool, utilizing industry-standard technologies for data processing, scraping, and building machine learning + artificial neural network solutions for fraud detection. Additionally, we employ Flask to create APIs for endpoint integration, further enhancing the tool's functionality and efficiency. PostgreSQL and Amazon S3 bucket are used to ensure that the data processed and used throughout the implemented framework follow the ACID properties. This ensures a reliable and trustworthy framework.

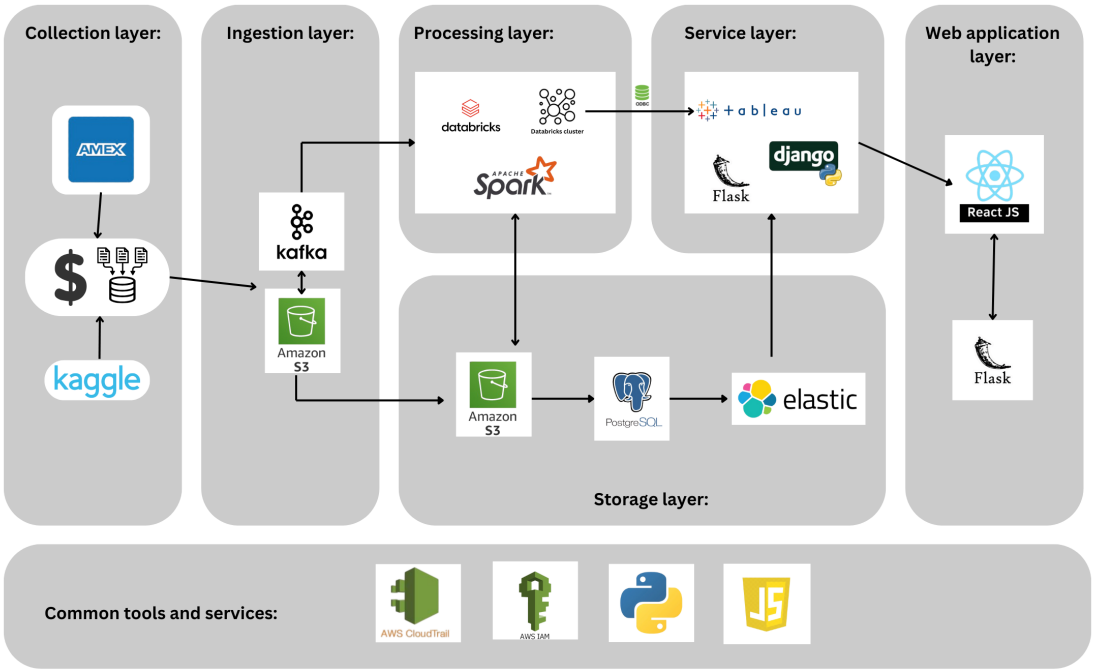

Figure 6.1: System Architecture Diagram

6.3 Database Design

For the implementation of this framework, the combination of PostgreSQL and Amazon AWS S3 bucket have been chosen. PostgreSQL particularly plays an integral part in this model for accurate and stable operations. In order to circumvent the repeated training requirement for the machine learning algorithm to understand and update fraudulent trends, a local database is utilized to store the recent transactions. Once a suitable threshold (number of transactions) is reached, the machine learning and artificial neural networks are retrained to ensure that both models capture the latest trends and variations in the data. For the administrator viewpoint, a more permanent database is maintained for storing the latest transactional data for visualization and suitable data analysis.
In order to achieve the portability of the framework, the entire implementation must be executed over cloud platform and resources. This is where Amazon AWS S3 bucket comes into play. It is used to ensure data is available for the model as well as various users at any point and from any location.

6.4 AI Logic Flow

One of the key unique selling points (USPs) of this framework is its seamless integration of a highly customized and purpose-built AI logic tailored specifically for this use case. From an external user's perspective, the model operates as a "black box" AI, delivering the desired output based on the provided input with minimal complexity. As illustrated in Fig. 6.2, the model, nicknamed **FinSpy**, functions like a large language model meticulously designed for financial applications. However, a deeper exploration into the model's internal mechanics reveals the true brilliance of the "model," showcasing its sophisticated architecture and advanced analytical capabilities.

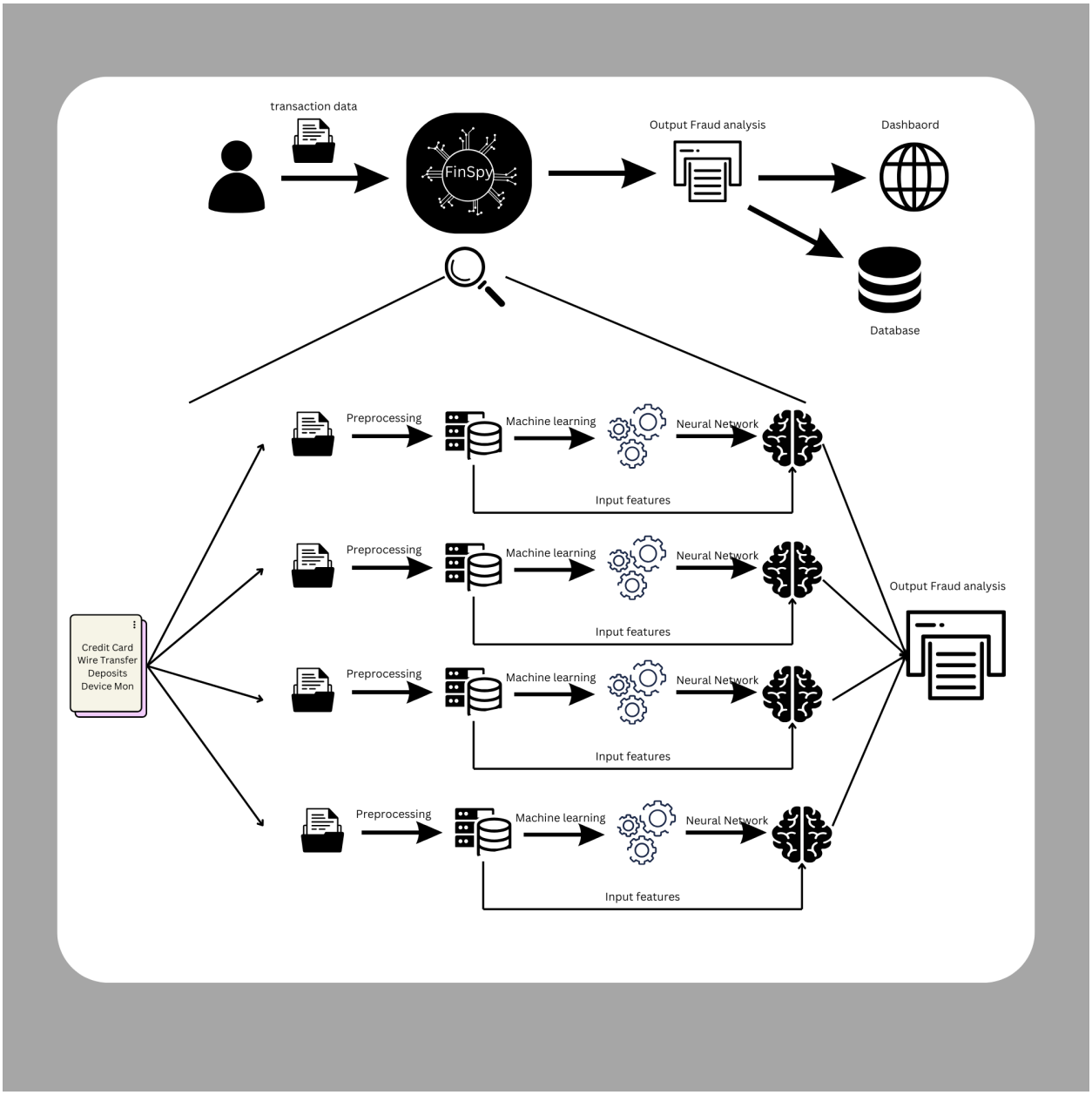

Figure 6.2: AI Logic Flow

The model is comprised of several layers of independent "AI logic" layers, each responsible for generating the output expected by the user. These layered "AI logic" components work in harmony, ensuring that the model not only processes input accurately but also adapts dynamically to different transaction types, providing robust and reliable performance across various financial scenarios. For our implementation of the framework, we have taken into consideration four of the major transactions undertaken by any bank.

- Bank Deposits and Withdrawal transactions.
- Credit Card transactions.
- Wire transfer transactions.
- Bank registration device monitoring

To expand on each logic layer, let us examine an example data flow within the framework. Beginning with the training phase, a dataset is input into the framework. Depending on the transaction type, the data is directed into one of several dedicated processing pipelines. Upon entering the pipeline, the dataset undergoes an essential preparation phase before being fed into the models.

This preparation involves executing multiple data cleaning and preprocessing scripts over the dataset. The preprocessing is conducted through a structured three-tier approach: bronze, silver, and gold levels, each enhancing the data quality progressively. Additionally, feature engineering techniques are applied to introduce new features or attributes to the dataset, enriching the input data for better model performance.

The most critical aspect of data processing is managing **Class Imbalance**. Given the nature of the dataset, there is a risk that the models might overlook classes or features with smaller representation. This oversight can lead to underfitting and hallucinations within the models. To address this, the framework employs various oversampling and undersampling techniques to maintain class balance within the dataset.

For our implementation, we prioritized oversampling techniques. After thorough research and experimentation, the SMOTE (Synthetic Minority Over-sampling Technique) balancing method was identified as the most effective approach. Consequently, SMOTE is applied to the dataset, ensuring balanced class distribution and enhancing the model's ability to generalize effectively across different transaction types.

The final step before training the model involves encoding categorical data and scaling the dataset to ensure optimal conditions for model training. The framework utilizes LabelEncoding for categorical data transformation and Standard Scaling to normalize the data. These encoders and scalers are stored as pickle files, ensuring consistency across all stages of the pipeline. This approach guarantees that any data ingested at any point in the pipeline

undergoes the same preprocessing steps, maintaining uniformity and reliability in data handling.

Once preprocessing is complete, the dataset is split into training and testing sets and then fed into the machine learning (ML) model. The model generates a fraud prediction, which is then concatenated with the input dataset and passed into the artificial neural network (ANN) as the next phase of the pipeline.

Machine learning models excel at identifying existing trends across large datasets, but they often struggle to recognize new patterns or anomalies that emerge for the first time in financial transactions. Their reliance on historical data makes them less adaptable to the fast-paced and dynamic nature of financial operations. This is where ANNs add significant value. Unlike traditional ML models, ANNs are better suited to detect emerging trends in data. Their inherent "lack of memory" is effectively mitigated by the ML model's strength in historical trend recognition, creating a balanced and robust hybrid model ideal for financial use cases.

After the ANN completes its operations, the processed dataset is stored in the database. This dataset serves a dual purpose: it is made available to the user or client through the web application, and it is also integrated with Tableau for dynamic and interactive data visualization. This seamless flow of data from preprocessing through predictive analytics to visualization ensures both accuracy in fraud detection and clarity in presenting insights to stakeholders.

Chapter 7

Methodology

This chapter details the methodologies employed in the development of the SIEM-driven financial fraud detection framework. It describes the various modules used, their functionalities, and how they contribute to the overall system.

7.1 Modules Used

The system is composed of multiple interconnected modules, each designed to facilitate seamless data pre-processing, feature engineering, data visualization, model training, and other essential processes. Each module plays a distinct role in optimizing efficiency and enhancing the overall user experience.

7.1.1 Data Pre-processing and Feature Engineering

This module processes the provided data to ensure its compatibility with the model's expected input. To achieve this, several validation and transformation steps are performed. Additionally, feature engineering is applied to extract and generate implicit information necessary for enhancing model performance. The key pre-processing steps and feature engineering techniques employed include training the model with data taken in as CSV and parquet files, decoded using csv-reader and FastParquet libraries. All NaN data entries in the dataset are replaced with 0. Data encoding and scaling are performed for optimal model performance, and irrelevant data like mobile numbers and emails are dropped.

7.1.2 Machine Learning and Artificial Neural Network Model Training

This module implements the backend fraud detection logic for the framework, designed to identify fraudulent activities across four primary data types: credit card transactions, wire transfers, deposits and withdrawals, and device monitoring. The detection mechanism is powered by a custom financial fraud detection model, which integrates four specialized hybrid ML-DL models. Following extensive research into the most suitable models for the framework's requirements, a unique combination of the XGBoost machine learning model and a Feedforward Artificial Neural Network (ANN) has been selected to optimize fraud detection accuracy and efficiency. The input stream from the pre-processing and feature engineering module is provided to the XGBoost model in the form of a pandas dataframe. Preloaded custom scalers and encoders ensure that transaction data aligns with the expected model input. The XGBoost model processes the data and generates a probabilistic fraud detection score. The XGBoost model's output, along with the original input data, is then fed into the Feedforward ANN, which produces a final fraud probability to support decision-making.

7.1.3 Model updation and retraining

This module manages the retraining of both the ML and ANN models to maintain optimal fraud detection performance. As new transactions are processed, they may introduce emerging trends that the ML model has not yet learned. Additionally, due to the absence of memory retention, the ANN model may forget previously learned patterns over time. To mitigate both issues, this module ensures periodic retraining using the latest transaction data. Based on extensive research, the retraining process is triggered every 50 new transactions, striking a balance between preventing over-frequent updates and avoiding infrequent training that could lead to model drift. The retraining process follows a structured approach to ensure model adaptability and accuracy:

- Incoming transactions are stored in a local PostgreSQL database as new entries.
- When the transaction count exceeds 50, an update script is triggered to retrain both the ML and ANN models using the latest data.

- After retraining is completed, the local database is cleared, and normal operations resume seamlessly.

This mechanism ensures that the models continuously adapt to evolving fraud patterns while maintaining efficiency and preventing unnecessary retraining.

7.1.4 Authentication and User Management

To ensure secure authentication, the system integrates Auth0, leveraging its advanced features such as social logins and multi-factor authentication. This module provides robust login and signup functionalities through Auth0's authentication services, maintaining a high standard of security. It employs token-based authentication to verify user identities effectively, ensuring that only authorized individuals gain access to the platform. Additionally, the module implements role-based access control, distinguishing between client administrators and framework administrators to manage permissions and maintain a secure and organized user management system.

7.1.5 Client Dashboard

This module, leveraging an API, provides a comprehensive overview of transaction flows within the client's system. It enables access to historical transaction data, facilitates the submission of new transactions for fraud assessment, and generates risk ratings. Additionally, dynamic and descriptive visualizations, including interactive graphs, enhance transaction analysis. These insights support in-depth system studies and assist in developing effective fraud prevention strategies. An additional section is incorporated to deliver AI-driven insights derived from transactional data, providing clients with actionable intelligence. These insights empower clients to make informed decisions, execute appropriate actions, and implement necessary responses based on the predictions generated by the AI model. By leveraging these analytical outputs, businesses can enhance their fraud detection mechanisms, optimize operational strategies, and improve overall risk management. The user dashboard offers the following key features:

- Secure integration with the Flask API for seamless fraud detection data transmission.
- Comprehensive data analysis and risk assessment for informed decision-making.

- Dynamic and interactive visualizations using `Tableau` to enhance interpretability.
- Robust data security measures implemented via PostgreSQL and a custom-defined security strategy.
- AI driven insights derived from the transactional data in the database.

7.1.6 Framework Admininstrator Dashboard

This module provides the administrator with a comprehensive overview of the framework's operation, ensuring the system functions as expected. It includes API monitoring, logging, and access control, granting the admin full oversight of system activities. Additionally, built-in debugging and diagnostic tools facilitate quick and efficient issue resolution, ensuring seamless framework performance.

7.1.7 Backend API Handling

This module is responsible for managing all interactions between the frontend and backend, ensuring a smooth and efficient data flow throughout the application. Its primary responsibilities include handling API requests for fraud detection and retrieving results in real-time. It also manages user inputs, such as information requests and graph updates, maintaining a responsive and dynamic user interface. Additionally, the module implements robust error-handling mechanisms to prevent data loss and enhance overall system stability, contributing to a reliable and seamless user experience.

7.1.8 Database

PostgreSQL is chosen for database implementation due to its robustness, scalability, and advanced security features. To ensure redundancy, security, and efficient data management, multiple databases are utilized within the system.

- **Local Database:** Maintains data for managing model retraining and capturing evolving data trends. It temporarily stores transaction data and triggers retraining processes when predefined thresholds are met, ensuring models remain adaptive to emerging fraud patterns.

- **Client-Specific Database:** Allocated to each client, this database provides a dedicated repository for transaction data. It enables the generation of customized dashboards, offering clients a comprehensive and real-time overview of their financial activities, fraud assessments, and risk analytics.
- **Master Database:** Maintained within the framework's core implementation system, this centralized and secure repository ensures redundancy and seamless recovery in case of failures. It also stores critical logs and system-wide fraud patterns, enhancing security measures.

By distributing data across these databases, the system achieves high availability, optimized performance, and enhanced fraud detection capabilities while maintaining strict access control and data integrity.

7.1.9 Data Visualisation

This module offers a dynamic and interactive graphical representation of data, enabling detailed analysis and system evaluation from the client's perspective. Transactional data stored in the client-specific PostgreSQL database is seamlessly retrieved and processed for visualization using Chart.js, a lightweight JavaScript library. The data is transformed into meaningful visualizations, including line charts, bar graphs, pie charts, and radar charts, providing valuable insights into transaction patterns, risk trends, and system performance. These interactive graphs are embedded directly into the client dashboard, ensuring a smooth and responsive user experience. By integrating real-time and historical data analytics, this module enhances decision-making, aiding in fraud detection and risk management. Features such as dynamic updates, tooltips, filtering, and drill-down capabilities allow users to explore data in greater depth, making the system more intuitive and effective.

7.1.10 Navigation and Routing

The platform leverages React.js routing to facilitate seamless navigation between various sections of the application. This module enhances both efficiency and security during transitions by incorporating several key features. It employs React Router's useNavigate hook to manage navigation and redirection based on the user's authentication status. Additionally, it implements protected routes using React Router and context-based authentication, ensur-

ing that only authorized users can access restricted sections of the application. These mechanisms contribute to smooth and efficient transitions between different modules, significantly improving the overall user experience.

7.1.11 Error Handling and Notifications

To enhance usability and provide meaningful feedback to users, this module incorporates several key features. It offers real-time alerts for file upload errors, API failures, and authentication issues, ensuring that users are immediately informed of any disruptions. Additionally, graceful fallback mechanisms are implemented to minimize disruptions during fraud probability detection, maintaining a smooth user experience even in unexpected scenarios. The module also includes robust logging and debugging tools, enabling effective system performance monitoring and facilitating the quick identification and resolution of potential issues.

The integration of these modules ensures that the **FinSpy** framework operates holistically to achieve exceptional fraud detection accuracy, regardless of emerging fraud trends or variations in data entry techniques. By leveraging the combined strengths of machine learning and artificial neural networks, the framework maintains adaptability and precision. The machine learning models provide stability by recognizing established patterns, while the neural networks excel at identifying novel trends. This synergy ensures that **FinSpy** remains a robust and reliable solution, delivering consistent performance even as financial transaction patterns evolve.

Chapter 8

System Implementation

This chapter presents a comprehensive walkthrough of the system implementation for the proposed framework, aligned with the architectural blueprint outlined in previous chapters. The research and development phases have been meticulously executed with a strong emphasis on reliability, fault tolerance, scalability, and seamless integration into existing enterprise ecosystems. The implementation strategy prioritizes robust performance and ensures that the framework adheres to industry best practices for software deployment and system resilience.

The framework is engineered for cloud-native deployment, eliminating the need for clients to invest in dedicated infrastructure solely for running this solution. By leveraging cloud-based infrastructure, the framework enables efficient resource utilization, high availability, and on-demand scalability. Additionally, it is designed to be platform-agnostic, allowing seamless integration with existing IT environments, thereby accelerating deployment timelines and reducing operational overhead.

8.1 System Implementation Architecture

The system architecture of FinSpy for implementation follows a modular, cloud-based implementation strategy designed for scalability, security, and reliability. It leverages a suite of cloud resources to enhance data security, accessibility, and operational robustness. Key components include an intuitive web dashboard, high-performance APIs, and advanced AI models that deliver accurate and actionable insights. This architecture ensures the efficient execution of the framework's objectives while maintaining optimal performance and resilience.

8.1.1 Architecture Components

- **Frontend:** Developed using React.js, providing a dynamic and intuitive dashboard for seamless user interaction and efficient execution of tasks. Mistral AI is also incorporated to provide AI driven insights based on the provided transactional data.
- **Backend:** Built with Apache Spark and Django, enabling advanced data processing, efficient API request handling, and comprehensive network and usage monitoring.
- **Database:** PostgreSQL is integrated to ensure secure and scalable data storage while seamlessly interfacing with cloud resources such as AWS S3 for efficient data management.
- **Data Visualisation:** PostgreSQL is integrated with chart.js to ensure secure and scalable data storage while seamlessly interfacing with cloud resources such as AWS S3 for efficient data management.
- **AI Processing:** Implemented using Keras and Scikit-learn, these in-house hybrid models combine the strengths of both machine learning and deep learning to deliver highly accurate and robust analytical insights.

8.2 Module Implementation

8.2.1 Data Preprocessing and Feature Engineering

Reliable datasets in CSV and Parquet formats undergo preprocessing before being passed to the model. This module ensures that only the most relevant features are selected for training, minimizing overfitting and noise. Apache Spark powers the preprocessing pipeline, implementing a structured three-tier strategy—Bronze, Silver, and Gold. To maintain security and client privacy, sensitive information such as client names and contact details is neither stored nor utilized. Additionally, PCA-based feature extraction is employed to enhance data security by reducing readability.

Before model training, dataset augmentation is performed to address class imbalances that could otherwise distort predictive accuracy. To achieve this, JP Morgan and Chase's proprietary QuantGant Neural Network model generates synthetic data that accurately reflects real dataset trends without in-

troducing noise. Furthermore, the SMOTE algorithm is applied for oversampling, ensuring balanced class distributions and improved model performance. The figure 8.1 depicts a preprocessing script employed to drop irrelevant dataset features and rename features as per the model requirements.

```python
import pandas as pd

def preprocess_data(data):
    #Drop irrelevant or sensitive columns
    data = data.drop(columns=['index', 'trans_date_trans_time', 'cc_num', 'first', 'last',
                              'street', 'city', 'state', 'zip', 'job', 'dob', 'trans_num'])
    data = data.fillna(0)
    return data

def fix_indexing(file_path):
    train_data = pd.read_csv(file_path)
    if train_data.columns[0] != 'index':
        train_data.rename(columns={train_data.columns[0]: 'index'}, inplace=True)
    output_path = file_path.replace('.csv', '_updated.csv')
    train_data.to_csv(output_path, index=False)
    print(f"The first column has been renamed to 'index' and saved to {output_path}.")

def print_features(file_path, data_dict):
    data = pd.read_csv(file_path)
    columns_to_drop = ['index', 'trans_date_trans_time', 'cc_num', 'first', 'last',
                       'street', 'city', 'state', 'zip', 'job', 'dob', 'trans_num']
    rem_cols = [col for col in data.columns if col not in columns_to_drop]
    print('Input features: \n')
    for col in rem_cols:
        print(f'{col}: {data_dict.get(col, "Feature description not available")}')

```

Figure 8.1: Data preprocessing

8.2.2 Hybrid AI model

Using the Gold-tier preprocessed dataset, a layered hybrid AI model is developed to detect fraudulent transactions by integrating XGBoost with a Feedforward Artificial Neural Network (ANN). Traditional machine learning models, such as decision trees and standalone XGBoost, struggle to adapt to unseen patterns, while ANNs excel at capturing nonlinear relationships but suffer from memory degradation. To mitigate these limitations, this hybrid approach combines XGBoost's structured feature learning with ANN's deep representation capabilities, enhancing both predictive performance and adaptability.

XGBoost efficiently extracts and selects important features, reducing noise and improving model interpretability, while the ANN refines these representations to identify complex patterns. Additionally, XGBoost's built-in regularization techniques help prevent overfitting, allowing the ANN to focus on deeper feature learning. This structured approach not only enhances model generalization but also ensures computational efficiency by leveraging XGBoost as a feature extractor before ANN training.

By first applying XGBoost to handle class imbalances and then training the ANN, the model maintains a balanced learning process. This methodology is widely applicable across finance, healthcare, and fraud detection domains, ensuring robust, scalable, and high-performing fraud detection capabilities.

Figures 8.2 to 8.6 illustrate the training processes of various machine learning models designed to analyze feature trends across different transaction categories. Given that each transaction type exhibits distinct characteristics and behavioral patterns, separate machine learning models are deployed to effectively capture and learn the unique feature distributions associated with each transaction type.

```python
#Dataset loading
data_file = "data1.csv"
df = pd.read_csv(data_file)

#Split target and features
target = 'fraud_bool'
X = df.drop(columns=[target])
y = df[target]

encoders = {}
#Preprocessing
#Encoding categorical variables
for col in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])
    encoders[col] = le
with open('encoders.pkl', 'wb') as en_file:
    pickle.dump(encoders, en_file)

#Missing values
X.replace(-1, np.nan, inplace=True)
X.fillna(X.mean(), inplace=True)

#Scaling
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
with open('scaler.pkl', 'wb') as sc_file:
    pickle.dump(scaler, sc_file)

#Splitting into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#GridSearchCV
xgb_model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='logloss', use_label_encoder=False)
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=3, scoring='accuracy', verbose=1)
grid_search.fit(X_train, y_train)

#Best model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

print("\nBest Parameters from GridSearchCV:")
print(best_params)

#Evaluate model
y_pred = best_model.predict(X_test)

pickle_file = "fraud_detection_model.pkl"
with open(pickle_file, 'wb') as f:
    pickle.dump(best_model, f)
print(f"\nModel saved as {pickle_file}")

```

Figure 8.2: XGBoost model for Bank Device monitoring

```python
#Dataset loading
data = pd.read_csv('data2.csv')

#Cleaning data
data.fillna(0, inplace=True)

#Creation of new features
data['balance_delta_orig'] = data['oldbalanceOrg'] - data['newbalanceOrig']
data['balance_delta_dest'] = data['oldbalanceDest'] - data['newbalanceDest']
data['transaction_ratio'] = data['amount'] / (data['oldbalanceOrg'] + 1)

#Categorical encoding
label_encoders = {}
for col in ['type']:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le

with open('label_encoders.pkl', 'wb') as f:
    pickle.dump(label_encoders, f)

#Targets
target = 'isFraud'
features = [col for col in data.columns if col not in ['isFraud', 'isFlaggedFraud', 'nameOrig', 'nameDest']]

X = data[features]
y = data[target]

#Imbalance correction
fraud = data[data[target] == 1]
non_fraud = data[data[target] == 0]

#Minority upsample
fraud_upsampled = resample(fraud, replace=True, n_samples=len(non_fraud), random_state=42)
balanced_data = pd.concat([fraud_upsampled, non_fraud])
X = balanced_data[features]
y = balanced_data[target]

#Train and test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

#Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
with open('trans_scal.pkl', 'wb') as f:
    pickle.dump(scaler, f)

#Model init and training
model = GradientBoostingClassifier(random_state=42)
model.fit(X_train, y_train)

#Evaluation
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
print("AUC-ROC:", roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]))

#Add to pickle jar
with open('fraud_detection_model.pkl', 'wb') as f:
    pickle.dump(model, f)
```

Figure 8.3: GBoost model for Wire transfer transactions

```
#Read dataset
train_csv = 'fraudTrain.csv'
test_csv = 'fraudTest.csv'
fix_indexing(train_csv)
fix_indexing(test_csv)
train_dat = pd.read_csv('fraudTrain_updated.csv')
test_dat = pd.read_csv('fraudTest_updated.csv')

train_dat = preprocess_data(train_dat)
test_dat = preprocess_data(test_dat)
print_features('fraudTrain_updated.csv', data_dict)

#Data split
X_train = train_dat.drop(columns=['is_fraud'])
y_train = train_dat['is_fraud']
X_test = test_dat.drop(columns=['is_fraud'])
y_test = test_dat['is_fraud']

#Encode and Scale
encoders = {}
cat_cols = X_train.select_dtypes(include=['object']).columns

for col in cat_cols:
    le = LabelEncoder()
    X_train[col] = le.fit_transform(X_train[col])
    X_test[col] = le.transform(X_test[col])
    encoders[col] = le

with open('encoders.pkl', 'wb') as en_file:
    pickle.dump(encoders, en_file)

```

Figure 8.4: XGBoost model for Credit card transactions

```
scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns)
X_test = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)

with open('scalar.pkl', 'wb') as sc_file:
    pickle.dump(scaler, sc_file)

#Apply SMOTE to balance the dataset
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)

#Initialise model
xg_model = XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    use_label_encoder=False,
    random_state=42
)

#Train model
eval_set = [(X_train, y_train), (X_test, y_test)]
results = {}
xg_model.fit(X_train, y_train, eval_set=eval_set, verbose=True)

#Predict model
y_pred = xg_model.predict(X_test)
y_pred_proba = xg_model.predict_proba(X_test)[:, 1]

#Save model
with open('cred_xgboost_model.pkl', 'wb') as file:
    pickle.dump(xg_model, file)
    print('Model saved successfully')
```

Figure 8.5: XGBoost model for Credit card transactions

```python
#Load the dataset
file_path = 'data/DirectDepositFraud_Data.csv'
data = pd.read_csv(file_path)

#Data Preprocessing
label_enc = LabelEncoder()
data['transactionType'] = label_enc.fit_transform(data['transactionType'])
data = data.drop(columns=['Step', 'startingClient', 'destinationClient'])
with open('dep_encoders.pkl', 'wb') as f:
    pickle.dump(label_enc, f)

X = data.drop('isfraud', axis=1)
y = data['isfraud']

#Scale numerical features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
with open('dep_scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)

#Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

#Split dataset
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

#Hyperparameter tuning
xgb = XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')

#Model Evaluation
y_pred_tuned = xgb.predict(X_test)

#Save the trained model
with open('dep_xgboost_model.pkl', 'wb') as f:
    pickle.dump(xgb, f)
```

Figure 8.6: XGBoost model for Deposit transactions

Figures 8.7 to 8.14 illustrate the implementation of artificial neural networks (ANNs) that operate as an additional abstraction layer over the machine learning models, forming a robust AI-driven hybrid framework. Similar to the ML models, each transaction type is assigned a custom-tailored ANN designed to dynamically capture real-time patterns from ingested data. This adaptive learning capability ensures that even previously unseen fraud patterns are effectively identified, enhancing the framework's ability to detect emerging threats with high precision.

```python
#Encode cat vars
with open('encoders.pkl', 'rb') as en_file:
    encoders = pickle.load(en_file)
for col, encoder in encoders.items():
    if col in X.columns:
        X[col] = encoder.transform(X[col])

#Handle missing values
X.replace(-1, np.nan, inplace=True)
X.fillna(X.mean(), inplace=True)

#Scale features
with open('scaler.pkl', 'rb') as file:
    scaler = pickle.load(file)
X_scaled = pd.DataFrame(scaler.transform(X), columns=X.columns)

#Split into train and test
X_train, X_val, y_train, y_val = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

#Class Imbalance Handling using SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

#XGBoost added as feature for hybrid model
with open('fraud_detection_model.pkl', 'rb') as file:
    xgb_model = pickle.load(file)
```

Figure 8.7: ANN for Bank device monitoring

```python
#Add XGBoost predictions as a feature
xgb_train_predictions = xgb_model.predict_proba(X_train_resampled)[:, 1].reshape(-1, 1)
xgb_val_predictions = xgb_model.predict_proba(X_val)[:, 1].reshape(-1, 1)

#XGBoost + input features
X_train_with_xgb = np.hstack((xgb_train_predictions, X_train_resampled))
X_val_with_xgb = np.hstack((xgb_val_predictions, X_val))

#Train ANN
input_shape = (X_train_with_xgb.shape[1],)
ann = build_ann(input_shape)
history = ann.fit(X_train_with_xgb, y_train_resampled, epochs=50, batch_size=32, validation_data=(X_val_with_xgb, y_val))

#Evaluate ANN
val_loss, val_accuracy = ann.evaluate(X_val_with_xgb, y_val)
print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}")

#Save ANN model
ann.save('ann_fraud_detection.keras')
print("ANN model saved as ann_fraud_detection.keras")
```

Figure 8.8: ANN for Bank device monitoring

```python
#Build ANN Model
def build_ann(input_shape):
    inp_layer = Input(shape=input_shape)
    x = Dense(64, activation='relu')(inp_layer)
    x = Dropout(0.3)(x)
    x = Dense(32, activation='relu')(x)
    x = Dropout(0.3)(x)
    out_layer = Dense(1, activation='sigmoid')(x)
    model = Model(inputs=inp_layer, outputs=out_layer)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

#Encoding
with open('label_encoders.pkl', 'rb') as f:
    label_encoders = pickle.load(f)
for col, encoder in label_encoders.items():
    if col in data.columns:
        data[col] = encoder.transform(data[col])

#Define Features and Target
target = 'isFraud'
features = [col for col in data.columns if col not in ['isFraud', 'isFlaggedFraud', 'nameOrig', 'nameDest']]
X = data[features]
y = data[target]

#Imbalance Correction (Resampling)
fraud = data[data[target] == 1]
non_fraud = data[data[target] == 0]

fraud_upsampled = resample(fraud, replace=True, n_samples=len(non_fraud), random_state=42)
balanced_data = pd.concat([fraud_upsampled, non_fraud])

X = balanced_data[features]
y = balanced_data[target]

#Scaling
with open('trans_scal.pkl', 'rb') as f:
    scaler = pickle.load(f)

X_scaled = pd.DataFrame(scaler.transform(X), columns=X.columns)
```

Figure 8.9: ANN for Wire transfer transactions

```python
#Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

#Load GB Model
with open('fraud_detection_model.pkl', 'rb') as f:
    gb_model = pickle.load(f)

#GB Predictions as Feature
gb_train_predictions = gb_model.predict_proba(X_train)[:, 1].reshape(-1, 1)
gb_test_predictions = gb_model.predict_proba(X_test)[:, 1].reshape(-1, 1)

#Combine GB Predictions with Input Features
X_train_with_gb = np.hstack((gb_train_predictions, X_train))
X_test_with_gb = np.hstack((gb_test_predictions, X_test))

#Convert to DataFrame
X_train_with_gb = pd.DataFrame(X_train_with_gb)
X_test_with_gb = pd.DataFrame(X_test_with_gb)

#Train ANN
input_shape = (X_train_with_gb.shape[1],)
ann = build_ann(input_shape)
history = ann.fit(
    X_train_with_gb.to_numpy(), y_train.to_numpy(),
    epochs=50, batch_size=32, validation_data=(X_test_with_gb.to_numpy(), y_test.to_numpy())
)

#Evaluate ANN
val_loss, val_accuracy = ann.evaluate(X_test_with_gb.to_numpy(), y_test.to_numpy())
print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}")

#Save ANN Model
ann.save('ann_bank_trans_fraud_detection.keras')
print("ANN model saved as ann_bank_trans_fraud_detection.keras")
```

Figure 8.10: ANN for Wire transfer transactions

```python
def build_ann(input_shape):
    inp_layer = Input(shape=input_shape)
    x = Dense(64, activation='relu')(inp_layer)
    x = Dropout(0.3)(x)
    x = Dense(32, activation='relu')(x)
    x = Dropout(0.3)(x)
    out_layer = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=inp_layer, outputs=out_layer)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Encoding and Scaling
with open('encoders.pkl', 'rb') as en_file:
    encoders = pickle.load(en_file)

for col, encoder in encoders.items():
    if col in X_train.columns:
        X_train[col] = encoder.transform(X_train[col])
        X_test[col] = encoder.transform(X_test[col])

with open('scalar.pkl', 'rb') as file:
    scaler = pickle.load(file)
X_train_scaled = pd.DataFrame(scaler.transform(X_train), columns=X_train.columns)
X_test_scaled = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)

# Apply SMOTE for balancing the dataset
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train)
```

Figure 8.11: ANN for Credit card transactions

```python
# Load XGBoost model
with open('cred_xgboost_model.pkl', 'rb') as file:
    xgb_model = pickle.load(file)

# Generate predictions from XGBoost for ANN input
xgb_train_predictions = xgb_model.predict_proba(X_train_resampled)[:, 1].reshape(-1, 1)
xgb_val_predictions = xgb_model.predict_proba(X_test_scaled)[:, 1].reshape(-1, 1)

# Concatenate XGBoost predictions with scaled features
X_train_with_xgb = np.hstack((xgb_train_predictions, X_train_resampled))
X_val_with_xgb = np.hstack((xgb_val_predictions, X_test_scaled))

# Build and train the ANN
ann = build_ann(input_shape=(X_train_with_xgb.shape[1],))
history = ann.fit(X_train_with_xgb, y_train_resampled, epochs=50, batch_size=32,
                  validation_data=(X_val_with_xgb, y_test))

val_loss, val_accuracy = ann.evaluate(X_val_with_xgb, y_test)
print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}")

ann.save('ann_cc_fraud_det.keras')
print('ANN model saved as ann_cc_fraud_det.keras')
```

Figure 8.12: ANN for Credit card transactions

```python
#Build ANN
def build_ann(input_shape):
    #Input layer
    inp_layer = Input(shape=input_shape)

    #Hidden layers
    x = Dense(64, activation='relu')(inp_layer)
    x = Dropout(0.3)(x)
    x = Dense(32, activation='relu')(x)
    x = Dropout(0.3)(x)

    #Output layer
    out_layer = Dense(1, activation='sigmoid')(x)
    model = Model(inputs=inp_layer, outputs=out_layer)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

#Encode categorical variables
with open('dep_encoders.pkl', 'rb') as f:
    label_enc = pickle.load(f)
data['transactionType'] = label_enc.transform(data['transactionType'])

#Handle missing values
X.replace(-1, np.nan, inplace=True)
X.fillna(X.mean(), inplace=True)

#Scale features
with open('dep_scaler.pkl', 'rb') as f:
    scaler = pickle.load(f)
X_scaled = scaler.transform(X)

#Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)

#Split dataset
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

Figure 8.13: ANN for Deposit transactions

```python
with open('dep_xgboost_model.pkl', 'rb') as f:
    xgb_model = pickle.load(f)

#Add XGBoost as feature for hybrid model
xgb_train_preds = xgb_model.predict_proba(X_train)[:, 1].reshape(-1, 1)
xgb_val_preds = xgb_model.predict_proba(X_test)[:, 1].reshape(-1, 1)

#XGBoost + Input Features
X_train_with_xgb = np.hstack((xgb_train_preds, X_train))
X_test_with_xgb = np.hstack((xgb_val_preds, X_test))

#Model Training
input_shape = (X_train_with_xgb.shape[1],)
ann = build_ann(input_shape)
history = ann.fit(X_train_with_xgb, y_train, epochs=50, batch_size=32, validation_data=(X_test_with_xgb, y_test))

#Evaluate model
val_loss, val_accuracy = ann.evaluate(X_test_with_xgb, y_test)
print(f"Validation Loss: {val_loss}, Validation Accuracy: {val_accuracy}")

#Save model
ann.save('dep_ann_model.keras')
```

Figure 8.14: ANN for Deposit transactions

Figure 8.15 illustrates the hybrid deployment architecture of the AI framework, where machine learning (ML) models and artificial neural networks (ANNs) are seamlessly integrated to form a unified predictive system. The input data undergoes preprocessing and transformation before being routed to its designated processing pipeline. Within this pipeline, data is systematically processed through the ML and ANN layers, ensuring high-precision predictions. Upon completing the inference phase, the predicted outcomes are stored in the appropriate databases, facilitating efficient data management, retrieval, and further analytical processing.

This hybrid deployment model is implemented across all four transactional model sets (ML + ANN) to ensure accurate and context-aware predictions. Each model set is optimized for its respective transaction type, enhancing adaptability and fraud detection accuracy.

```python
#Load the encoders and scaler
with open('encoders.pkl', 'rb') as en_file:
    encoders = pickle.load(en_file)
with open('scaler.pkl', 'rb') as sc_file:
    scaler = pickle.load(sc_file)

xgb_model = pickle.load(open('fraud_detection_model.pkl', 'rb'))
print("XGB Model loaded successfully.\n")

ann_model = load_model('ann_fraud_detection.keras')
print("ANN model loaded successfully.")

def preproc_trans(transaction):
    transaction_df = pd.DataFrame([transaction])

    #Encode categ_features
    for col, encoder in encoders.items():
        if col in transaction_df.columns:
            transaction_df[col] = transaction_df[col].apply(
                lambda x: np.where(encoder.classes_ == x)[0][0] if x in encoder.classes_ else -1
            )

    #Scale features
    transaction_scaled = scaler.transform(transaction_df[fts])
    return transaction_scaled

def predict_fraud(transaction_json):
    preprocessed_data = preproc_trans(transaction_json)
    xgb_prediction = xgb_model.predict_proba(preprocessed_data)[:, 1]
    ann_input = np.column_stack((xgb_prediction, preprocessed_data))
    fraud_probability = float(ann_model.predict(ann_input)[0][0])
    is_fraud = fraud_probability > 0.5

    return {
        "fraud_probability": fraud_probability,
        "is_fraud": is_fraud
    }
```

Figure 8.15: Hybrid Deployment of ML+ANN

8.2.3 Client Administrator module

The Client Administrator Module provides financial institutions, banks, and cryptocurrency exchanges with a centralized interface to manage fraud alerts, transaction reports, and system insights. It ensures secure access control, enabling client-specific fraud monitoring and response mechanisms. Figure 8.16 illustrates the React module used for implementing the client dashboard.

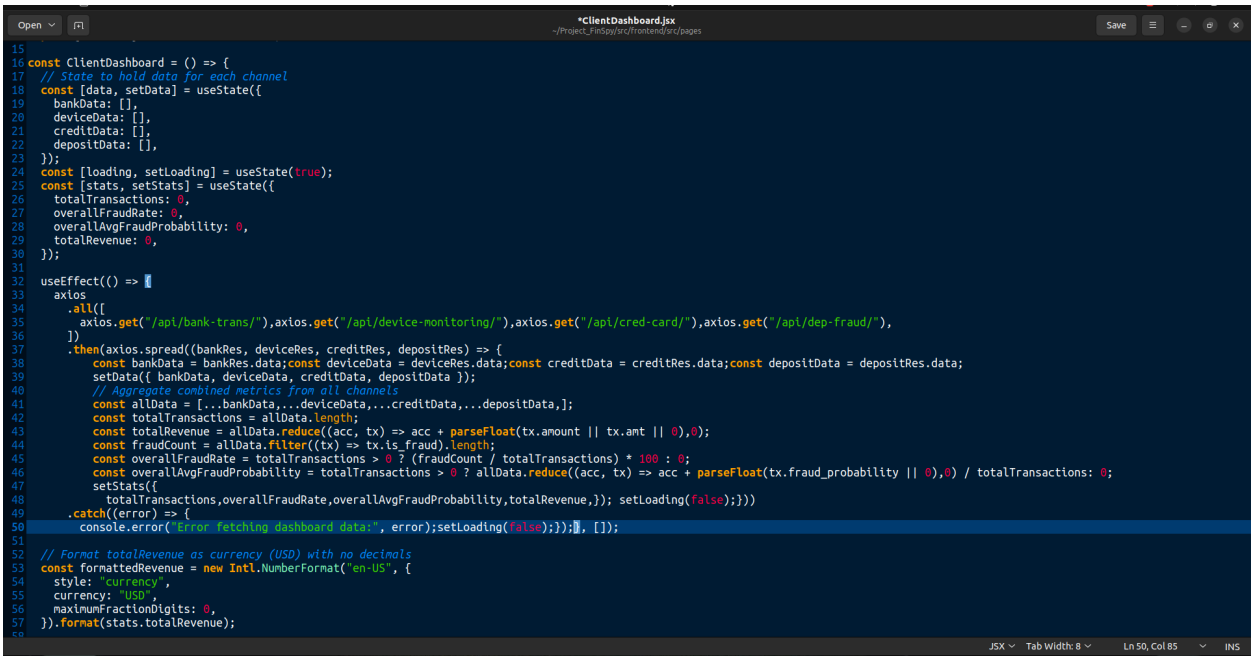

Figure 8.16: React code for client dashboard

8.2.3.1 Fraud Alert Dashboard

Displays real-time flagged transactions categorized by risk level (low, medium, high)and provides detailed event logs with transaction IDs, user activity, and fraud probability scores. It also allows administrators to review flagged transactions and approve or escalate cases.

8.2.3.2 Threat Intelligence Integration

Leverages external threat intelligence feeds to enrich fraud detection insights. Allows administrators to cross-reference suspicious activities with known fraud patterns and Indicators of Compromise (IoCs). Provides visual risk heatmaps for real-time monitoring.

8.2.3.3 Real time AI driven Insights

Harnessing the advanced capabilities of Mistral AI, the system generates AI-driven insights derived from transactional data, offering clients a deeper understanding of emerging patterns and anomalies. These insights equip clients with actionable intelligence, enabling them to proactively respond to potential threats, optimize decision-making, and enhance the security and integrity of their systems. By leveraging real-time analysis and adaptive learning, the framework ensures continuous monitoring and risk mitigation,

empowering businesses to stay ahead of evolving fraud tactics. Figure 8.17 illustrates the implementation in React for AI insights.

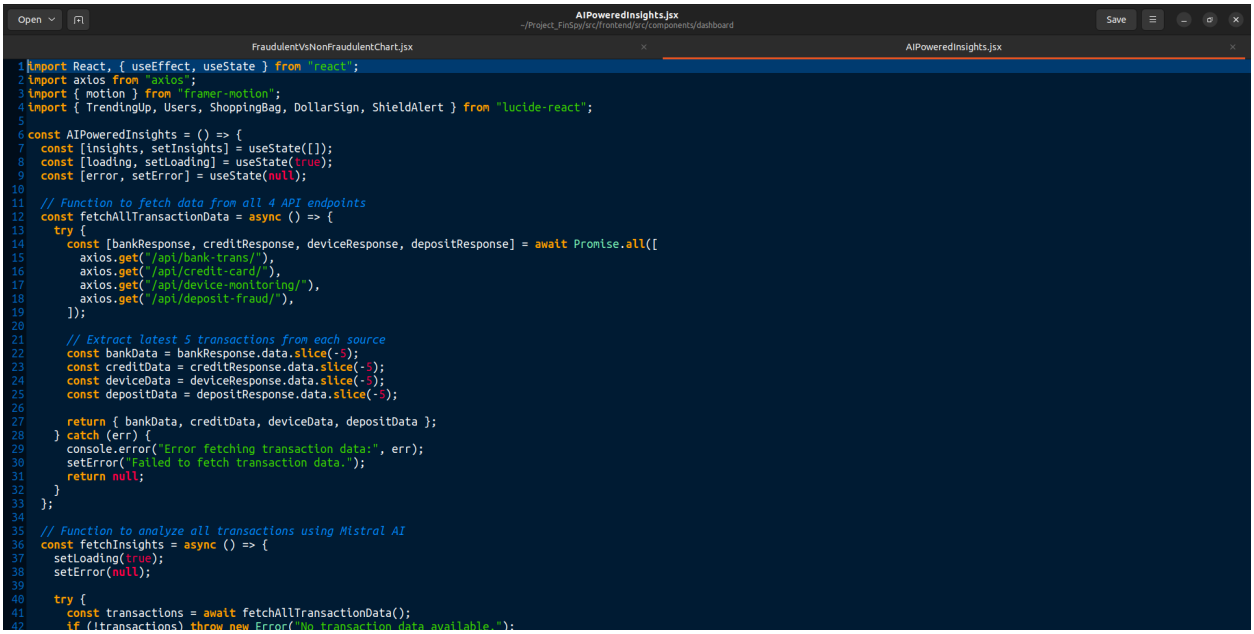

Figure 8.17: AI driven insights

8.2.3.4 User Access & Permissions

Each financial institution can define custom user roles (e.g., Fraud Analyst, Compliance Officer, IT Security Team). Uses RBAC (Role-Based Access Control) to prevent unauthorized access. Supports multi-factor authentication (MFA) for enhanced security.

8.2.3.5 Compliance & Audit Reporting:

Automatically generates compliance reports aligned with PCI-DSS, GDPR, and SOC2 regulations. Logs all administrative actions for audit trail maintenance. Enables secure export of reports for regulatory bodies.

8.2.3.6 Transaction Whitelisting & Blacklisting

Administrators can mark transactions as false positives to refine the fraud detection algorithm. Supports manual whitelisting of verified transactions and blacklisting of fraudulent entities.

8.2.3.7 User-Friendly UI

Interactive web dashboard built with React.js for intuitive navigation. Supports custom queries and data filtering for deep analysis.

8.2.4 Framework Administrator module

The Framework Administrator Module is responsible for managing system-wide configurations, fraud detection thresholds, and machine learning model lifecycle. This module is primarily used by SIEM system operators and financial security teams to oversee the entire fraud detection framework. Figure 8.18 illustrates the React module for the framework administrator dashboard.

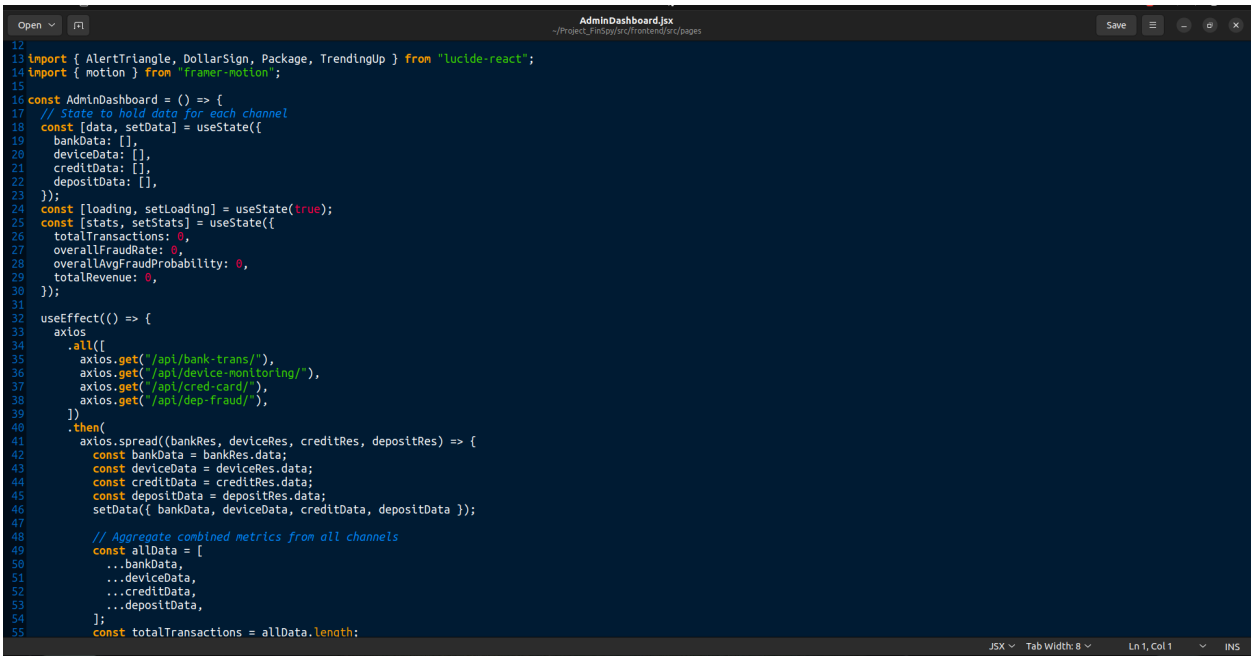

Figure 8.18: React module for Admin dashboard

8.2.4.1 System Monitoring & Health Checks

Provides real-time insights into system health, including API uptime, database performance, and transaction processing speed. Uses Prometheus + Grafana dashboards for live monitoring. Implements alert mechanisms for infrastructure issues or high false-positive rates.

8.2.4.2 Machine Learning Model Management

Allows administrators to configure fraud detection thresholds dynamically. Supports on-demand model retraining when new fraud patterns emerge. Pro-

vides version control for ML models (TensorFlow Model Server integration).

8.2.4.3 Security Access Control:

Manages encryption policies, API keys, and IAM roles for system security. Supports SIEM log integration to detect suspicious administrative activities. Enforces Zero Trust security policies.

8.2.4.4 Data Ingestion & Processing Configurations

Configures Apache Kafka pipelines for real-time transaction ingestion. Optimizes batch processing intervals in Apache Spark. Enables tuning of feature extraction techniques (e.g., PCA-based dimensionality reduction).

Figure 8.19 illustrates the react module responsible for ingesting client data and feeding to the necessary AI pipelines.

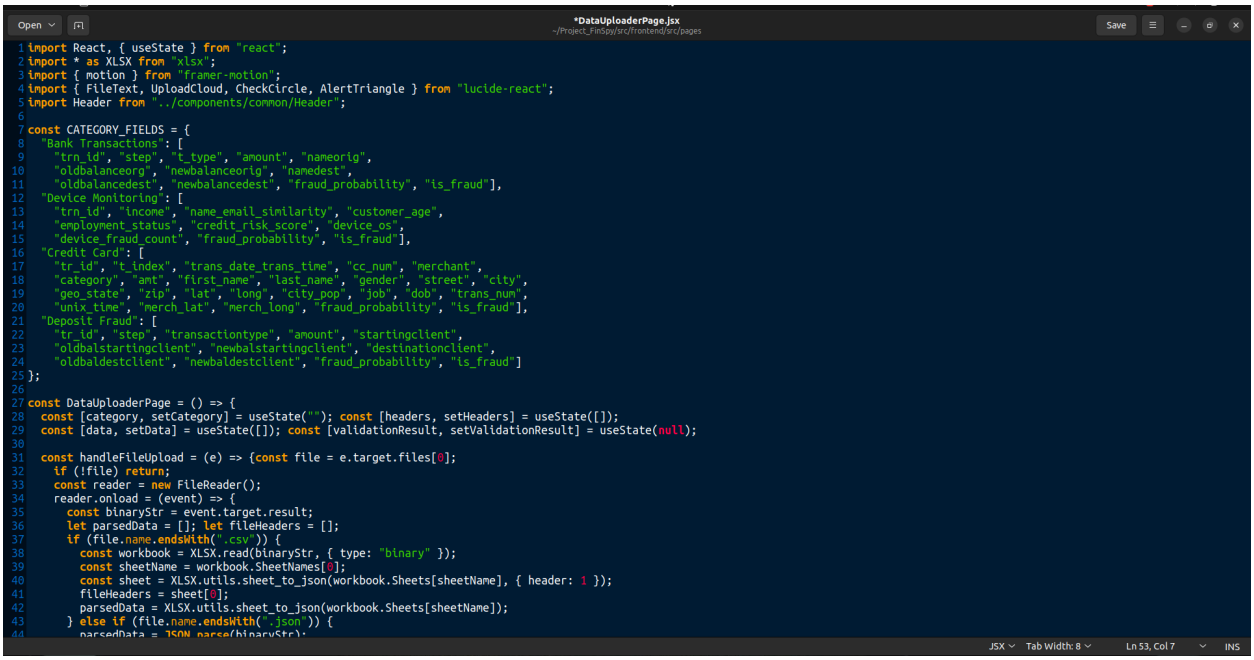

Figure 8.19: Caption

8.2.4.5 Admin Dashboard UI:

The dashboard is built with React.js + Flask APIs to provide a comprehensive fraud detection performance summary.It includes interactive settings panels for configuring fraud rules, system alerts, and compliance reports.

8.2.5 Model updating logic

Utilizing a PostgreSQL-based local database, the system accumulates transaction records until a predefined threshold is reached. Once triggered, the stored data is seamlessly ingested into the hybrid model for retraining, ensuring that the model remains current with emerging trends and features.

An optimized database management script automates both the update process and the periodic refresh of the database. Furthermore, the newly acquired dataset undergoes the same comprehensive transformations and preprocessing as outlined in the preceding module, thereby ensuring consistency and reliability across the entire framework.

8.2.6 Data Visualization module

This project leverages transactional data stored in a PostgreSQL database to generate dynamic visualizations, enabling client administrators to gain a comprehensive understanding of their financial networks. The robust database infrastructure ensures that all transaction data is securely maintained and readily available for analysis.
Through an intuitive dashboard, administrators can effortlessly monitor trends, detect anomalies, and evaluate key performance indicators in real time. By converting raw data into clear, actionable insights, the visualization component empowers decision-makers to optimize financial operations and effectively manage risks across their network.
Figure 8.20 illustrates the data visualisation module using chart.js for dynamic data visualization.

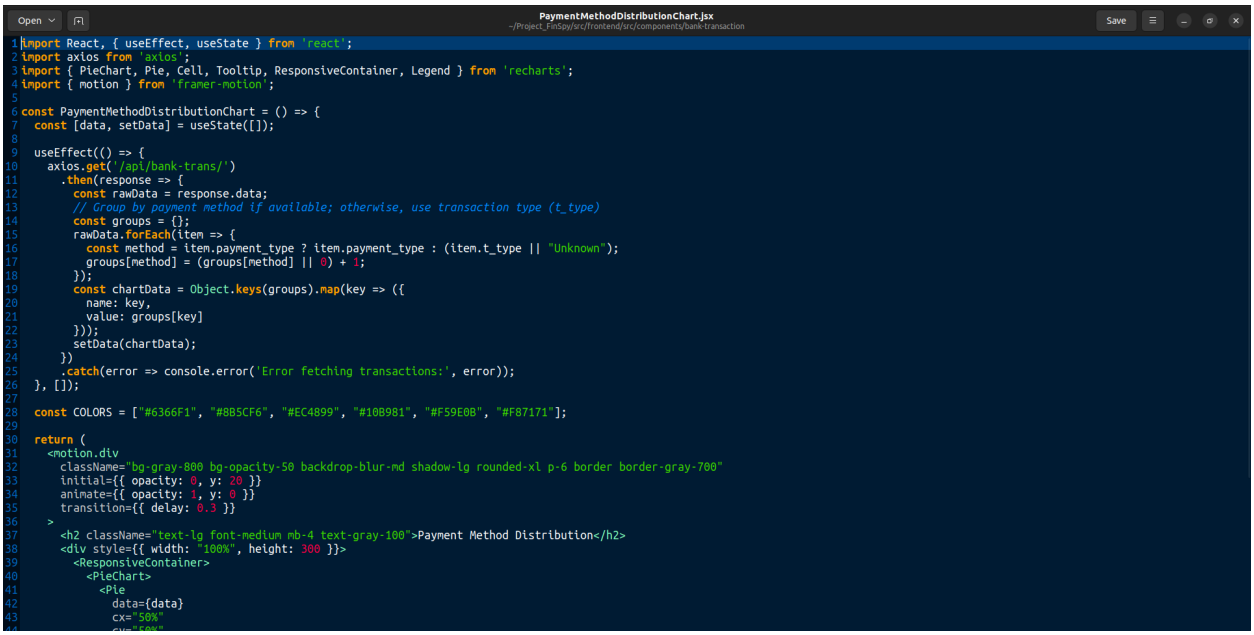

Figure 8.20: React module for Data visualization

8.2.7 Authentication

Our project leverages Auth0 to manage user authentication, ensuring secure access to personalized financial dashboards and sensitive transaction data. Users can log in using social accounts or email-based credentials, with optional multi-factor authentication available for enhanced security. This robust authentication framework safeguards both client and administrative access, maintaining the integrity of the platform while providing a seamless and secure user experience.

Figures 8.20 and 8.21 illustrate the React modules that integrate Auth0 for user authentication, enabling the creation of new user profiles and role-based access to the dashboard.

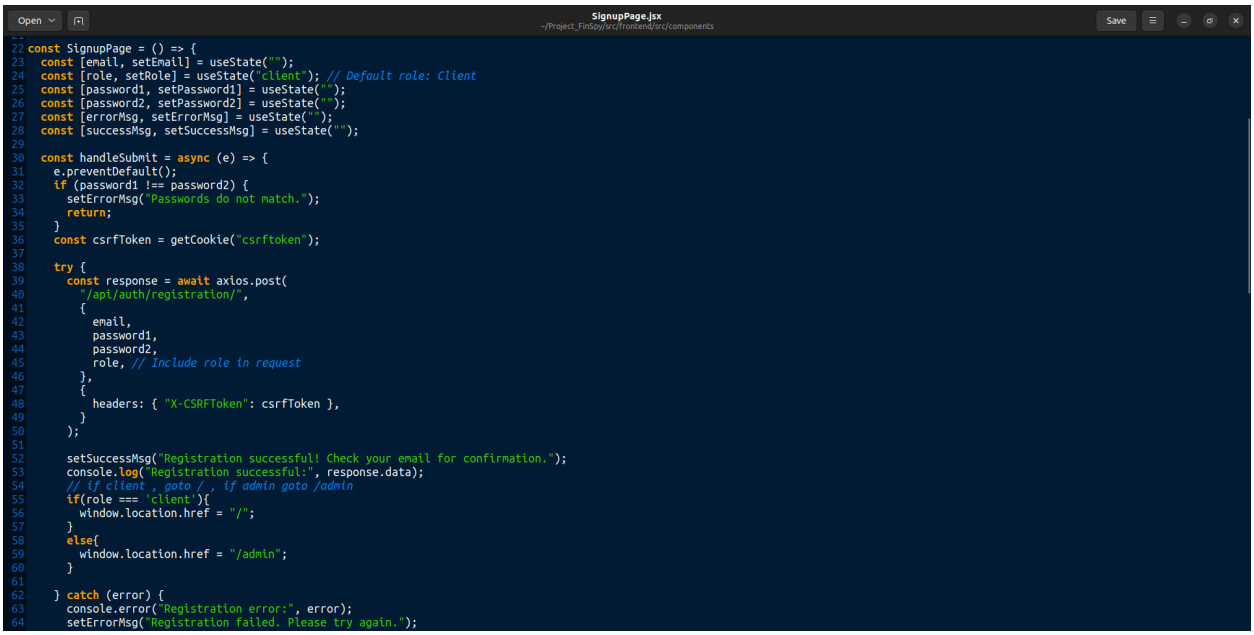

Figure 8.21: React module for sign-up page

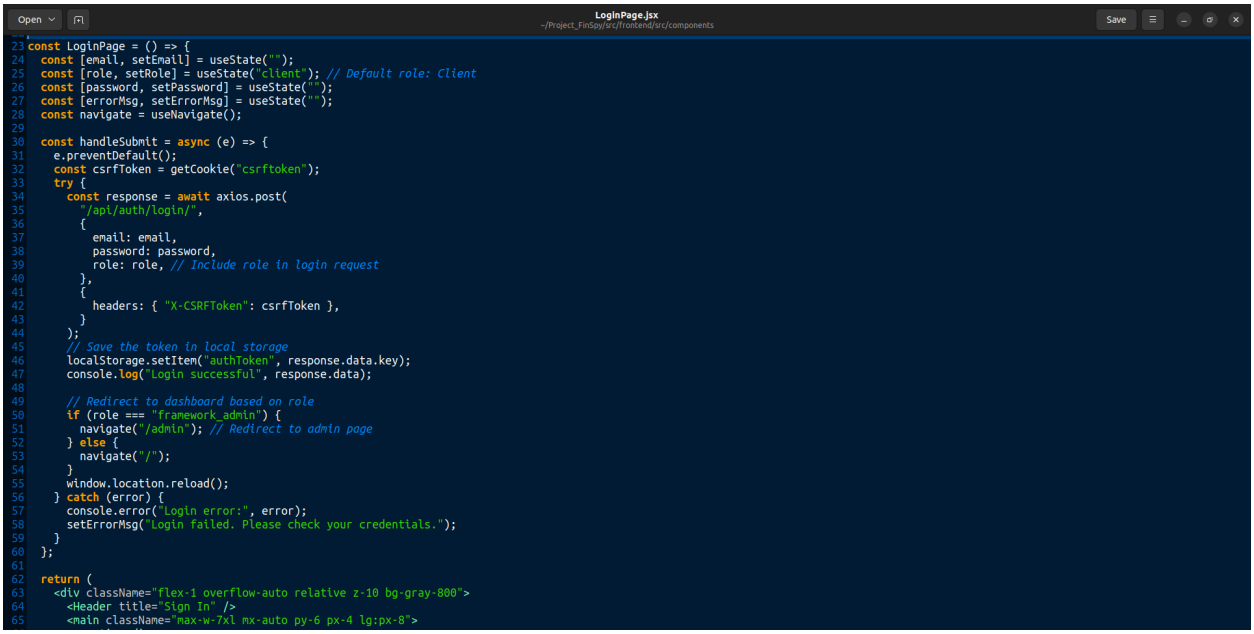

Figure 8.22: React module for Login page

8.3 Technology Stack

8.3.1 Frontend

The frontend is built using React.js for dynamic UI, with Redux Toolkit for state management and Tailwind CSS for styling. Recharts/D3.js enables in-

teractive data visualization, while Auth0/JWT ensures secure authentication and access control.

8.3.2 Backend

The backend is developed in Python Flask and powered by Apache Spark for data processing. SMOTE handles class imbalances, and the QuantGant NN model is used for data synthesis. Authentication is managed via Google Auth0.

8.3.3 AI Hybrid Model

The fraud detection system integrates a hybrid AI model using Keras for a feedforward Artificial Neural Network (ANN) and scikit-learn for XGBoost. Model updating logic is implemented in Python for continuous learning and accuracy improvement.

8.4 Deployment Strategy

The deployment strategy ensures seamless integration, scalability, and high availability of the SIEM-driven financial fraud detection system. It supports both cloud-based and on-premises infrastructures for optimal real-time fraud detection.

The system utilizes AWS EC2 instances for backend services and AWS S3 for data storage. Kubernetes or Docker enables scalable microservices deployment. For on-premises setups, it supports Linux-based servers (Ubuntu/CentOS) on bare-metal or VMs. Data processing relies on Apache Kafka for real-time streaming and Apache Spark for high-speed computation, with PostgreSQL and Elasticsearch for storage and analytics.

The CI/CD pipeline automates deployment, reducing manual effort. Version control is managed via GitHub/GitLab, with GitHub Actions for continuous integration (CI). Docker and Kubernetes handle continuous deployment (CD) for efficient container management.

The system follows a service-oriented model, with a React.js frontend served via Nginx. The Flask/Django backend handles fraud detection, authentication, and database interactions. PostgreSQL ensures reliable transactional storage, while Elasticsearch enables fast querying and fraud analytics. Monitoring uses Prometheus and Grafana, with centralized logging via the

ELK stack (Elasticsearch, Logstash, Kibana).

For real-time fraud detection, Apache Kafka processes incoming transactions, while fraud models deployed via TensorFlow Serving or FastAPI analyze patterns and flag anomalies.

To ensure high availability and scalability, the system uses AWS Elastic Load Balancer (ELB) or NGINX Reverse Proxy for load balancing and auto-scaling. It is deployed across multiple AWS availability zones for redundancy and fault tolerance.

This strategy keeps the fraud detection system resilient, scalable, and efficient, meeting financial security requirements while enabling real-time, AI-driven fraud detection.

8.5 Security Considerations

Network security is a critical aspect of the system. It follows a Zero Trust Architecture (ZTA), where all access requests are strictly verified before being granted. Network segmentation is implemented to isolate the frontend, backend, and database servers, preventing unauthorized access between different system components. Additionally, TLS 1.3 encryption secures API communications, protecting against Man-in-the-Middle (MITM) attacks and ensuring transaction data integrity.

Application security measures are in place to prevent cyber threats. Input validation and sanitization protect the system from SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) attacks. The system enforces OAuth2.0/JWT authentication to secure API access. Additionally, Role-Based Access Control (RBAC) is implemented to restrict system privileges, ensuring that Admins, Analysts, and Users can only access relevant functionalities according to their roles.

Data security is a top priority, with all sensitive transaction data encrypted using AES-256 encryption, preventing unauthorized access. Furthermore, data masking is applied to anonymize personally identifiable information (PII) in logs and dashboards, ensuring compliance with privacy standards while maintaining usability.

Fraud detection and threat intelligence are integral components of the system. The platform integrates external threat intelligence feeds, continuously updating fraud detection capabilities using Indicators of Compromise (IoCs)

and malware signatures. Additionally, unsupervised machine learning models analyze transactions in real time, identifying anomalies and reducing false positives to improve fraud detection accuracy.

Compliance and audit logging mechanisms ensure adherence to financial security regulations. The system is designed to meet GDPR, PCI-DSS, and SOC2 compliance, guaranteeing secure transaction handling. Audit logging records all access, fraud alerts, and system modifications, providing a complete and traceable history for regulatory reporting and security analysis.

This security framework ensures that the system remains robust, resilient, and compliant while maintaining the highest standards of fraud prevention and data protection.

Chapter 9

Implementation Results

9.1 Client Dashboard

Figures 9.1 to 9.8 illustrate the client dashboard, which provides clients with access to the framework via a dedicated website developed by the implementation team.

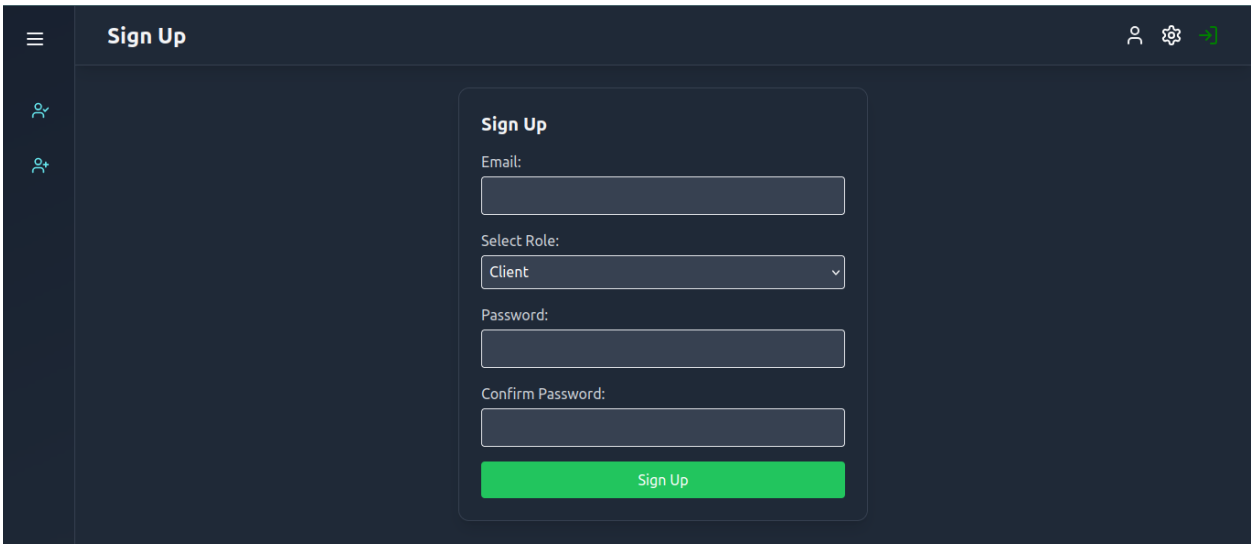

Figure 9.1: Client Signup page

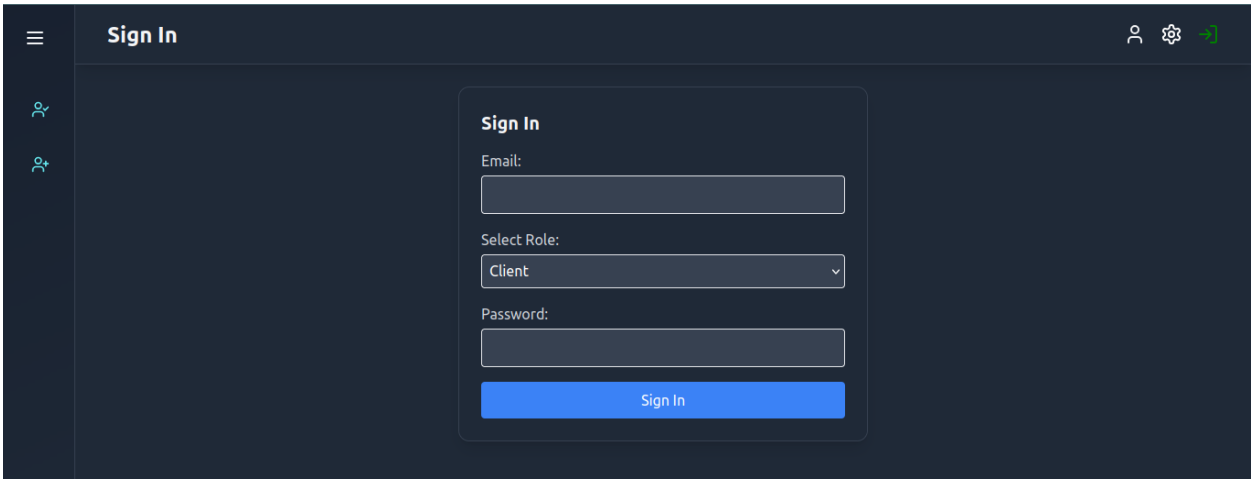

Figure 9.2: Client Login page

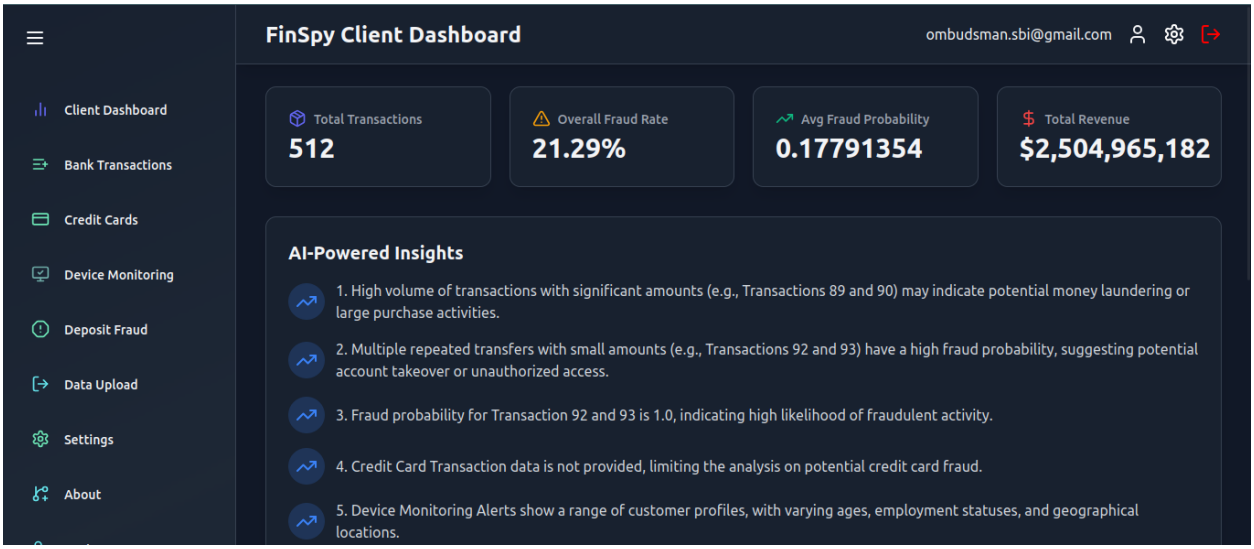

Figure 9.3: Client Dashboard with AI insights

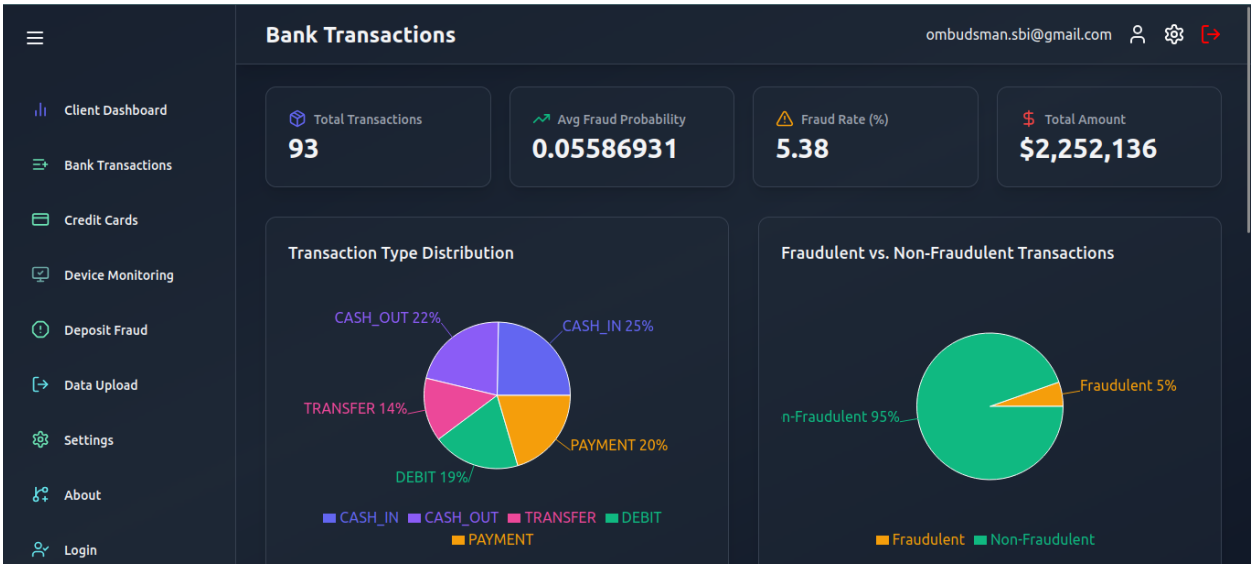

Figure 9.4: Bank Transactions page

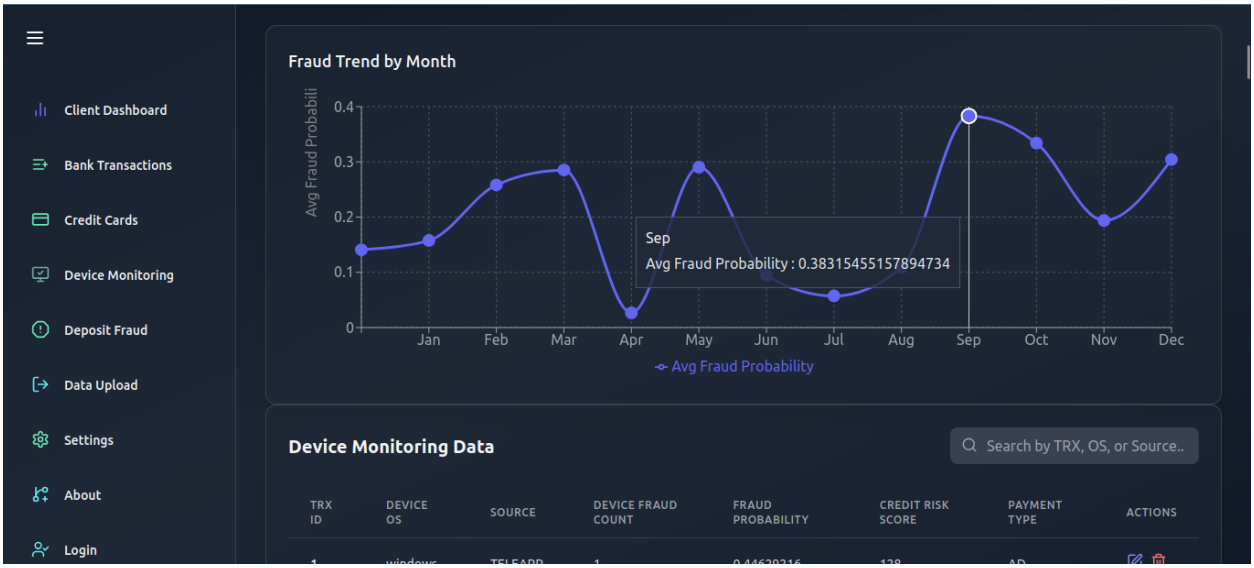

Figure 9.5: Device monitoring Page

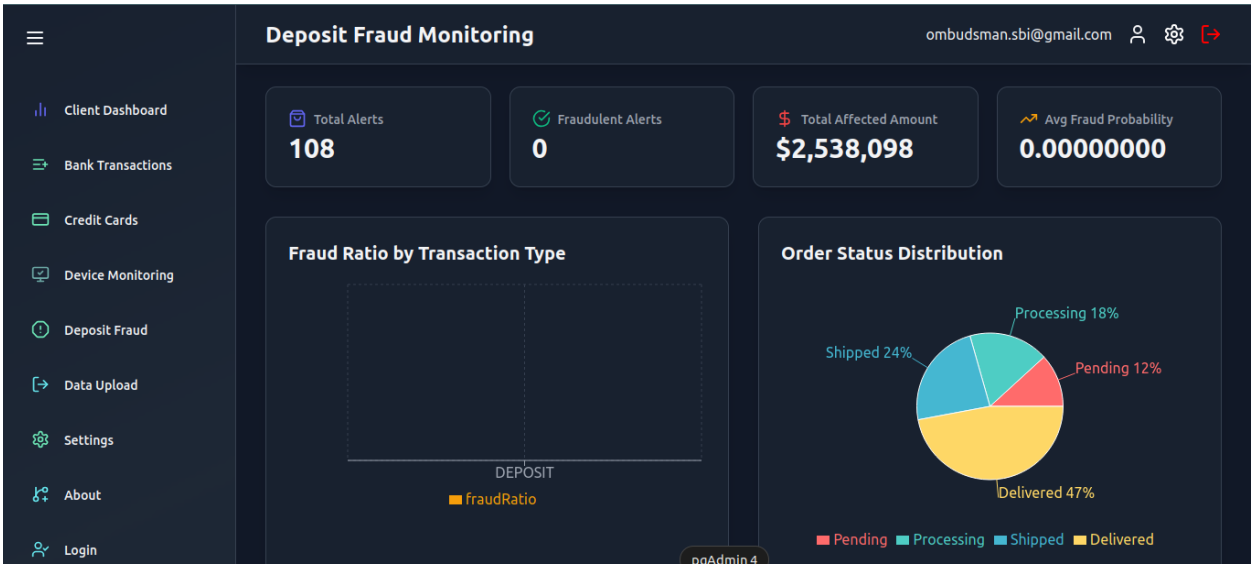

Figure 9.6: Deposit Fraud page

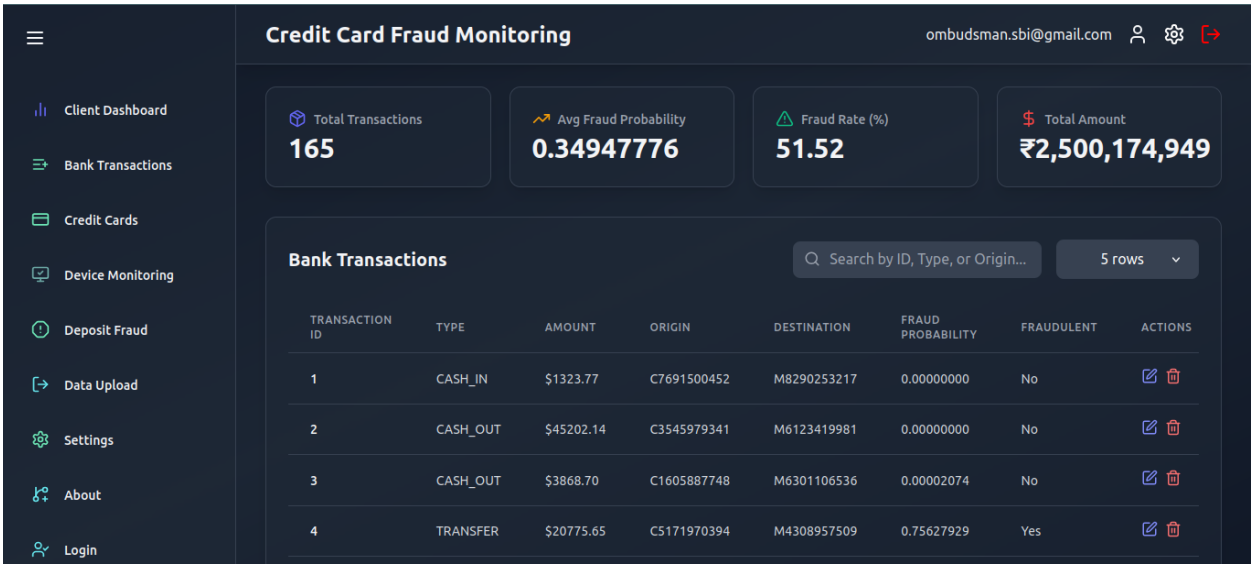

Figure 9.7: Credit Card transactions page

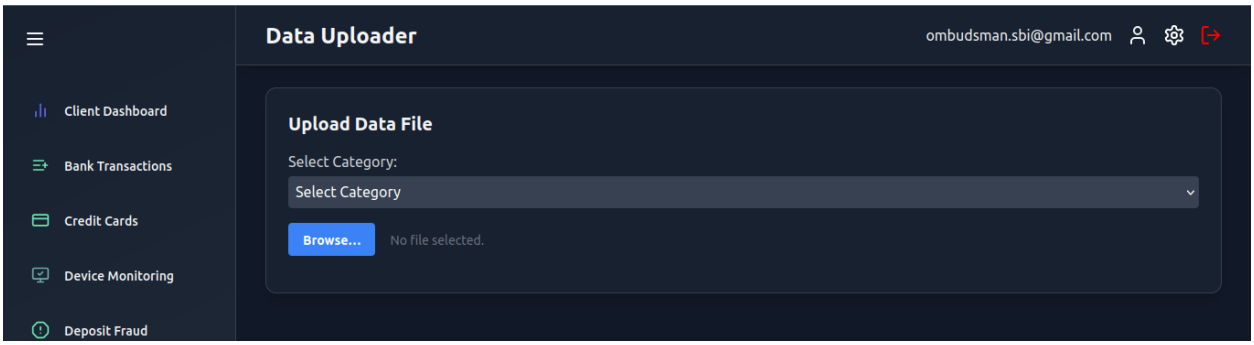

Figure 9.8: Data upload page

9.2 Framework Administrator Dashboard

Figures 9.9 to 9.12 illustrate the administrator dashboard, designed for real-time monitoring of key system parameters, including model health, database status, API activity, active users, and total users. This dashboard provides a comprehensive overview, enabling administrators to efficiently track system performance and quickly identify and resolve any issues as they arise.

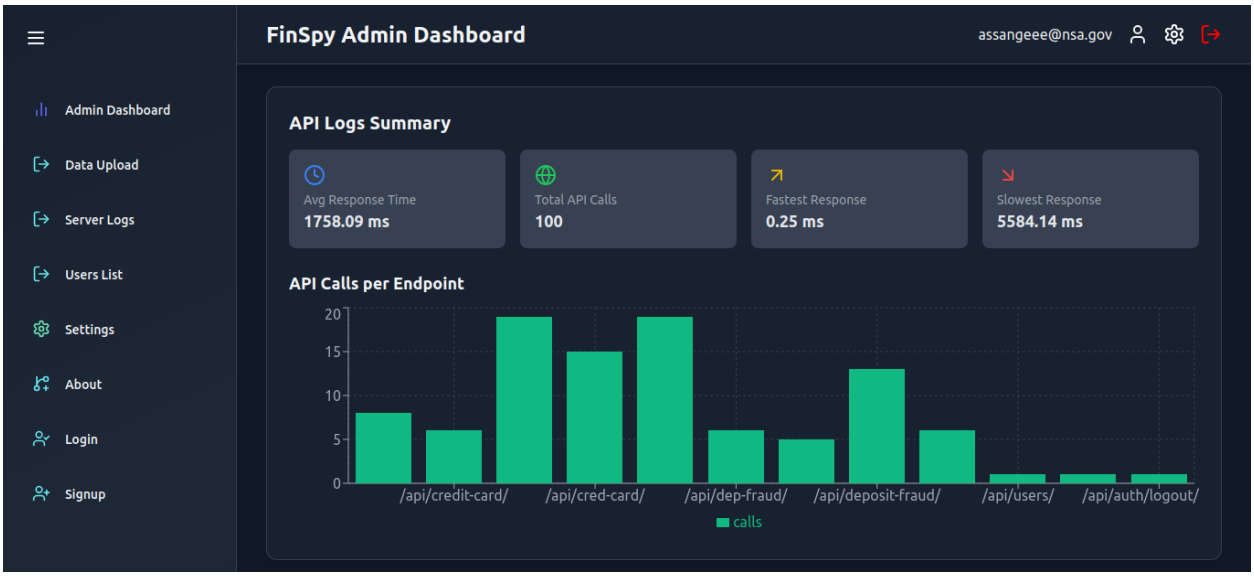

Figure 9.9: Framework Administrator Dashboard

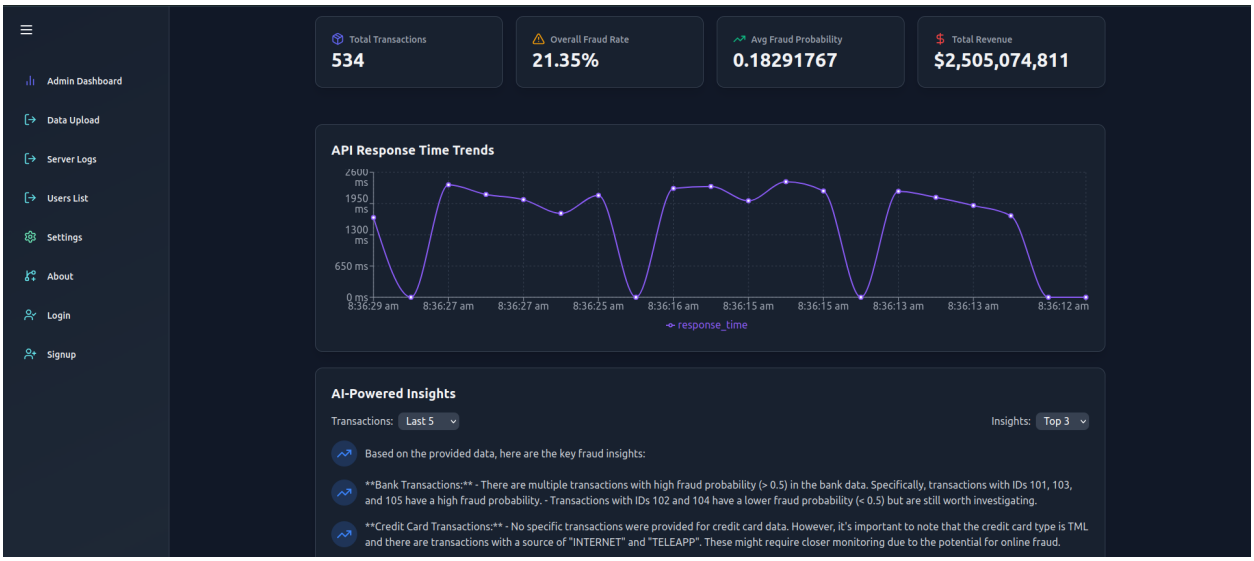

Figure 9.10: Framework Administrator Dashboard

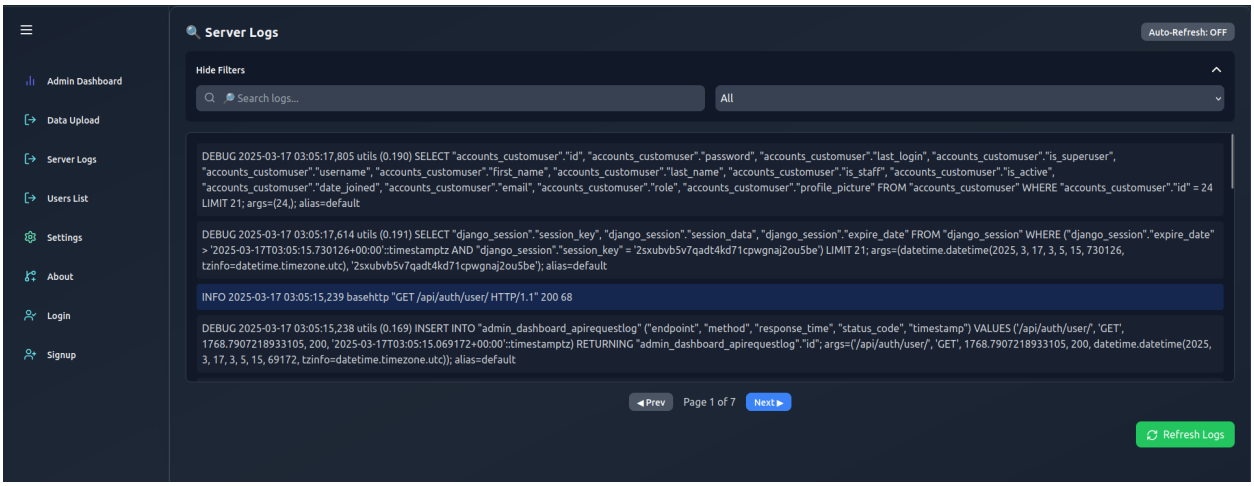

Figure 9.11: Server Logs

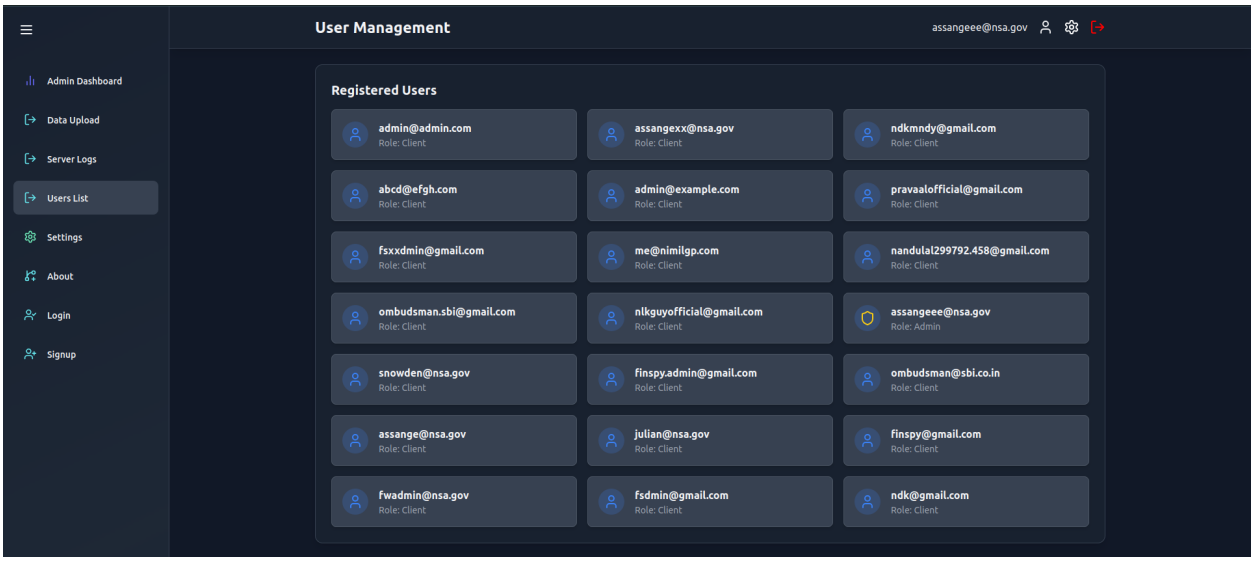

Figure 9.12: Users list

9.3 Client API endpoints

Figures 9.13 to 9.16 illustrate the APIs available for client integration, enabling seamless incorporation of the designed framework into existing infrastructure and platforms. These APIs facilitate efficient communication and interoperability with the client's system.

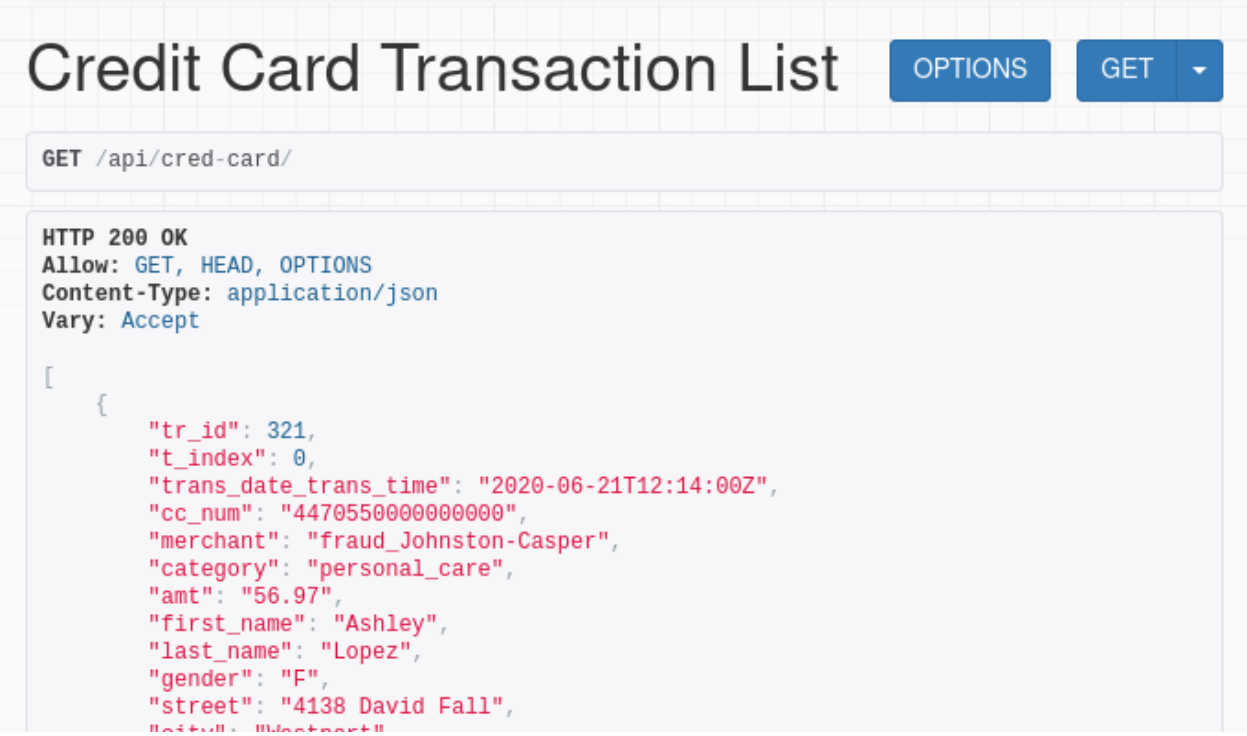

Figure 9.13: Credit Card transactions API

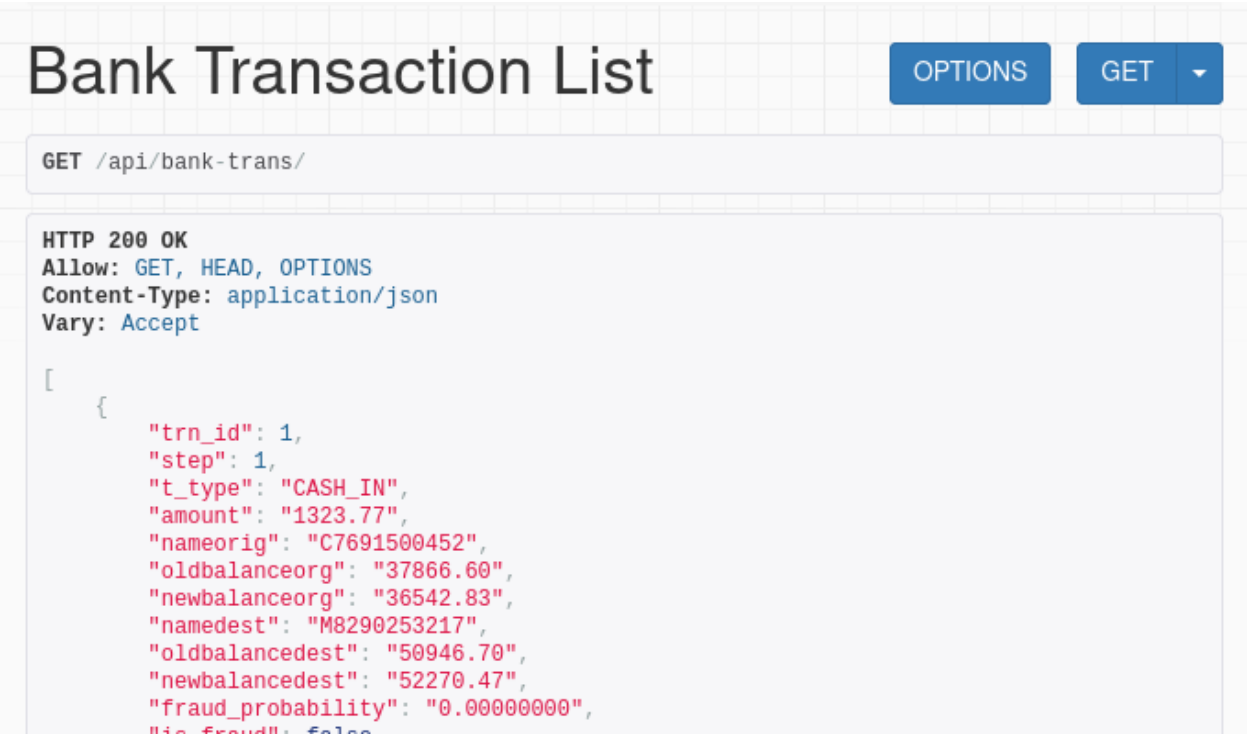

Figure 9.14: Bank transactions API

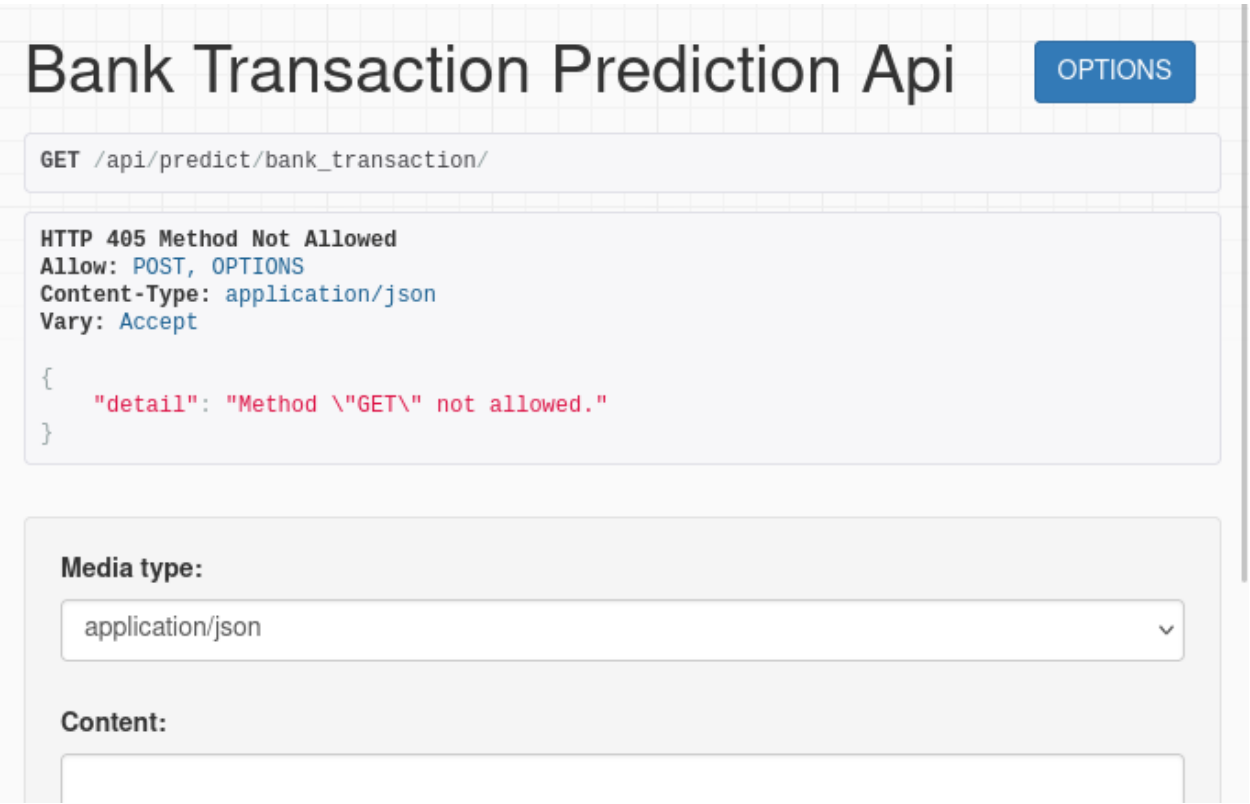

Figure 9.15: Bank transactions API

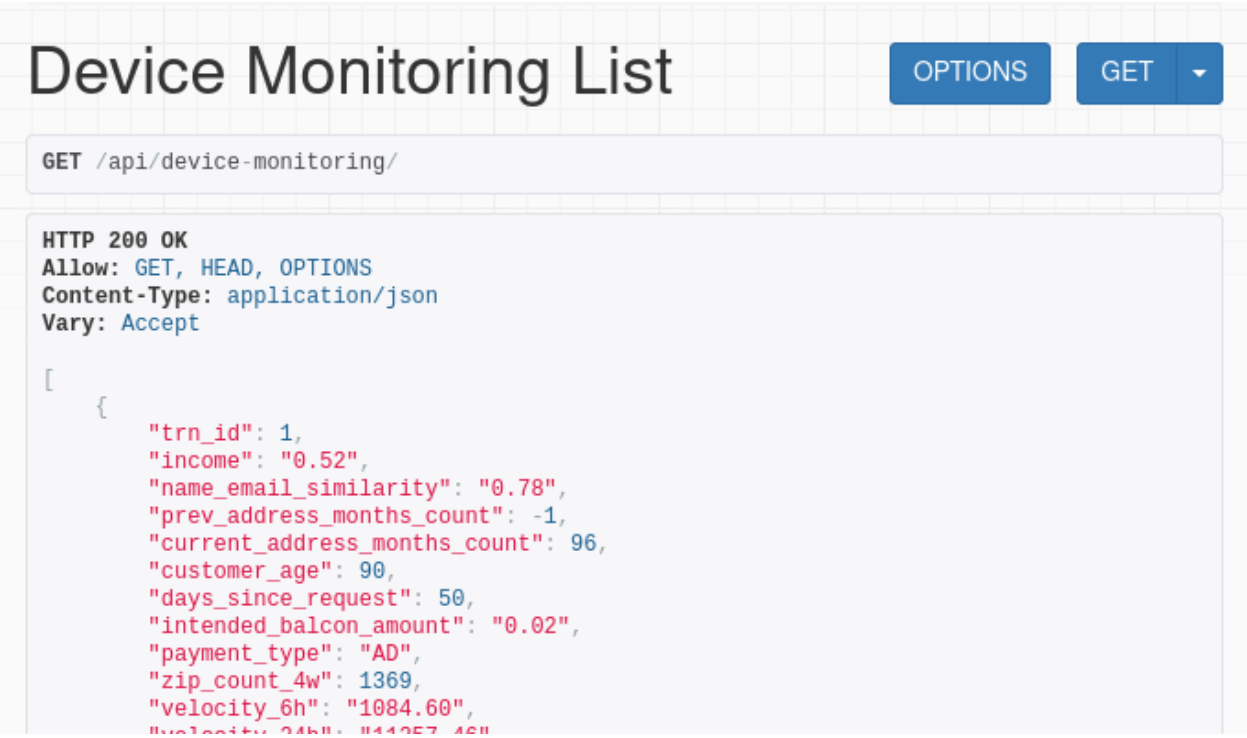

Figure 9.16: Device Monitoring API

9.4 Framework Admin API endpoints

Figures 9.13 to 9.16 illustrate the framework monitoring API, which enables real-time tracking of API calls, database performance, model health, and logs of all data access requests.

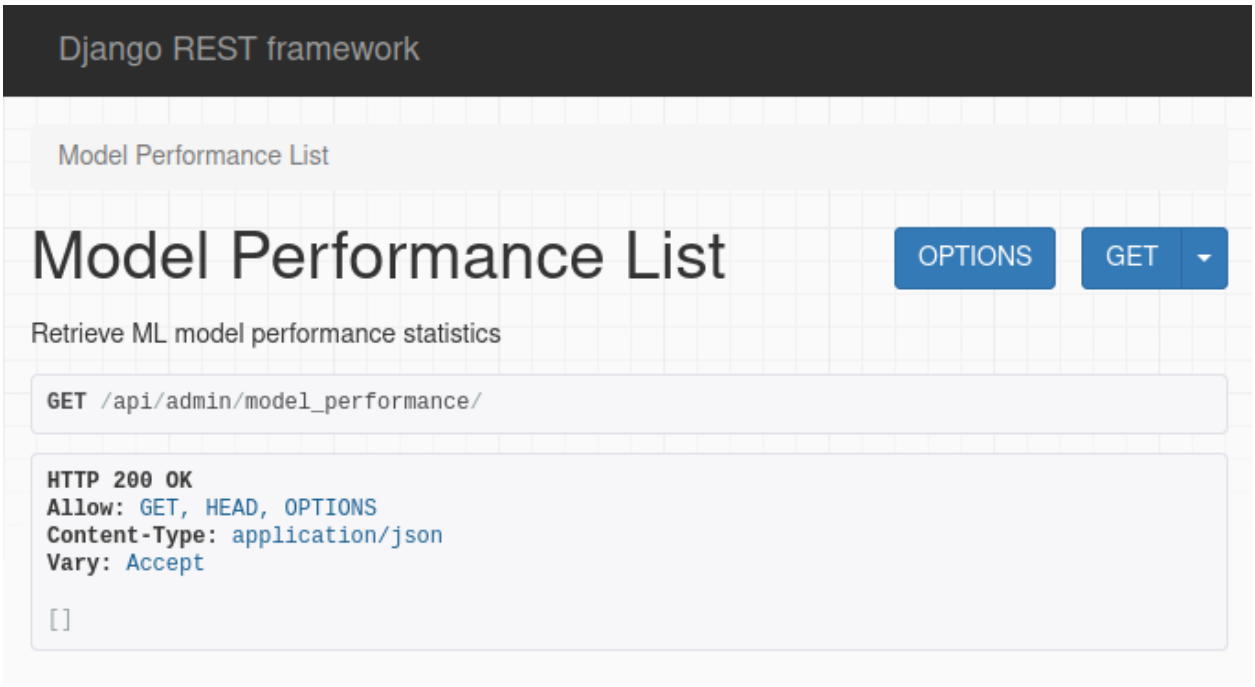

Figure 9.17: Model Health API

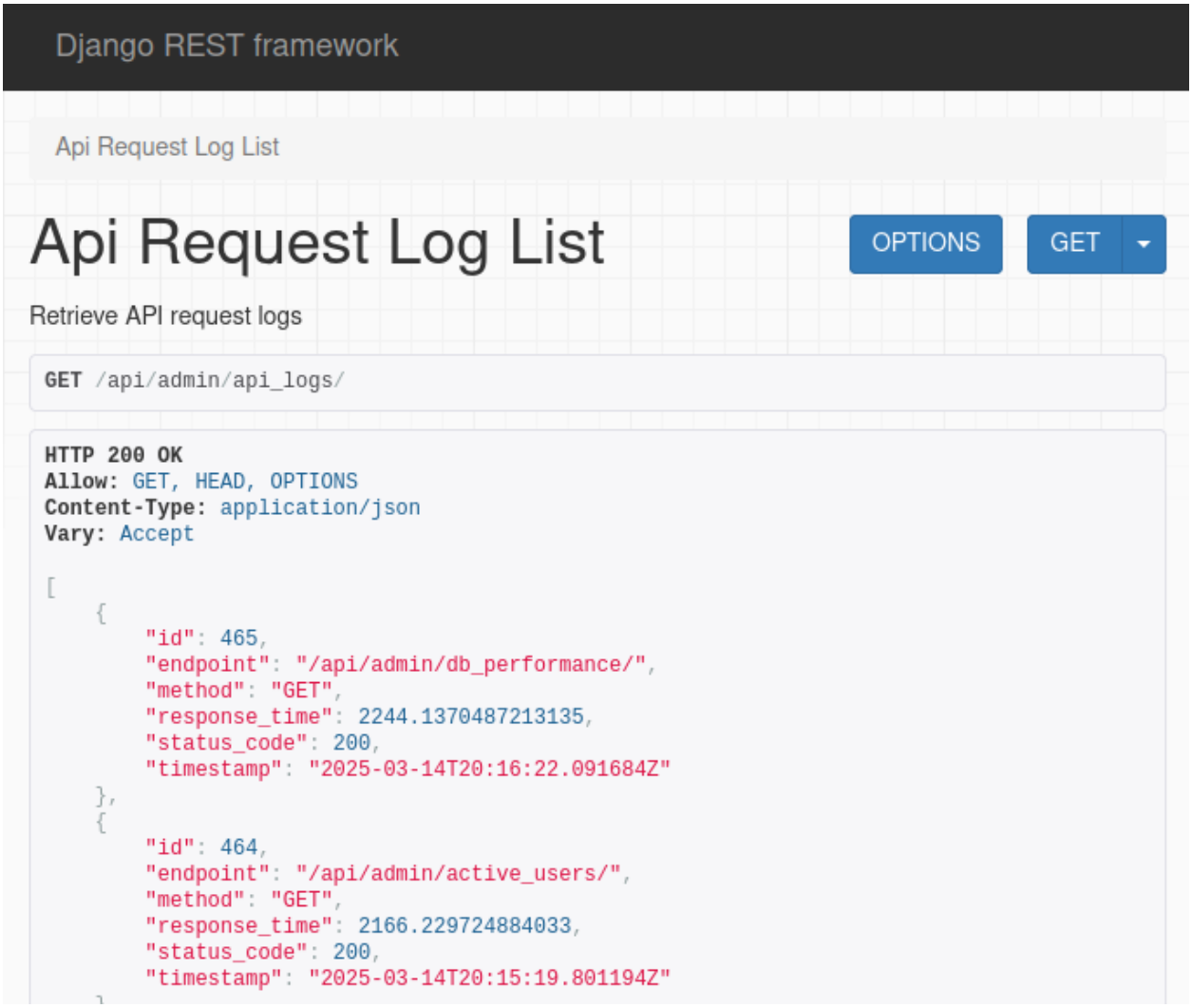

Figure 9.18: API Request log list

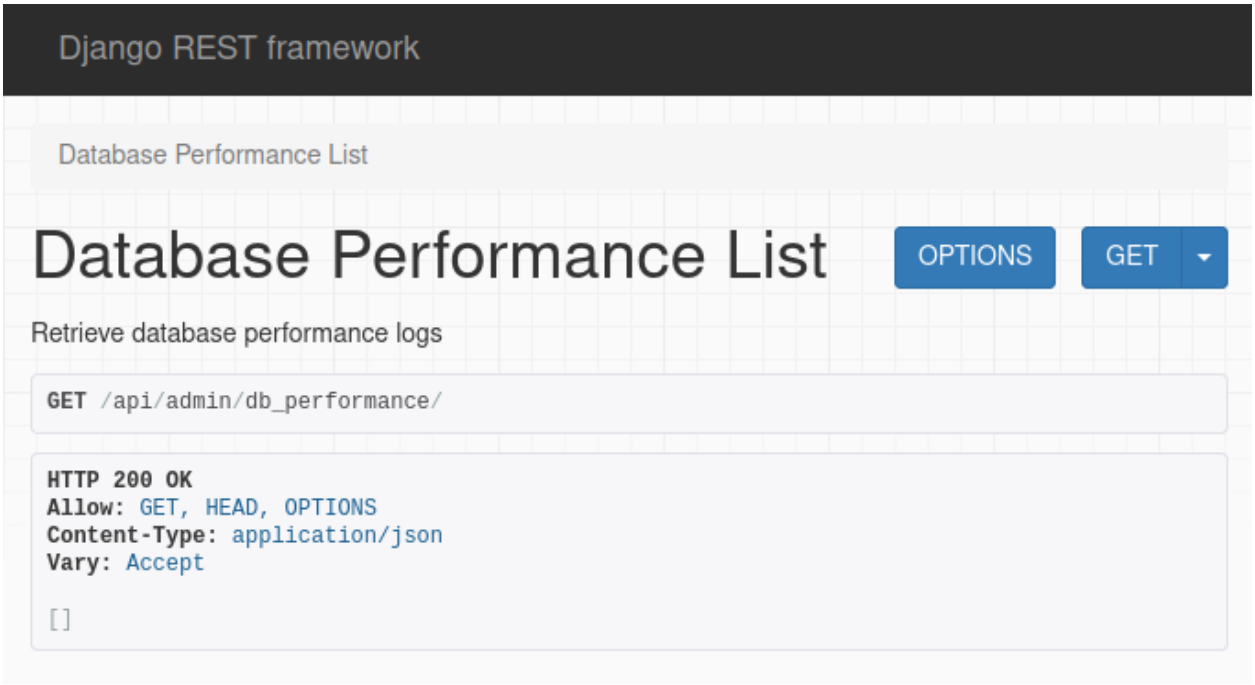

Figure 9.19: Database Performance API

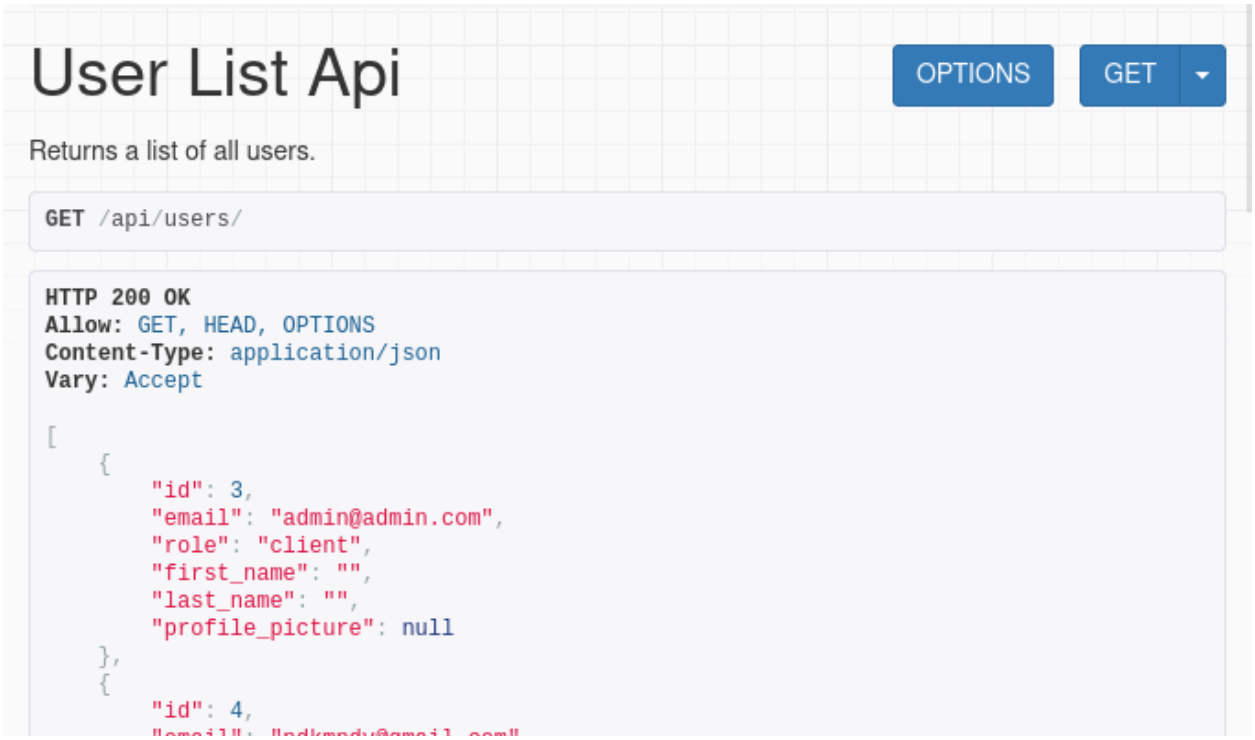

Figure 9.20: User list API

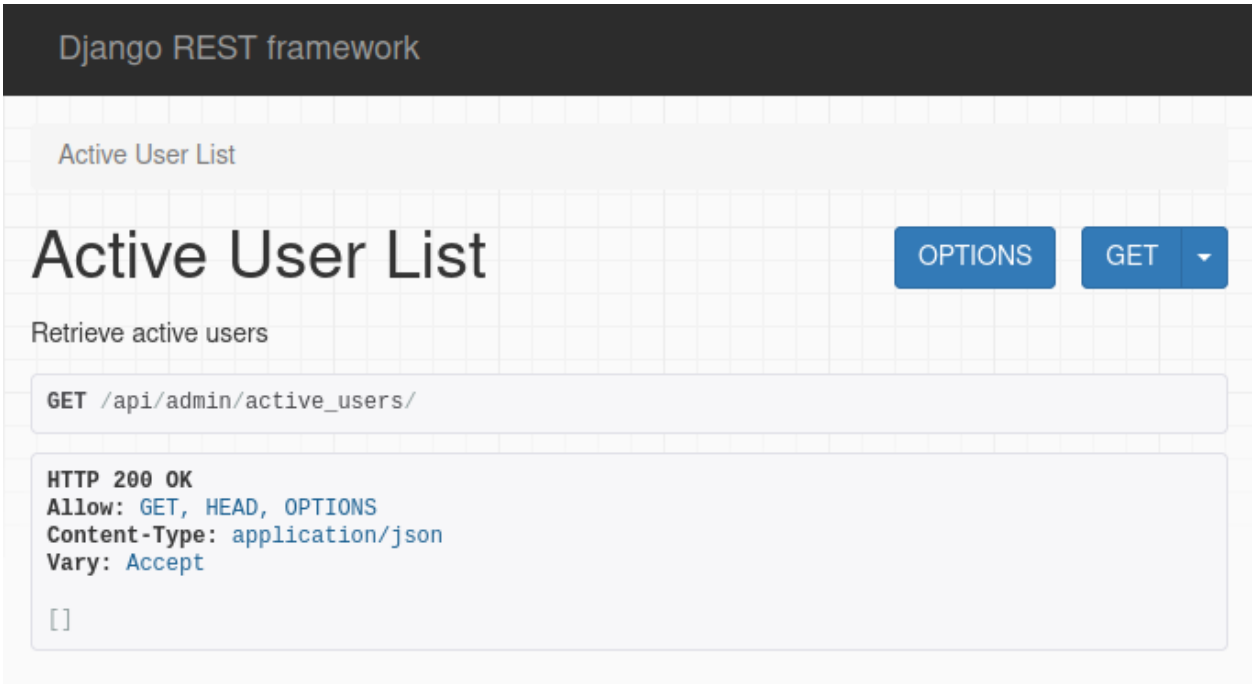

Figure 9.21: Active user list API

Chapter 10

Conclusion

The proposed fraud detection system represents a significant advancement in financial security, addressing the limitations of traditional fraud monitoring solutions while setting a new industry benchmark for real-time detection and compliance management. By leveraging hybrid AI modeling and external threat intelligence, the system enhances detection accuracy, streamlines compliance reporting, and ensures proactive fraud mitigation. Its well-defined functional and non-functional requirements, along with a scalable and robust architecture, enable seamless adaptation to the evolving financial fraud landscape.

With features such as real-time fraud detection, automated alerts, and seamless data integration across networks, the platform meets the immediate needs of financial institutions while remaining future-proof in an increasingly digital economy. The availability of multiple API endpoints and flexible deployment options ensures compatibility across diverse infrastructures without additional overhead.

Security Information and Event Management (SIEM) plays a crucial role in network monitoring, ensuring endpoint security across a system. Similarly, financial institutions operate within a complex transactional network where monetary assets are among the most valuable resources. Implementing a robust and reliable SIEM-inspired fraud detection framework in the financial domain ensures a stable and trustworthy model for fraud prevention.

By harnessing the power of hybrid AI—integrating machine learning with neural networks—the system thoroughly verifies every transaction and user

interaction within a client's financial network. This guarantees enhanced security and reliability, safeguarding institutional assets and customer trust. Ultimately, this framework not only offers state-of-the-art fraud detection capabilities but also provides financial institutions with peace of mind and seamless access to cutting-edge security solutions.

Chapter 11

Future Scope

The future scope of this project involves expanding its capabilities to address increasingly sophisticated fraud patterns across diverse financial ecosystems. By incorporating advanced AI techniques such as federated learning, the system can enhance privacy-preserving fraud detection while enabling collaborative intelligence across financial institutions without compromising sensitive data. Additionally, the integration of newer machine learning models, including self-supervised learning and transformer-based architectures, can further refine anomaly detection and predictive accuracy.

To ensure broader applicability, the framework can evolve to support a wider range of regulatory environments, adapting to compliance standards across multiple jurisdictions. Enhancing real-time threat intelligence capabilities by integrating external data sources, behavioral analytics, and AI-driven risk scoring can further improve fraud prevention. Moreover, expanding interoperability with existing financial security infrastructures—such as SIEM systems and transaction monitoring platforms—will enable a more comprehensive, end-to-end fraud management solution. These advancements will position the system as a next-generation fraud detection framework, capable of safeguarding financial institutions of all sizes in an increasingly complex digital landscape.

Bibliography

[1] Xin Zhao, Qiong Zhang, Chang Zhang, "Enhancing Transaction Fraud Detection with a Hybrid Machine Learning Model," in *2024 IEEE 4th International Conference on Electronic Technology, Communication and Information (ICETCI)*, Changchun, China, 2024, pp. 427-435.

[2] Noyan Tendikov, Leila Rzayeva, Bilal Saoud, Ibraheem Shayea, Marwan Hadri Azmi, Ali Myrzatay, Mohammad Alnakhli, "Security Information Event Management data acquisition and analysis methods with machine learning principles," *Results in Engineering*, vol. 22, pp. 102254, 2024.

[3] Ahmad Amjad Mir, "Adaptive Fraud Detection Systems: Real-Time Learning from Credit Card Transaction Data," *Advances in Computer Sciences*, vol. 7, pp. 1-10, 2024.

[4] H. Cheddy and R. K. Sungkur, "Identifying Fraudulent Credit Card Transactions Using AI," *2024 4th International Conference on Information Communication and Software Engineering (ICICSE)*, Beijing, China, 2024, pp. 75-79

[5] M. Marripudugala, "AI-Powered Fraud Detection in the Financial Services Sector: A Machine Learning Approach," *2024 2nd International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS)*, Erode, India, 2024, pp. 795-799

[6] R. Sharma and D. Minhas, "Comparative Performance of Random Forest versus Gradient Boosting Machines in Detecting Financial Fraud," *2024 7th International Conference on Circuit Power and Computing Technologies (ICCPCT)*, Kollam, India, 2024, pp. 1027-1030

[7] In-seok Jeon, Keun-hee Han, Dong-won Kim, Jin-yung Choi, "Using the SIEM Software Vulnerability Detection Model Proposed."

[8] P. Nagpal, "The Transformative Influence of Artificial Intelligence (AI) on Financial Organizations Worldwide," *2023 IEEE International Conference on ICT in Business Industry Government (ICTBIG)*, Indore, India, 2023, pp. 1-4,

[9] A. N. Ahmed and R. Saini, "A Survey on Detection of Fraudulent Credit Card Transactions Using Machine Learning Algorithms," *2023 3rd International Conference on Intelligent Communication and Computational Techniques (ICCT)*, Jaipur, India, 2023, pp. 1-5,

[10] Assefa, Samuel, "Generating Synthetic Data in Finance: Opportunities, Challenges and Pitfalls", 2020

[11] O. Podzins and A. Romanovs, "Why SIEM is Irreplaceable in a Secure IT Environment?," *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, Vilnius, Lithuania, 2019, pp. 1-5