

**Tribhuvan University**

**Faculty of Humanities and Social Science**

**A PROJECT REPORT ON  
FraudShield System**

**Submitted to**

**Department of Computer Application**

**Nepal Mega College**

**In partial fulfillment of the requirements for the Bachelors in Computer Application**

**Submitted by**

Bijay koirala (26602078)

Kaushal Joshi (26602082)

June, 2025

Under the Supervision of

**Basanta Chapagain**



**Tribhuvan University**

**Faculty of Humanities and Social Sciences**

**Nepal Mega College**

# SUPERVISOR’S RECOMMENDATION

I hereby recommend that this project prepared under my supervision by “**Bijay Koirala**” and “**Kaushal Joshi**”, entitled “**FraudShield**” in partial fulfillment of the requirements for the degree of Bachelor of Computer Applications is recommended for the final evaluation.

**Signature of the Supervisor**

**Basanta Chapagain**

BCA 6th Semester

Nepal Mega College, Babarmahal, Kathmandu



**Tribhuvan University**

**Faculty of Humanities and Social Sciences**

**Nepal Mega College**

# LETTER OF APPROVAL

This is to certify that this project prepared by, “**Bijay Koirala**” and “**Kaushal Joshi**” entitled “**FraudShield**” in partial fulfillment of the requirements for the degree of Bachelor in Computer Application has been evaluated. In our opinion it is satisfactory in the scope and quality as a project for the required degree.

|  |  |
| --- | --- |
| ………………………………  Basanta Chapagain  Supervisor  Babarmahal, Kathmandu | ---------------------------------  Dharma Raj Poudel  Coordinator  Babarmahal, Kathmandu |
| -------------------------------  Internal Examiner | ------------------------------  External Examiner |

# ABSTRACT

FraudShield is a fraud detection system developed for small online stores to identify suspicious transactions. Instead of using complex machine learning, the system use straightforward rule-based approach that checks for common red flags. The system analyzes transaction details including email addresses, IP addresses, and card information, looking for warning signs such as disposable email accounts (like 10minutemail), suspicious IP addresses, or the same card being used with multiple different emails. When a purchase is made, the system quickly evaluates these patterns and assigns a fraud score. FraudShield's key feature is transparency - when flagging a transaction, it provides exact reasons, such as temporary email service usage or cards seen in multiple locations. Store owners can view these explanations and make informed decisions about order acceptance or review. Built using Python for the backend, MongoDB for the database, and JavaScript for the web interface, the system runs locally or on a simple server. While unable to detect sophisticated fraud using advanced techniques, FraudShield effectively prevents obvious fraud attempts commonly faced by small businesses. Though not 100% accurate, but it serves as a practical first line of defense that is easy to understand and implement.

***Keywords: FraudShield, Fraud detection, Rule-based system, E-commerce security***

# ACKNOWLEDGEMENT

We would like to express our sincere gratitude to everyone who supported and contributed

to the development of the FraudShield project. First and foremost, we would like to thank our teachers and mentors for their invaluable guidance and encouragement throughout this project. We are also deeply grateful to our supervisor for their continuous support and direction. Your expertise and feedback helped steer this project towards success. Lastly, we would like to acknowledge the online communities and resources that provided the technical knowledge and inspiration necessary for this project. Its contributions to the broader field of educational technology are deeply appreciated. Thank you all for your support and encouragement. This project would not have been possible without you.

Bijay Koirala (26602078)

Kaushal Joshi (26602082)

# TABLE OF CONTENTS

[SUPERVISOR’S RECOMMENDATION i](#_Toc204969045)

[LETTER OF APPROVAL ii](#_Toc204969046)

[ABSTRACT iii](#_Toc204969047)

[ACKNOWLEDGEMENT iv](#_Toc204969048)

[TABLE OF CONTENTS v](#_Toc204969049)

[LIST OF ABBREVIATIONS ix](#_Toc204969050)

[CHAPTER 1: 1](#_Toc204969051)

[INTRODUCTION 1](#_Toc204969052)

[1.1 Introduction 1](#_Toc204969053)

[1.2. Problem Statement: 1](#_Toc204969054)

[1.3. Objectives: 2](#_Toc204969055)

[1.4. Scope and Limitation 2](#_Toc204969056)

[1.4.1. Scope 2](#_Toc204969057)

[1.4.2 Limitations 2](#_Toc204969058)

[1.5. Development Methodology 3](#_Toc204969059)

[1.6. Report Organization 3](#_Toc204969060)

[CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW 5](#_Toc204969061)

[2.1. Background Study 5](#_Toc204969062)

[2.2. Literature Review 5](#_Toc204969063)

[2.2.1 Review of similar system 6](#_Toc204969064)

[CHAPTER 3: SYSTEM ANALYSIS AND DESIGN 7](#_Toc204969065)

[3.1. System Analysis 7](#_Toc204969066)

[3.1.1. Requirement Analysis 7](#_Toc204969067)

[3.1.2. Feasibility Study 8](#_Toc204969068)

[3.1.3. Object Modelling 11](#_Toc204969069)

[3.1.4. Dynamic Modeling (State & Sequence diagram) 12](#_Toc204969070)

[3.1.5 Process modelling: Activity Diagram 13](#_Toc204969071)

[3.2. System Design 15](#_Toc204969072)

[3.2.1. Refinement of Classes and Object 15](#_Toc204969073)

[3.2.2. Component Diagram 16](#_Toc204969074)

[3.2.3. Deployment Diagram 17](#_Toc204969075)

[3.3. Algorithm Details 18](#_Toc204969076)

[CHAPTER 4: IMPLEMENTATION AND TESTING 21](#_Toc204969077)

[4.1. Implementation 21](#_Toc204969078)

[4.1.1. Tools Used 21](#_Toc204969079)

[4.1.2. Implementation Details of Modules 21](#_Toc204969080)

[4.2. Testing 28](#_Toc204969081)

[4.2.1. Test Cases for Unit Testing 28](#_Toc204969082)

[4.2.2. Test Cases for System Testing 29](#_Toc204969083)

[CHAPTER 5: CONCLUSION AND FUTURE RECOMMENDATIONS 30](#_Toc204969084)

[5.1. Conclusion 30](#_Toc204969085)

[5.2 Lesson Learnt / Outcome 30](#_Toc204969086)

[5.3. Future Recommendation 30](#_Toc204969087)

[REFERENCES 31](#_Toc204969089)

**TABLE OF FIGURES**

[Figure 1.1: Iterative Waterfall Model of FraudShield 3](#_Toc204412573)

[Figure 3.1: Use Case Diagram 8](#_Toc204412574)

[Figure 3.2: Gannt Chart of FraudShield 10](#_Toc204412575)

[Figure 3.3: Class Diagram of FraudShield 11](#_Toc204412576)

[Figure 3.4: State Diagram of FraudShield 12](#_Toc204412577)

[Figure 3.5: Sequence Diagram of FraudShield 13](#_Toc204412578)

[Figure 3.6: Activity Diagram of FraudShield 14](#_Toc204412579)

[Figure 3.7: Refinement of Class Diagram of FraudShield 15](#_Toc204412580)

[Figure 3.8: Refinement of Object Diagram of FraudShield 16](#_Toc204412581)

[Figure 3.9: Component Diagram of FraudShield 16](#_Toc204412582)

[Figure 3.10: Deployment Diagram of FraudShield 17](#_Toc204412583)

**LIST OF TABLES**

[Table 3.1: Activity Table of FraudShield 10](#_Toc202066461)

[Table 4.1: Unit Testing 27](#_Toc202066462)

[Table 4.2: System Testing 29](#_Toc202066463)

# LIST OF ABBREVIATIONS

API – Application Programming Interface

BCA: Bachelor of Computer Applications

CSS: Cascading Style Sheets

DBMS – Database Management System

HTML: Hypertext Markup Language

IP – Internet Protocol

JS – JavaScript

ML – Machine Learning

VPN – Virtual Private Network

# CHAPTER 1:

# INTRODUCTION

## 1.1 Introduction

FraudShield is a real-time fraud detection system designed to help small online sellers to identify and prevent suspicious transactions. In today's digital commerce landscape, online fraud has become a significant challenge for businesses of all sizes, with fraudulent transactions costing merchants billions annually. Small and medium-sized businesses are particularly vulnerable as they often lack the resources to implement modern fraud prevention systems.

This project addresses this gap by providing an accessible, transparent fraud detection solution. FraudShield analyzes transaction and user behavior patterns to provide a fraud score, decision, and clear explanation for each case. Unlike black-box machine learning systems, it uses rule-based detection that merchants can understand and trust. The system checks for common fraud indicators such as disposable email addresses, suspicious IP addresses, unusual transaction patterns, and card reuse across multiple accounts.

The system emphasizes three core principles: speed, transparency, and easy integration. Transactions are analyzed in real-time, typically within milli or seconds, allowing merchants to make immediate decisions. Each fraud decision comes with detailed reasoning, explaining which rules were triggered and why. Integration is simplified through a lightweight JavaScript snippet that can be added to any checkout page.

Built using Python Flask for the backend API, MongoDB for data storage, and vanilla JavaScript for the frontend, FraudShield operates as a practical solution for businesses seeking basic fraud protection. This report outlines the project's objectives, system architecture, implementation process, testing results, and future development possibilities.

## 1.2. Problem Statement:

In today’s online world, fraud isn’t just a technical issue — it’s a daily headache for small and medium-sized businesses. Imagine a seller wakes up to see that someone used a fake identity, a stolen credit card, or a VPN to place a high-value order — only for the payment to later fail or get disputed. By the time the seller realizes, the damage is done.

Most fraud detection tools out there are either too expensive, too complex, or act like a black box — you don’t know why a transaction was flagged.

FraudShield aims to solve this by offering a fast, transparent, and easy-to-integrate fraud detection system. It helps store owners catch shady transactions before they cause harm, and shows exactly why a transaction is marked suspicious — no guesswork, no mystery.

## 1.3. Objectives:

FraudShield is a stateless, real-time fraud detection platform for online sellers. It is designed to:

* To implement a fraud detection system that provides risk scores based on predefined security rules and delivers transaction decisions within acceptable response time.

## 1.4. Scope and Limitation

### 1.4.1. Scope

This project focuses on building a lightweight, rule-based fraud detection system designed for small to medium-sized online stores. The system analyses basic transaction and user behaviour data to detect fraud patterns such as VPN usage, price tampering, and reused cards.

It includes:

* A backend engine for fraud scoring and decision making
* A ruleset that explains why a transaction is flagged
* Integration through a JavaScript snippet for real-time use
* Testing with simulated fraud scenarios

The project does not cover machine learning, or payment processing on real e-commerce platforms.

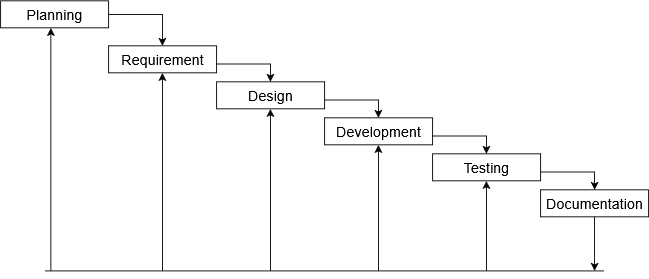
### 1.4.2 Limitations

While FraudShield is designed to be effective and practical, it has the following limitations:

* The system relies on predefined rules, which may miss new or evolving fraud techniques.
* Detection accuracy depends on the quality of input data; incomplete or fake data may reduce effectiveness.
* It requires transaction and user data from the checkout page or e-commerce site to function, which may not always be available.
* No support for machine learning or adaptive behavior in this version.

## 1.5. Development Methodology

The Iterative Waterfall Model is a hybrid software development lifecycle (SDLC) approach that combines the structured, sequential phases of the traditional Waterfall Model with iterative elements to allow for flexibility and incremental improvements.



**Figure 1: Iterative Waterfall Model of FraudShield**

## 1.6. Report Organization

**Chapter 1**: Provides an overview of online fraud challenges in e-commerce and introduces FraudShield as a rule-based solution. Defines project objectives, establishes the scope of fraud detection capabilities, and acknowledges system limitations.

**Chapter 2**: Examines existing fraud detection approaches, comparing rule-based systems with machine learning solutions. Reviews current fraud patterns in e-commerce and analyzes gaps in affordable fraud prevention for small businesses.

**Chapter 3**: Presents the technical architecture including Flask API design, MongoDB database schema, and fraud scoring algorithm. Details the implementation of core modules: authentication system, rule engine, bulk processing, and admin dashboard.

**Chapter 4**: Documents unit testing of individual components, integration testing of the complete system, and performance benchmarks. Presents fraud detection accuracy results using sample transaction datasets.

**Chapter 5**: Summarizes project achievements against stated objectives, discusses practical applications for small businesses, and identifies areas for enhancement including machine learning integration and real-time pattern updates.

# CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW

## 2.1. Background Study

Fraud detection refers to identifying abnormal or malicious behaviour during online transactions. In e-commerce, fraud can appear in many forms — from stolen payment details to fake identities or automated bots exploiting the checkout process.

This project uses a rule-based detection model, where specific rules are written to flag known suspicious behaviours. It avoids machine learning and instead provides clear, logic-driven decisions with reasons.

Key concepts and terminologies involved include:

* **VPN/Proxy Detection** – used to mask real user location, often a red flag
* **Temporary Email Identification** – spotting disposable or one-time emails often used in scams
* **Fake Address Detection** – checking for mismatched or unverifiable billing/shipping addresses
* **Card BIN Analysis** – verifying card origin, type, and legitimacy using the BIN number
* **Device Fingerprinting** – tracking repeated use of the same device in multiple transactions
* **Fast Checkout Behaviour** – identifying bots or scripts based on abnormal speed
* **IP and Geo-Location Check** – ensuring the IP address aligns with the expected country or region

These elements are processed in real time using backend logic, supported by MongoDB, and triggered via a JavaScript snippet embedded on checkout pages.

## 2.2. Literature Review

Fraud detection in online transactions has been the focus of various research studies, primarily cantered around machine learning techniques. Mitchell et al. pro [1]posed a neural network-based model for real-time credit card fraud detection, demonstrating strong accuracy in classifying suspicious transactions. However, their approach required large amounts of labelled data and lacked transparency in how decisions were made, making it less practical for small to mid-sized businesses. Similarly, Mendez and Stein in [2] introduced an ensemble model combining multiple ML algorithms, which improved detection rates but significantly increased system complexity and computational overhead.

In contrast, Müller and Park [3] explored a rule-based method to evaluate browser behaviour and detect fraud at the frontend level. While their approach was simpler and more interpretable than ML models, it lacked real-time responsiveness and was not designed for direct integration into active e-commerce checkout flows. Most publicly available tools, such as CC checkers or black-box APIs, also fall short in terms of explanation, customization, and real-time usability.

From the reviewed literature, it is evident that there is a gap in the availability of lightweight, explainable fraud detection systems tailored for small businesses. Existing models are either too complex to implement without technical expertise or too opaque to trust in high-risk environments. This highlights the need for a system that is not only easy to integrate but also capable of providing fraud decisions with clear reasoning in real time.

FraudShield addresses this gap by offering a rule-based, transparent fraud detection solution that works instantly at the point of transaction. Unlike existing systems, it does not rely on machine learning or massive datasets, making it more accessible, explainable, and practical for businesses with limited resources.

### 2.2.1 Review of similar system

**Ahmed et al.’s Ontology-Based Fraud Detection System (2021)** [4]**:**

This system uses a set of predefined rules linked to a knowledge base to spot unusual financial activities. It can rate the seriousness of suspicious transactions, which makes it easy to understand, but the setup process is complex and not ideal for smaller organizations.

**Automated Rules Management System (ARMS, 2020)** [5]**:**

ARMS improves human‑written rules by removing unnecessary ones and re‑ordering the rest so they work more efficiently. This reduces false alarms while keeping the system accurate, though it still depends heavily on manual rule creation.

# CHAPTER 3: SYSTEM ANALYSIS AND DESIGN

## 3.1. System Analysis

### 3.1.1. Requirement Analysis

Requirement analysis is a critical step in determining the success of a system or software project. It involves identifying and documenting the essential needs and expectations that the project must fulfill during its development. These requirements serve as the foundation for designing and implementing the system effectively.

i. Functional Requirements

These are the requirements that the end user specifically demands as basic facilities that the

system should offer.

**Fraud Check API Input Data:**

{

"email": string, // user@example.com

"ip": string, // 192.168.1.1

"card\_number": string, // 4111111111111111

"price": number, // 99.99

"fingerprint": string // device\_id\_12345

}

What happens with wrong data:

- Missing email → Fraud score: 0, can't check disposable domains

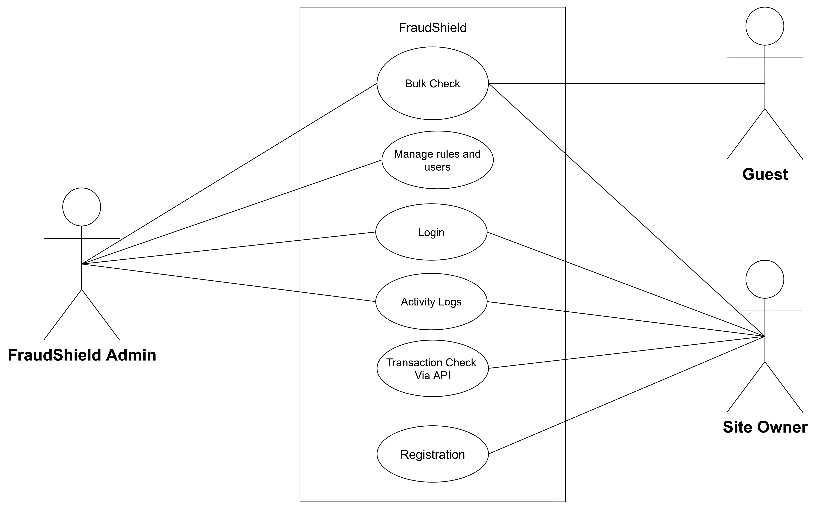
- Invalid IP → Fraud score: +0.2, treated as suspicious

- Card number too short → Can't check BIN, skips card checks

- Price as string → Converts to float or returns error

- Missing fingerprint → Can't check device reuse

Functional requirements can be represented in use case diagram.

**Use** **Case** **Diagram**

**Figure 2: Use Case Diagram**

ii. Non-functional Requirements

These are the quality constraints that the system must meet, often referred to as non-functional requirements. They do not describe specific behaviors or functions of the system core functionality.

* API must respond within 500ms (Available in test).
* System should handle multiple requests at once.
* Same input should always give same result (Available in test).
* Only authorized users can access admin panel and logs.
* Rules and backend should be easy to update.

### 3.1.2. Feasibility Study

Feasibility refers to the practicality or possibility of a proposed plan, project, or system being successful and effective. Following feasibilities were studied while building the system to ensure the system be built with exact requirements in specified time.

3.1.2.1. Technical Feasibility

The system is technically feasible using JavaScript, Python, and MongoDB. All required tools and technologies are open-source and already tested during development.

3.1.2.2. Operational Feasibility

The system is easy to use after integration. For API-based use, store owners must insert a lightweight JavaScript snippet into their checkout page. Once added, the system runs automatically without manual input, and admins can manage rules and logs through the dashboard.

3.1.2.3. Economic Feasibility

The project was built without any licensing or paid tools. Hosting and backend requirements are minimal, making it cost-effective.

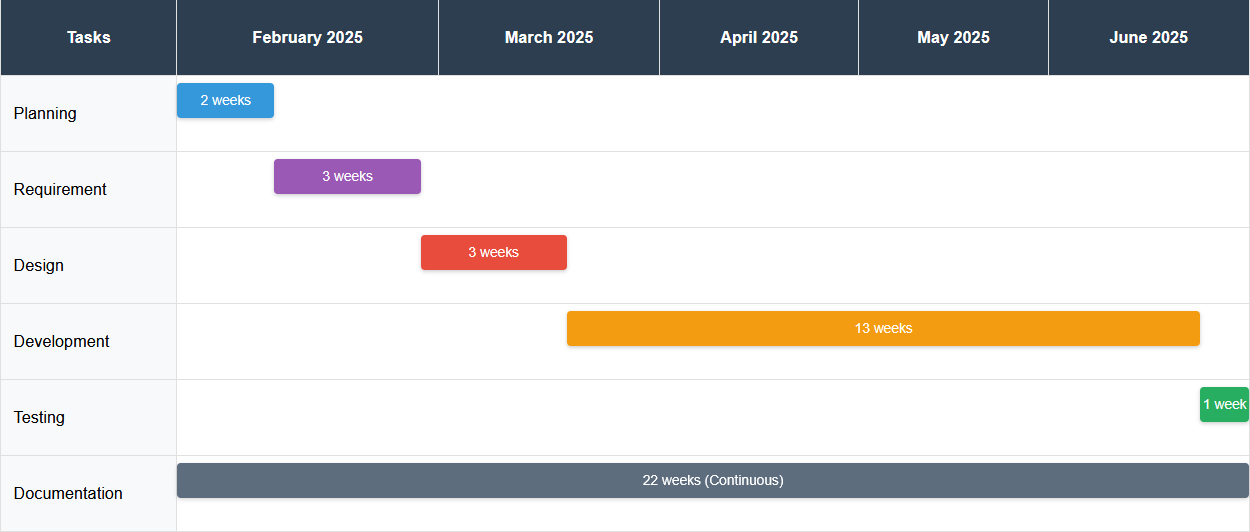
In a feasibility study, three main aspects are considered: technical, operational, and economic feasibility. Among these, a Gantt chart is most closely related to technical and operational feasibility, as it helps in planning, scheduling, and monitoring project tasks. While it does not directly measure economic feasibility, it can indirectly support cost management by identifying potential delays or resource issues.

**Gant chart:**

The purpose of a Gantt chart is to help people see and understand the schedule of a project. It shows all the tasks that need to be done, when they start, and when they finish.

**Table 1: Activity Table of FraudShield**

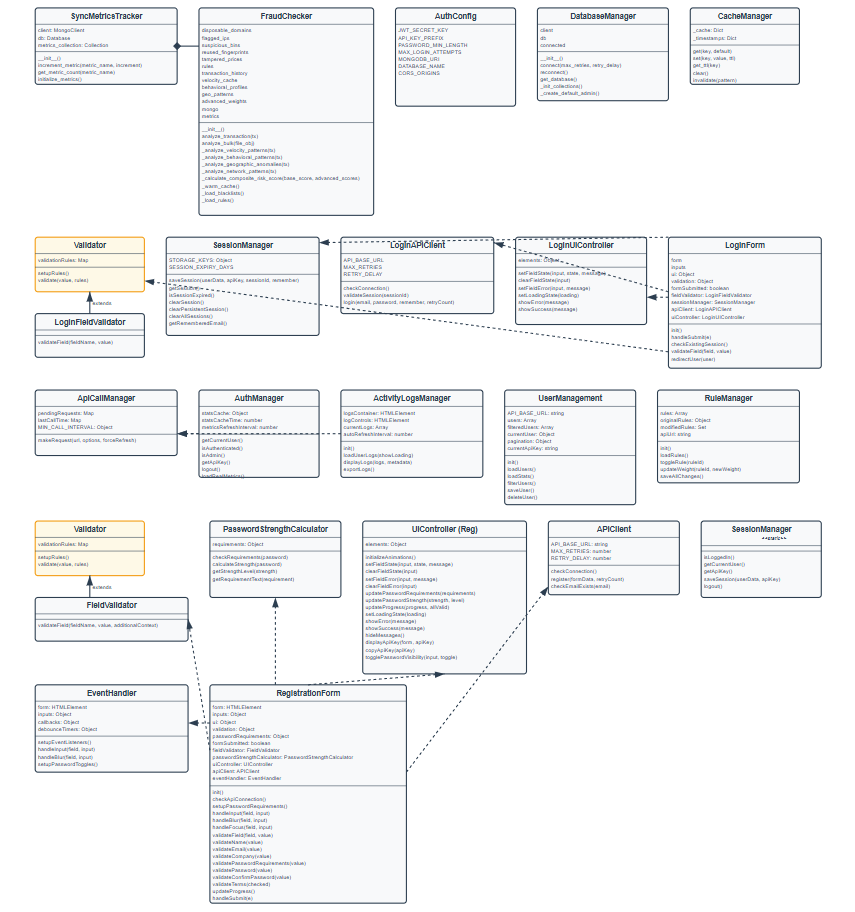




**Figure 3: Gannt Chart of FraudShield**

### 3.1.3. Object Modelling

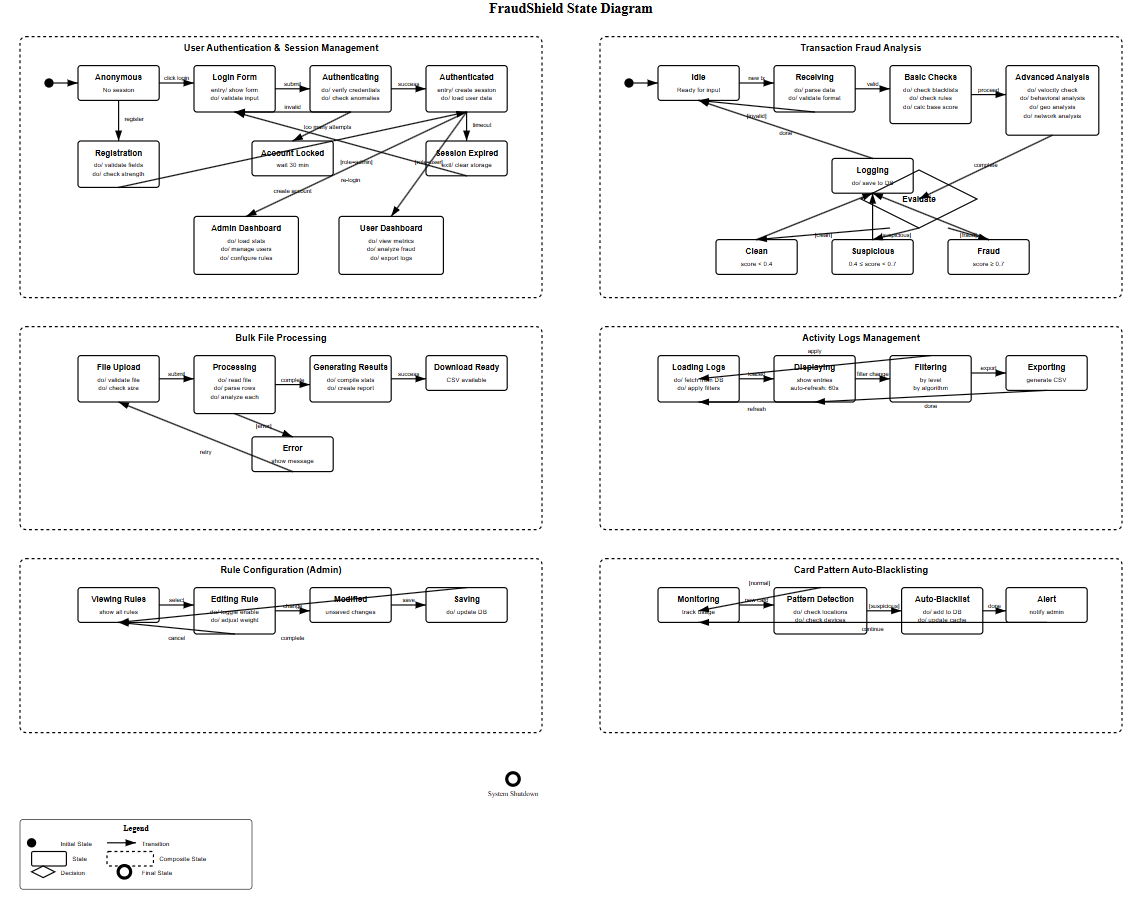
Object modelling is used to represent the static structure of the system. It includes class diagrams that define the classes, their attributes, methods, and relationships. Alongside, object diagrams show actual instances of those classes with real data. This helps in understanding how different parts of the system are connected and interact at the data level.



**Figure 4: Class Diagram of FraudShield**

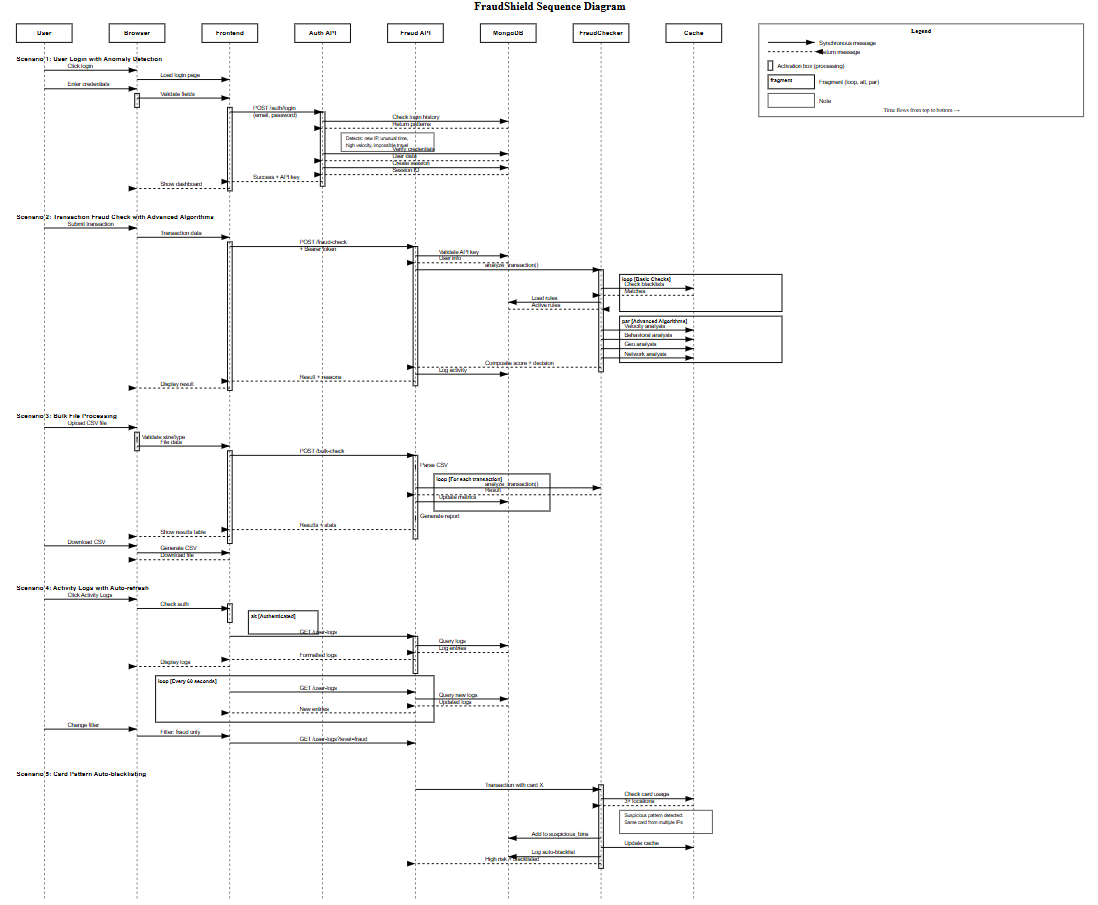
### 3.1.4. Dynamic Modeling (State & Sequence diagram)

A state diagram shows how an object changes its state based on events. It helps to track object behavior from one state to another, like Login → Dashboard → Logout. This is useful to model lifecycle and control flow in the system.



**Figure 5: State Diagram of FraudShield**

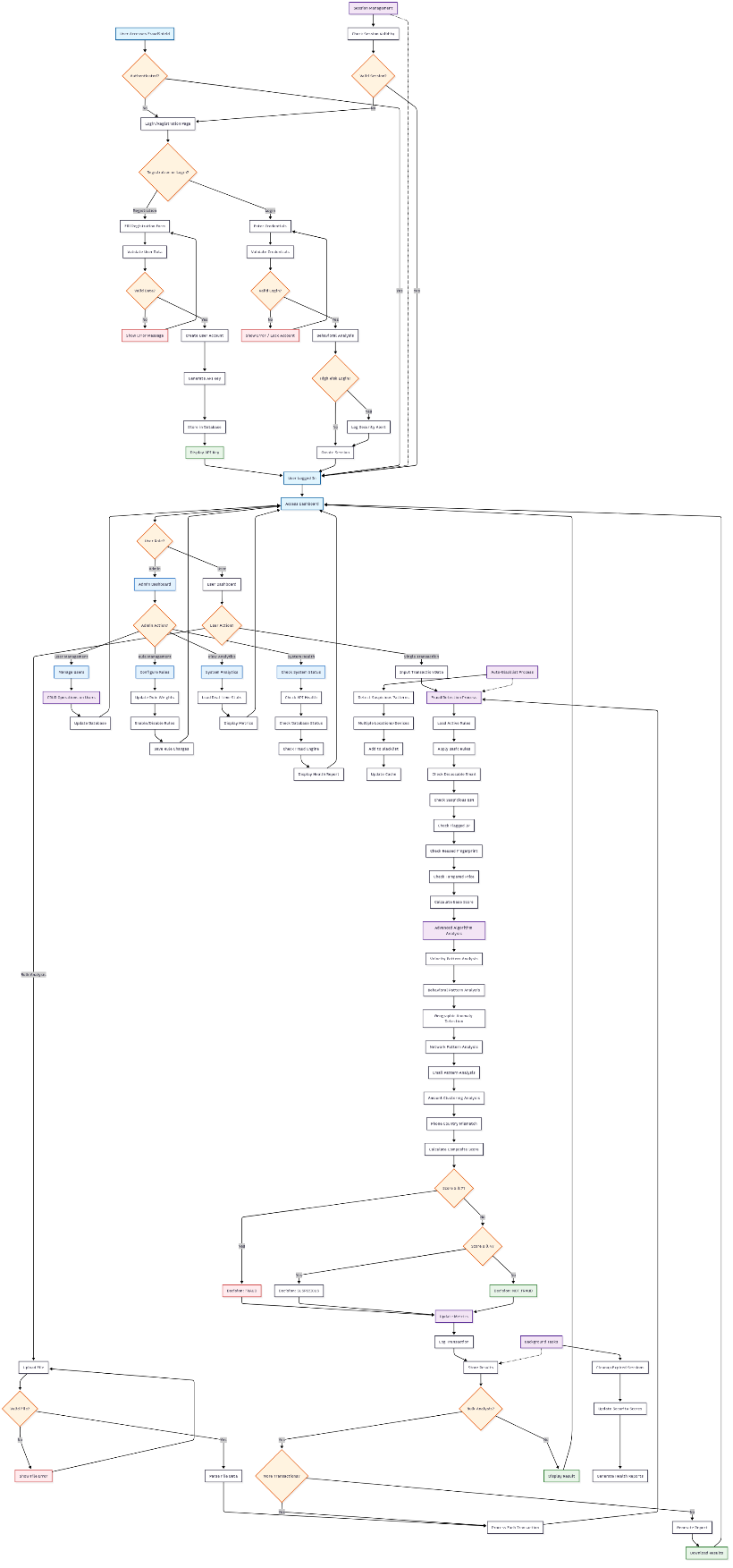
A sequence diagram shows how objects interact with each other in a specific order. It displays the flow of messages between objects over time. This helps to understand the step-by-step process of a use case.



**Figure 6: Sequence Diagram of FraudShield**

### 3.1.5 Process modelling: Activity Diagram

An activity diagram shows the flow of actions or processes in the system. It’s like a flowchart that represents different steps from start to end. This helps to understand how the system behaves during different operations.

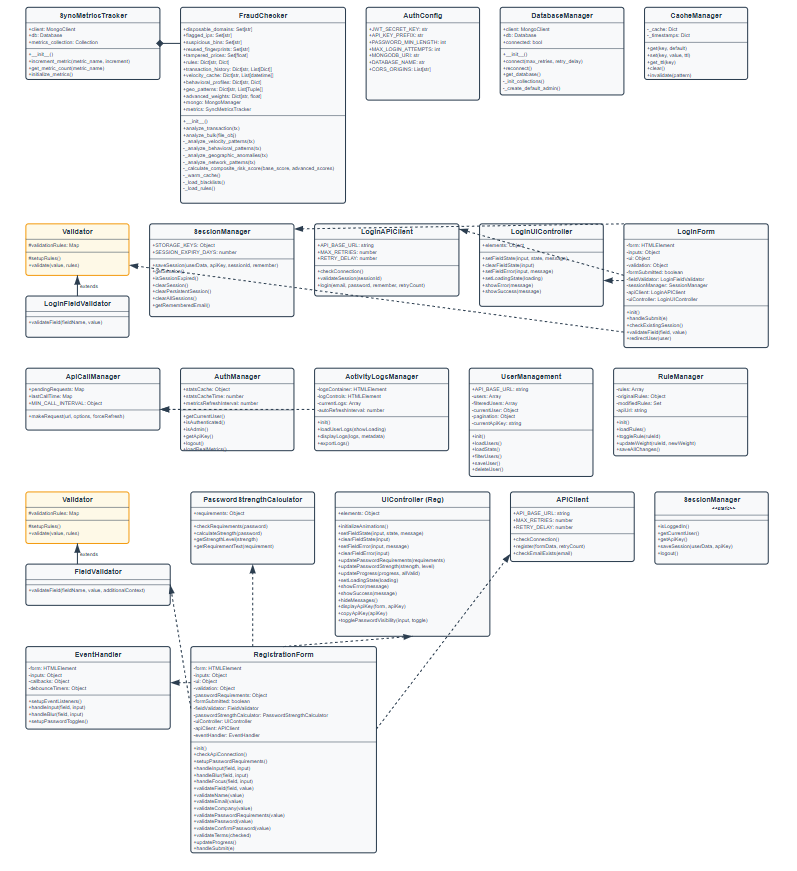


**Figure 7: Activity Diagram of FraudShield**

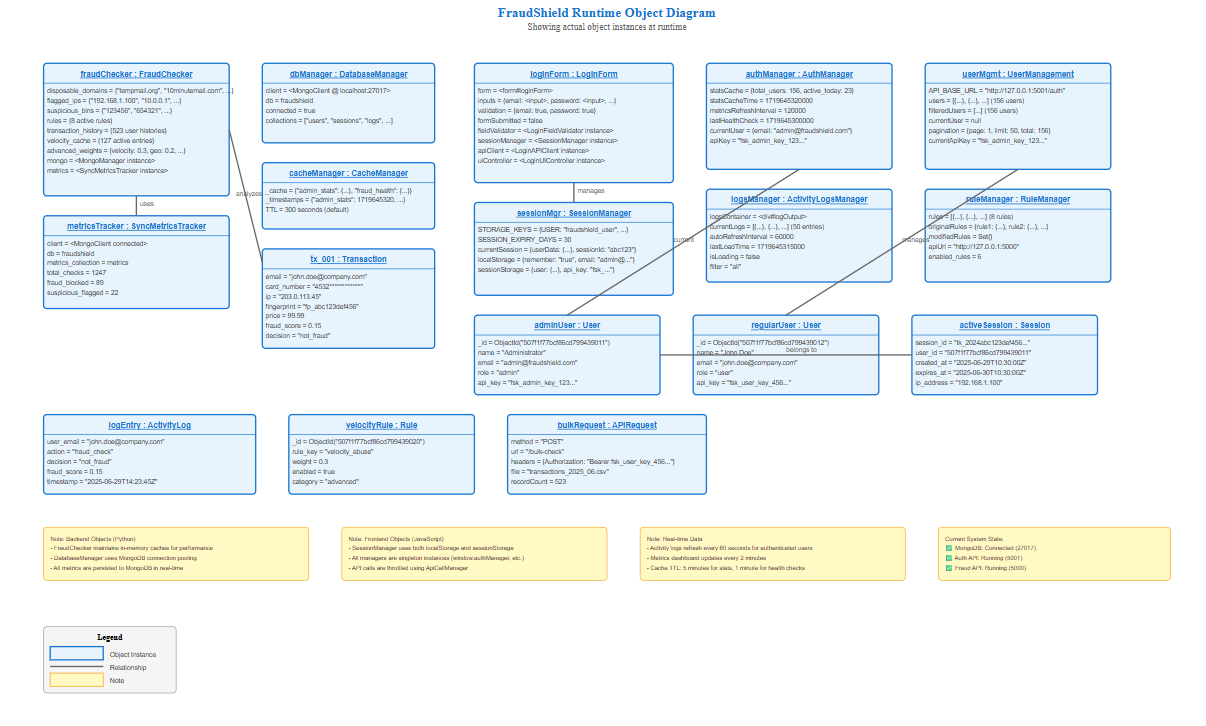
## 3.2. System Design

System design is the process of defining the architecture, components, interfaces, and data for a software system to meet specific requirements.

### 3.2.1. Refinement of Classes and Object

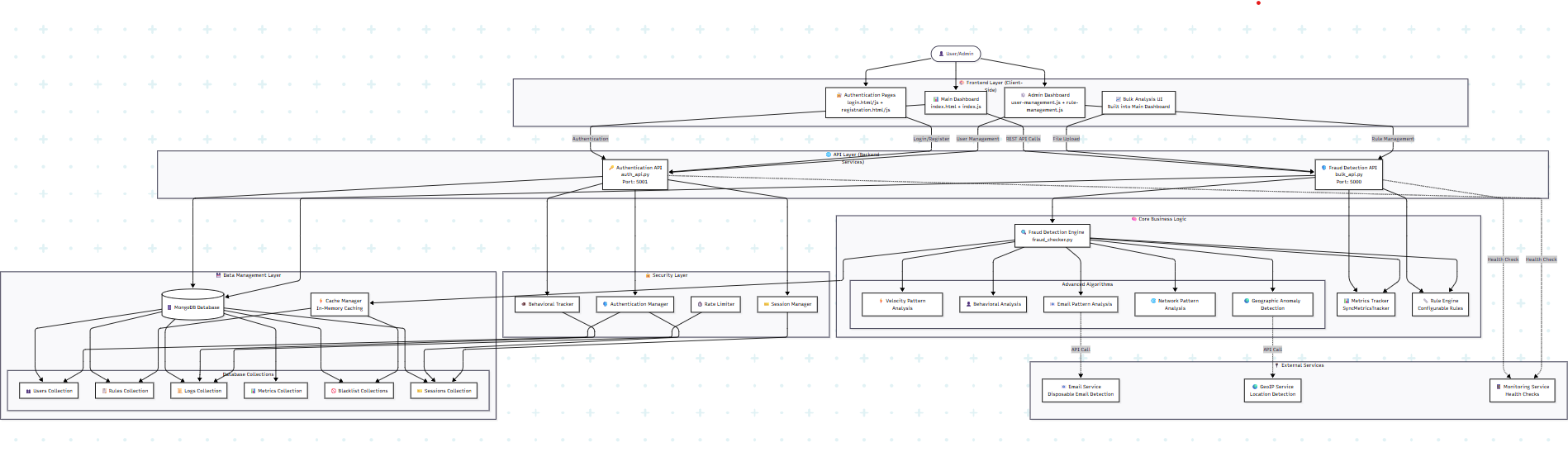


**Figure 8: Refinement of Class Diagram of FraudShield**



**Figure 9: Refinement of Object Diagram of FraudShield**

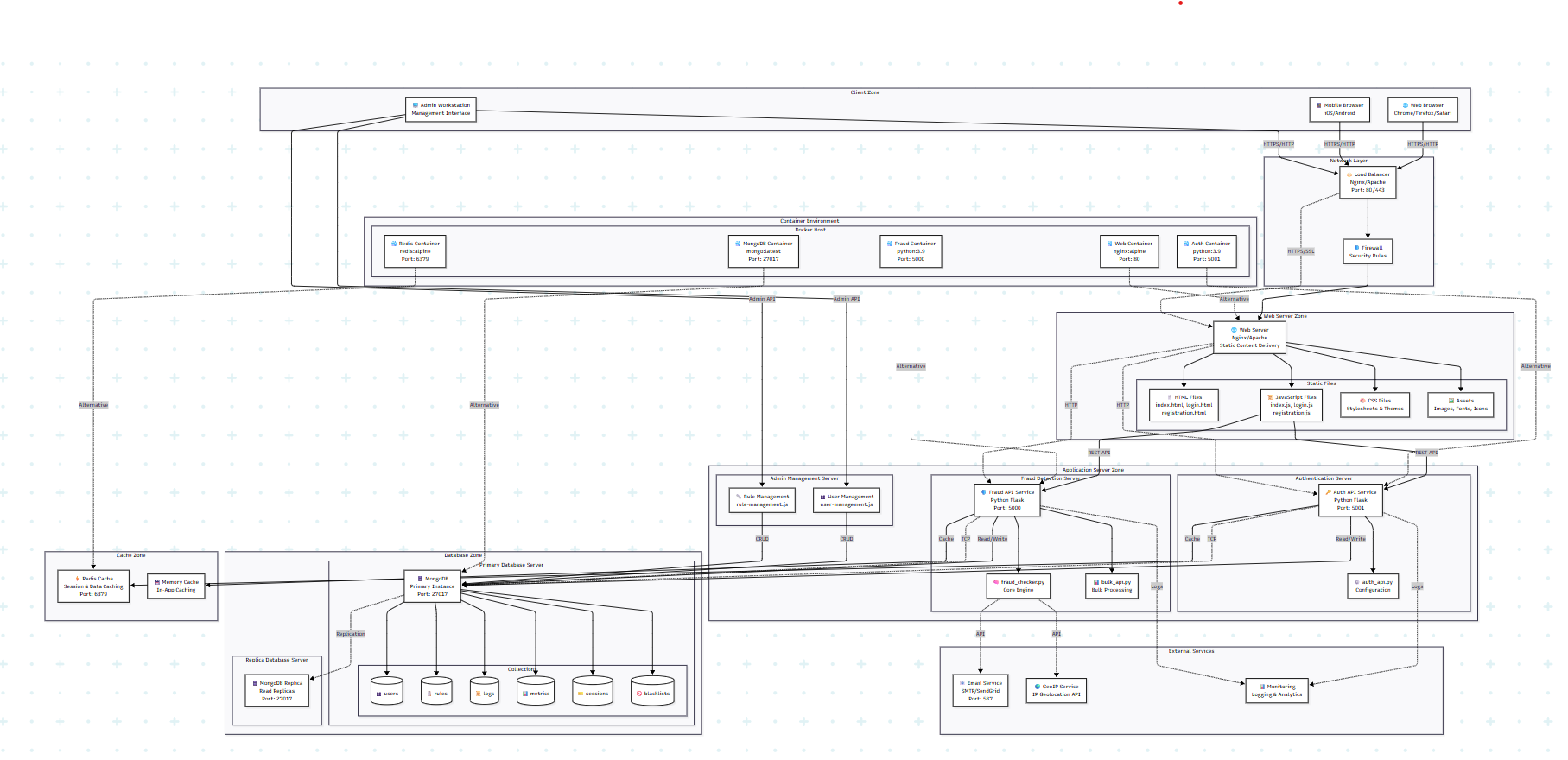
### 3.2.2. Component Diagram

A component diagram shows the main parts (components) of the system and how they interact. It represents modules like login, database, or fraud detection as separate blocks. This helps in visualizing the system’s structure at a higher level.

**Figure 10: Component Diagram of FraudShield**

### 3.2.3. Deployment Diagram

A deployment diagram shows how the system is physically deployed on hardware. It includes nodes like servers, databases, and client devices, and how they are connected. This helps to understand where and how each part of the system runs.



**Figure 11: Deployment Diagram of FraudShield**

## 3.3. Algorithm Details

**3.3.1 3-Velocity Pattern Detection Algorithm**

**Purpose:** This catches fraudsters who try many transactions quickly.

**Algorithm Steps:**

1. Store each transaction timestamp for user email
2. When new transaction comes, count how many in:
   * Last 1 minute (limit: 3)
   * Last 5 minutes (limit: 5)
   * Last 1 hour (limit: 10)
3. If over limit, add 0.3 to fraud score
4. Delete timestamps older than 24 hours

**Code:**

for window\_name, window\_delta in time\_windows.items():

cutoff = current\_time - window\_delta

recent\_txs = [t for t in self.velocity\_cache[email] if t >= cutoff]

threshold = thresholds[window\_name]

if len(recent\_txs) > threshold:

velocity\_score = min(0.3, (len(recent\_txs) - threshold) \* 0.05)

score += velocity\_score

**Location:** \_analyze\_velocity\_patterns() in fraud\_checker.py

**3.3.2 Behavioral Deviation Algorithm**

**Purpose:** Checks if transaction amount is abnormal for this user.

**Algorithm Steps:**

1. Get user's transaction history (last 30 days)
2. Calculate average amount and standard deviation
3. If current amount is more than 3 standard deviations from average, flag it
4. Add 0.15 to fraud score if flagged

**Formula:** |current\_amount - average| > 3 × standard\_deviation

**Code:**

if std\_amount > 0 and abs(current\_amount - avg\_amount) > 3 \* std\_amount:

score += 0.15

if hour\_counts[current\_hour] == 0 and len(historical\_hours) > 5:

score += 0.1

if current\_ip and current\_ip not in historical\_ips and len(historical\_ips) < 3:

score += 0.1

**Location:** \_analyze\_behavioral\_patterns() in fraud\_checker.py

**3.3.3 Composite Score Calculation**

**Purpose:** Combines all rule scores into final fraud score.

**Algorithm Steps:**

1. Add all rule weights (base score)
2. Add all algorithm scores (advanced score)
3. Final = (base × 0.7) + (advanced × 0.3)
4. If many rules triggered, multiply score by 1.1 to 1.3
5. Return final score (0.0 to 1.0)

**Code:**

total\_advanced\_score = sum(score \* self.advanced\_weights.get(algo, 0.1) for algo, score in advanced\_scores.items())

if base\_score >= 0.6:

composite\_score = base\_score \* 0.7 + total\_advanced\_score \* 0.3

elif base\_score >= 0.3:

composite\_score = base\_score \* 0.6 + total\_advanced\_score \* 0.4

else:

composite\_score = base\_score \* 0.5 + total\_advanced\_score \* 0.5

rules\_triggered = len(advanced\_scores)

if rules\_triggered >= 10:

composite\_score \*= 1.3

elif rules\_triggered >= 7:

composite\_score \*= 1.2

elif rules\_triggered >= 5:

composite\_score \*= 1.15

elif rules\_triggered >= 3:

composite\_score \*= 1.1

else:

composite\_score \*= 1.05

if base\_score >= 0.8:

composite\_score = max(composite\_score, 0.7)

elif base\_score >= 0.6:

composite\_score = max(composite\_score, 0.5)

elif base\_score >= 0.4:

composite\_score = max(composite\_score, 0.35)

final\_score = min(composite\_score, 0.99)

**Location:** \_calculate\_composite\_risk\_score() in fraud\_checker.py

# CHAPTER 4: IMPLEMENTATION AND TESTING

## 4.1. Implementation

This phase includes writing the code for the system, creating the user interface, and setting up the database. Various programming languages and tools are used to build each part of the system.

### 4.1.1. Tools Used

* Programming Languages: HTML, CSS, JavaScript (for frontend), Python Flask (for backend logic), and MongoDB (for database) for system's development.
* Development Environment: Tools such as VS Code for coding, draw.io for diagrams, git for version control, and Firefox as browser.

### 4.1.2. Implementation Details of Modules

**fraud\_checker.py**

**SyncMetricsTracker**

**Code:**

result = self.metrics\_collection.update\_one(

{"\_id": metric\_name},

{

"$inc": {"count": increment},

"$set": {"last\_updated": datetime.now()}

},

upsert=True

)

* Tracks fraud detection metrics in MongoDB synchronously
* Key methods: increment\_metric(), get\_metric\_count()

**FraudChecker**

**Code:**

total\_rules = len(all\_reasons)

if total\_rules >= 5:

# Many rules triggered - boost the score significantly

rule\_penalty = min(0.05 \* total\_rules, 0.5)

composite\_score = min(composite\_score + rule\_penalty, 0.99)

# Ensure minimum scores for many violations

if total\_rules >= 12:

composite\_score = max(composite\_score, 0.95)

elif total\_rules >= 10:

composite\_score = max(composite\_score, 0.9)

elif total\_rules >= 8:

composite\_score = max(composite\_score, 0.85)

elif total\_rules >= 6:

composite\_score = max(composite\_score, 0.75)

* Core fraud detection engine with multiple algorithms
* Key methods: analyze\_transaction(), analyze\_bulk()
* Maintains blacklists and implements velocity, behavioral, and geographic analysis

**bulk\_api.py**

**Config**

* API configuration (file size limits, allowed extensions)

**DatabaseManager**

**Code:**

    mongo\_client = pymongo.MongoClient("mongodb://localhost:27017")

    auth\_db = mongo\_client.fraudshield

    users\_collection = auth\_db.users

    audit\_logs\_collection = auth\_db.audit\_logs

    transactions\_collection = auth\_db.transactions

    app.logger.info(" Authentication database connected")

* MongoDB connection with retry logic
* Key methods: validate\_apikey(), get\_database()

**auth\_api.py**

**AuthConfig**

* Authentication settings

**DatabaseManager**

**Code:**

self.client = pymongo.MongoClient(

    AuthConfig.MONGODB\_URI,

    serverSelectionTimeoutMS=5000,

    connectTimeoutMS=5000,

    socketTimeoutMS=5000

)

# Test connection and db access

self.client.server\_info()

self.db = self.client[AuthConfig.DATABASE\_NAME]

self.db.users.count\_documents({}, limit=1)

self.connected = True

logger.info("MongoDB connection established successfully")

# Initialize collections if needed

self.\_init\_collections()

return True

* Auth database connection and initialization
* Creates indexes and default admin user

**CacheManager**

    def get\_ttl(self, key: str) -> int:

        """Get TTL for specific cache keys"""

        if 'admin\_stats' in key:

            return 300  # 5 minutes for admin stats

        elif 'fraud\_health' in key:

            return 60   # 1 minute for health checks

        elif 'user\_stats' in key:

            return 180  # 3 minutes for user stats

        elif 'fraud\_detection\_stats' in key:

            return 120  # 2 minutes for fraud stats

        return 300  # Default 5 minutes

* In-memory cache with TTL support
* Caches stats and health checks

**index.js**

**ApiCallManager**

**Code:**

// Check if we're calling too frequently

if (!forceRefresh) {

    const lastCall = this.lastCallTime.get(fullUrl);

    const minInterval = this.MIN\_CALL\_INTERVAL[fullUrl] || 10000;

    if (lastCall && (Date.now() - lastCall) < minInterval) {

console.log(`⏳ Skipping ${fullUrl} - called too recently (${Math.round((Date.now() - lastCall) / 1000)}s ago)`);

return null;

    }

}

* Prevents API spam with request deduplication
* Enforces minimum call intervals

**AuthManager**

**Code:**

    static isAuthenticated() {

        // Check both sessionStorage and localStorage

        const sessionUser = sessionStorage.getItem('fraudshield\_user');

        const sessionApiKey = sessionStorage.getItem('fraudshield\_api\_key');

        const persistentUser = localStorage.getItem('fraudshield\_persistent\_user');

        const persistentApiKey = localStorage.getItem('fraudshield\_persistent\_api\_key');

        return !!(

            (sessionUser && sessionApiKey) ||

            (persistentUser && persistentApiKey)

        );

    }

* Dashboard authentication and role management
* Handles metrics loading and user interface

**ActivityLogsManager**

**Code:**

async loadUserLogs(showLoading = true) {

// Check if we're loading too frequently

const timeSinceLastLoad = Date.now() - this.lastLoadTime;

if (!showLoading && timeSinceLastLoad < this.minLoadInterval) {

console.log(`📋 Logs loaded ${Math.round(timeSinceLastLoad / 1000)}s ago, skipping auto-refresh`);

return;

}

* Displays and filters user activity logs
* Auto-refresh and export functionality

**login.js**

**LoginForm**

const response = await fetch(`${this.API\_BASE\_URL}/login`, {

    method: 'POST',

    headers: {

        'Content-Type': 'application/json'

    },

    body: JSON.stringify({

        email: email,

        password: password,

        remember: remember

    })

* Main login orchestrator
* Handles validation, API calls, redirects

**user-management.js**

**UserManagement**

**Code:**const searchTerm = document.getElementById('searchInput').value.trim();

const roleFilter = document.getElementById('roleFilter').value;

if (searchTerm) params.search = searchTerm;

if (roleFilter) params.role = roleFilter;

const result = await this.api.get('/users', params);

if (result.success) {

    this.users = result.data.users || [];

    this.pagination = result.data.pagination || this.pagination;

    this.filteredUsers = [...this.users];

    this.renderUsers();

} else {

    throw new Error(result.error || 'Failed to load users');

}

* Admin CRUD operations for users
* Search, filter, pagination features

**rule-management.js**

**RuleManager**

**Code:**

    rulesToRender.forEach(rule => {

        const category = rule.category || 'uncategorized';

        if (!filteredCategories[category]) {

            filteredCategories[category] = [];

        }

        filteredCategories[category].push(rule);

    });

    Object.entries(filteredCategories).forEach(([category, rules]) => {

        const section = this.createCategorySection(category, rules);

        container.appendChild(section);

    });

    }

}

* Configure fraud detection rules and weights
* Batch updates with change tracking

## 4.2. Testing

Testing is the crucial process of evaluating and verifying that a software to ensures that the application functions correctly and meets user expectations.

### 4.2.1. Test Cases for Unit Testing

**Table 2: Unit Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S.N | Description | Input | Expected Outcome | Actual Outcome | Result |
| 1 | Temporary email detection | Email: something@tempmail.com | Fraud score with reason. | Fraud score with reason. | Pass |
| 2 | Country code mismatch detection | Country code: +91<br>Billing Country: US | Flagged as "phone\_country\_mismatch", score ≥ 0.15 | No flag raised, score: 0.0 | Fail |
| 3 | Negative price validation | Price: -99.99 | Error: "Invalid price value". | Transaction proceeds | Fail |
| 4 | Normal Email | Email: bijay@gmail.com | Success | Success | Pass |
| 5 | Country Code mismatch with billing | Country code : +91  Billing Country: US | Flagged as number mismatch. | Flagged as number mismatch. | Pass |

### 4.2.2. Test Cases for System Testing

**Table 3: System Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test id | Description | Input | Expected Outcome | Actual Outcome | Result |
| 1 | Public bulk check with oversized file | Upload 20MB CSV file (limit is 16MB) | Error: "File too large. Maximum size: 16MB" | Server timeout after 30 seconds No error message | Fail |
| 2 | Fraud check via API | Valid Transaction JSON from integrated website | Fraud score with reason returned | Fraud score with reason returned | Pass |
| 3 | Database connection failure | Disconnect database during transaction processing | Graceful error: "Service temporarily unavailable" | Application crash with stack trace exposed | Fail |
| 4 | Public bulk check | Uploaded CVS with 10 email+IP entries | Results returned with fraud flags per entry | Results returned with flags per entry | Pass |
| 5 | Response Time check | Given 40 data in .json for bulk check | Less than a second | Processing time: 0.56s | Pass |
| 6 | Same data test | Same json given from 5 number | Must be same result overall in | Same analysis result (in numbers) | Pass |

# CHAPTER 5: CONCLUSION AND FUTURE RECOMMENDATIONS

## 5.1. Conclusion

FraudShield successfully fulfills its objective of detecting potentially fraudulent transactions through both manual input and API integration. The system supports secure registration, real-time fraud scoring, rule-based evaluation, and administrative control. Unit and system testing confirmed that the platform behaves as expected in various scenarios. Overall, FraudShield is a reliable and effective solution for enhancing transaction security.

## 5.2 Lesson Learnt / Outcome

Through the development of FraudShield, we gained hands-on experience in designing secure, scalable fraud detection systems. We learned how to integrate rule-based logic, manage user roles, and build an API-driven architecture. Testing highlighted the importance of edge case handling and real-time response. Overall, the project strengthened our skills in system design, backend logic, and application security.

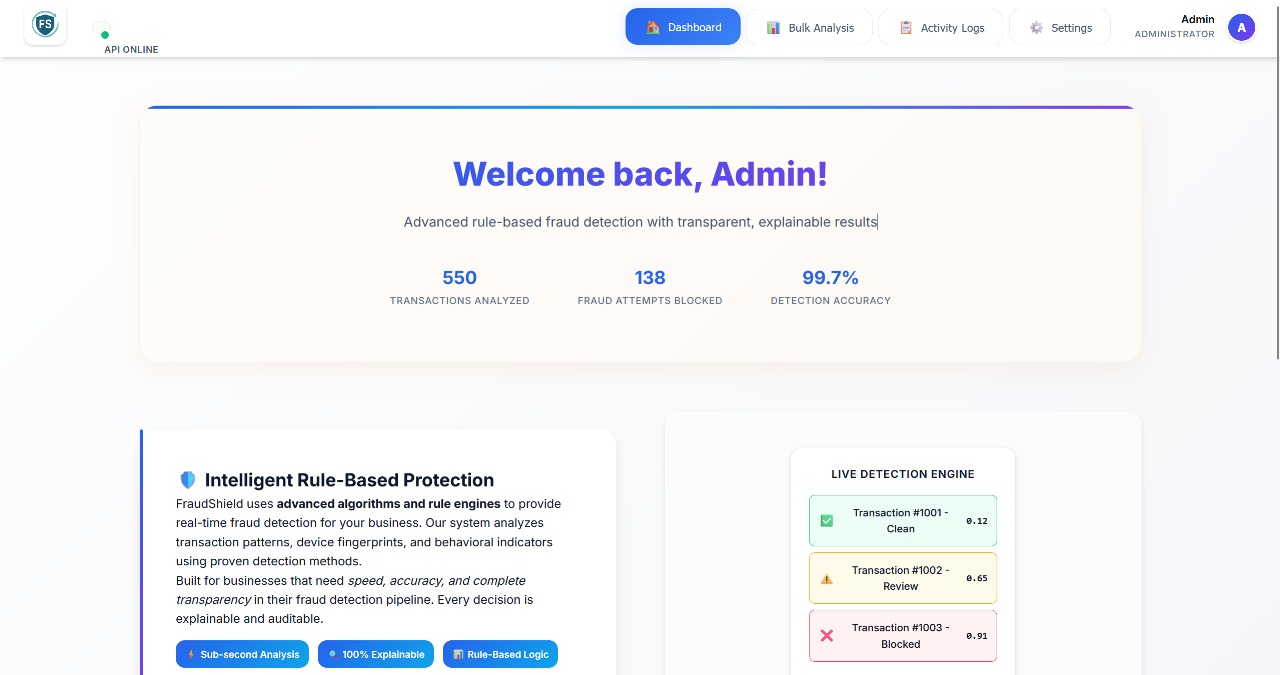
## 5.3. Future Recommendation

Down the line there are various functionalities that can be integrated into the system such as:

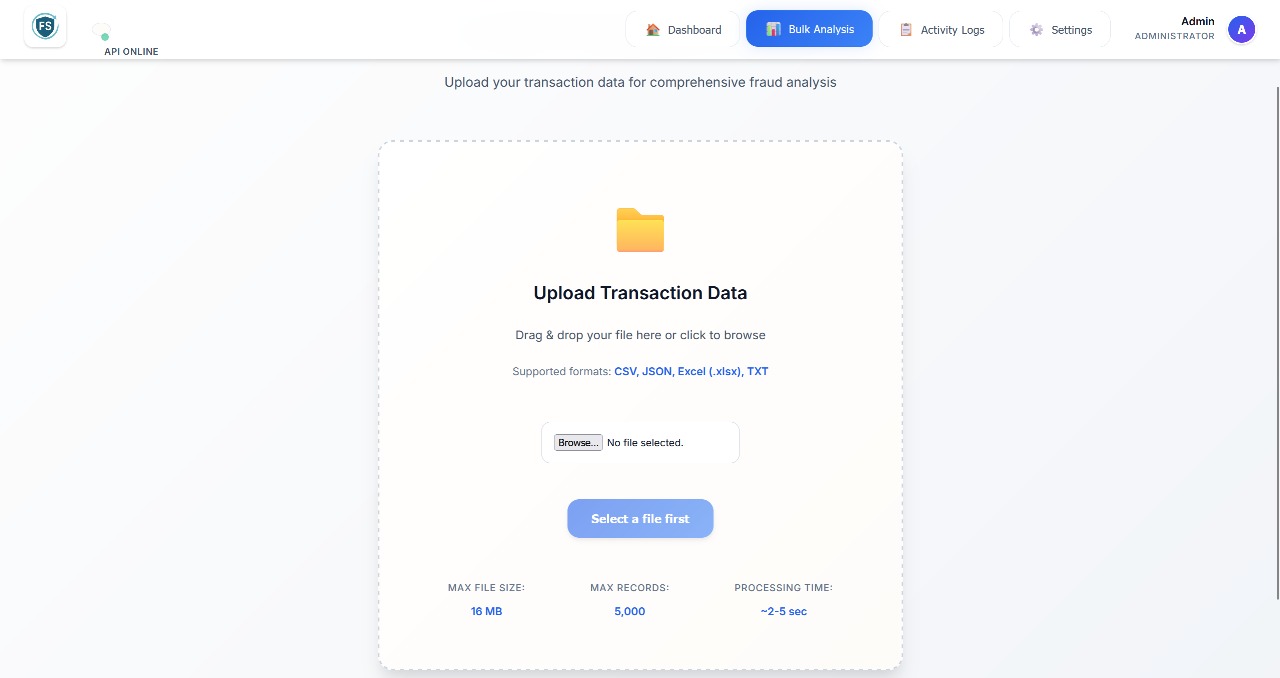
* Integrate machine learning to enhance fraud detection accuracy and adapt to evolving patterns.
* Optimize the existing codebase to improve system performance and reduce response time.
* Allow site owners to set custom fraud score thresholds based on their business needs.
* Provide site owners the ability to manually blacklist specific emails, devices, or IPs from their dashboard.

# APPENDICES

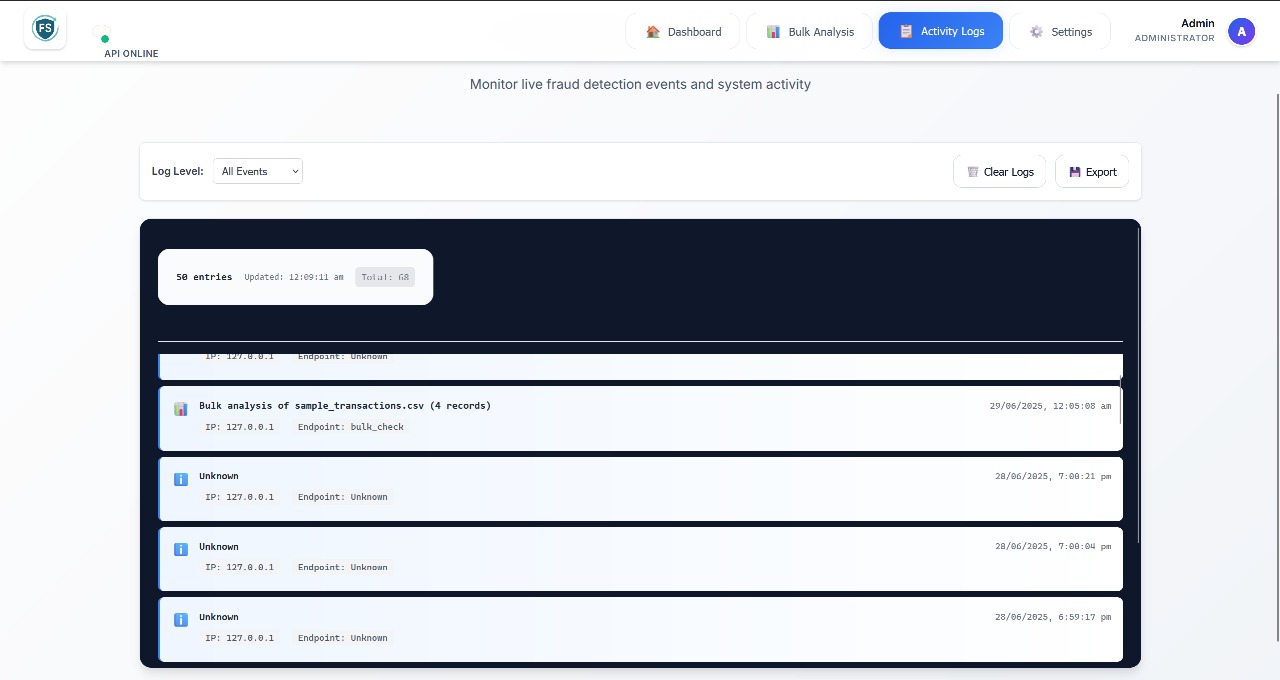
Homepage



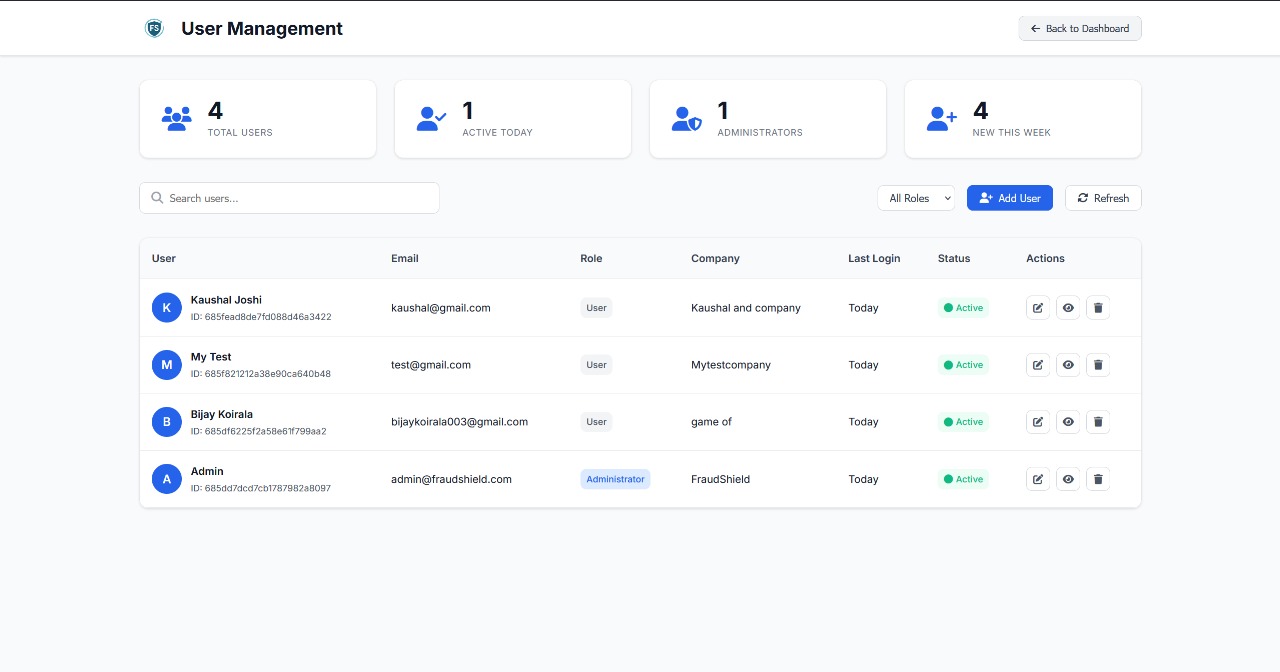
Bulk analysis page



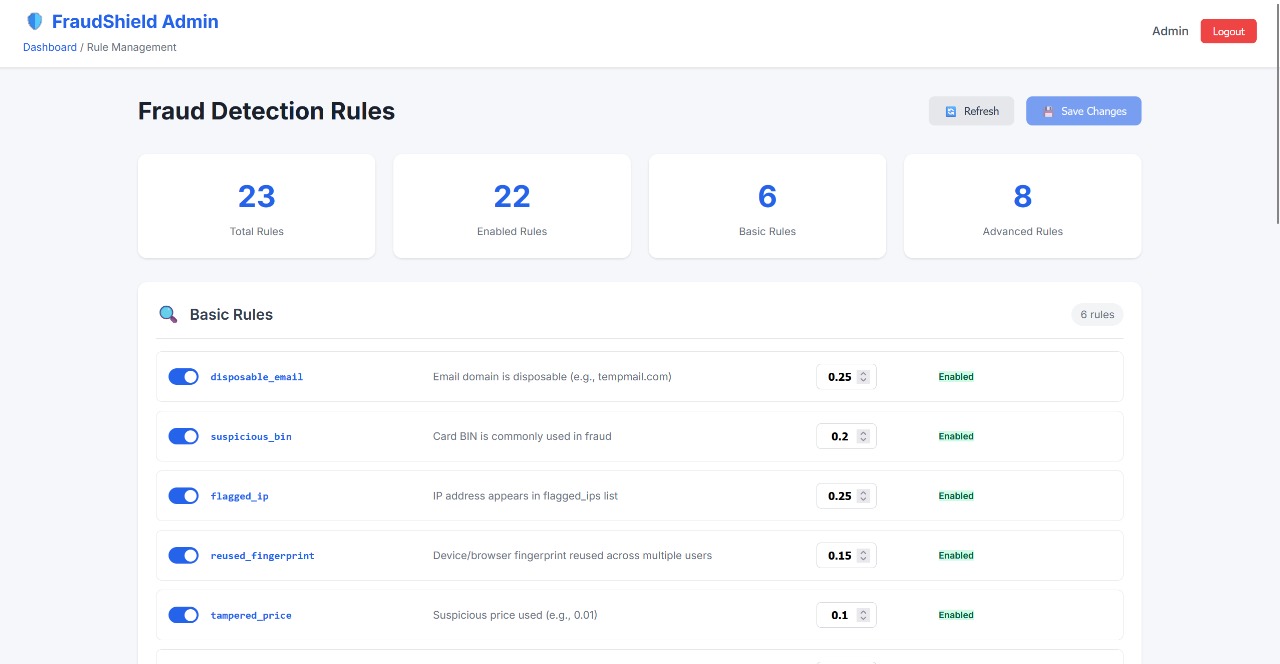
Log Page



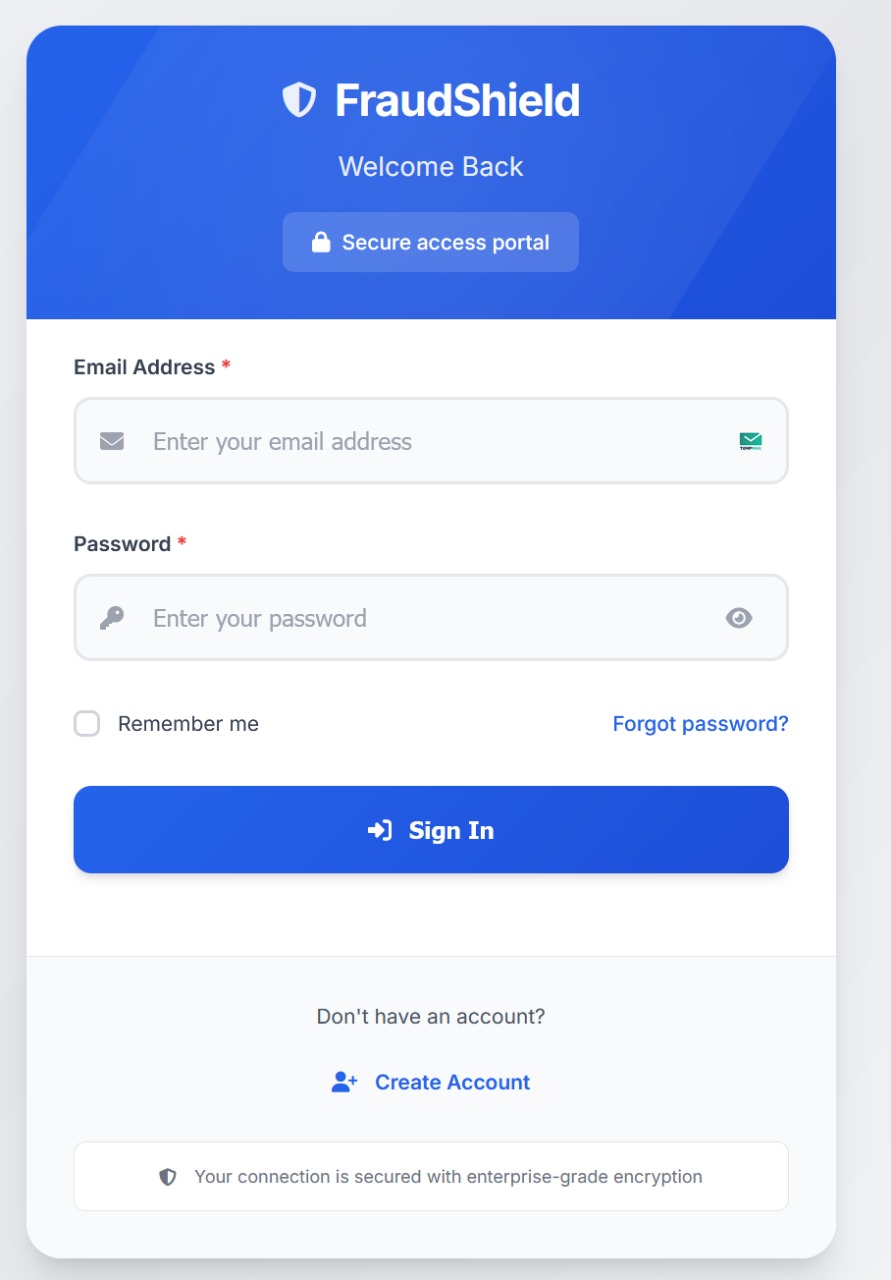
User management (admin page)



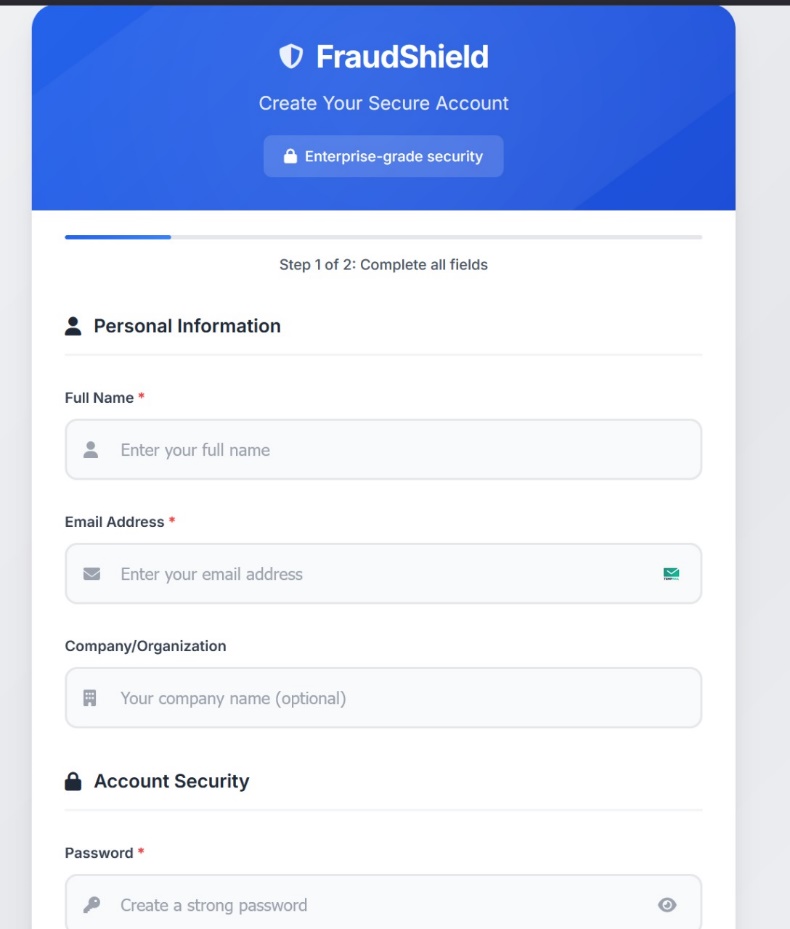
Rules management (admin page)



Login page



Registration page



# REFERENCES

|  |  |
| --- | --- |
| [1] | E. J. a. K. Y. Laura Mitchell, "Real-Time Credit Card Fraud Detection using Neural Networks," *nternational Journal of Computer Applications,* vol. 182, p. 25–29, 2021. |
| [2] | F. L. &. Octavio, "An Ensemble Approach for Financial Fraud Detection," in *IEEE Conference on Data Science and Engineering*, 2020. |
| [3] | J. M. a. C. Park, "Rule-Based Risk Evaluation for Browser Transactions," *Journal of Web Security and Analytics,* Vols. 11, No. 2, pp. 66–72, 2019. |
| [4] | M. A. K. M. C. B. K. A. A. A. &. T. M. Ahmed, "A semantic rule based digital fraud detection," *PeerJ Computer Science,* 2021. |
| [5] | D. B. R. B. J. A. J. T. &. B. P. Aparício, "Automated Rules Management System for Fraud Detection," arXiv, 2020. |