



SCHOOL OF ENGINEERING AND NATURAL SCIENCES

Final Report

Group: 7

Fraud Detection System

23.06.2025

Student Name	Student ID	Department
Erva Şengül	64240048	COE
Azra Karakaya	64210010	COE
Beril Mutlu	64220089	COE

1. Introduction

The Fraud Detection System is a web-based application developed to detect unusual financial activities by analyzing user transactions. As online payments become more common, the risk of fraud also increases. This system aims to identify suspicious behavior using rule-based checks and personalized user profiles. Each user has a profile containing typical spending patterns, such as preferred time windows, usual locations, and category limits. New transactions are compared against these patterns to detect anomalies. If a transaction appears risky, the system creates an alert and notifies the user. The project combines database design, backend logic, and user interface development. It was built using PHP and MySQL, with a responsive frontend created in HTML, CSS, and JavaScript. The system provides a solid base for future improvements like machine learning and mobile integration.

2. Methodology and System Requirements

The Fraud Detection System was developed using a full-stack web development approach, combining secure backend logic with a responsive frontend and a normalized relational database. The backend was implemented in PHP and connected to a MySQL database hosted on a local Apache server using XAMPP on macOS. All server-side operations—such as user registration, transaction processing, and anomaly detection—were performed through a RESTful API built with PHP. Passwords were securely hashed using `password_hash()`, and all SQL operations were executed using prepared statements through PDO to prevent SQL injection attacks. Each module of the system, including user management, transaction logging, anomaly tracking, and alert generation, was handled by separate PHP scripts to maintain modularity and scalability. The database schema followed normalization principles up to 3NF and enforced data integrity using primary and foreign key constraints among six main entities: User, Profile, Transaction, Category, AnomalyRecord, and Alert.

On the frontend, HTML5, CSS3, JavaScript, and Bootstrap were used to create an interactive and user-friendly interface. jQuery and AJAX were utilized to perform asynchronous operations such as form submissions and real-time dashboard updates, allowing for a smooth user experience without requiring page reloads. The interface consists of various pages, including login, registration, dashboard, transaction history, alerts, and profile settings. Users can submit new transactions, view existing ones, receive fraud alerts, and customize their fraud sensitivity preferences such as trusted locations or category spending limits. All frontend components communicate securely with the backend through JSON-formatted API requests, with CORS policies configured to prevent unauthorized access. The system was designed with future extensibility in mind, enabling the integration of mobile apps, advanced analytics, and machine learning-based fraud detection in future versions. With real-time data processing and alerting, the system demonstrates how well-structured software and database design can be applied to address real-world financial security challenges.

3. Entity-Relationship Design

3.1. Entities

USER

- UserID (primary key, unique)
- Name
- Email
- Phone
- Password (hashed)
- RegistrationDate

TRANSACTION

- TransactionID (primary key, unique)
- Amount
- Date
- Time
- Location (city/country)
- PaymentMethod (e.g., credit card, bank transfer)

CATEGORY

- CategoryID
- CategoryName

PROFILE

- AvgMonthlySpend
- CategoryLimits
- TimeWindows (usual transaction times)
- FrequentLocations

ANOMALY RECORD

- AnomalyID (primary key, unique)
- AnomalyType (e.g., high amount, unusual time, strange location)
- DetectedDate
- Status (confirmed fraud, false alarm, under investigation)

ALERT

- AlertID (primary key, unique)
- Timestamp

3.2. Relationships

The system keeps track of each user's UserID, name, email, phone number, password (stored in hashed format), and registration date. The application needs to authenticate users securely while maintaining their contact information for notification purposes. The UserID is unique for each user and serves as the primary identifier throughout the system

Each transaction is recorded with TransactionID, Amount, Date, Time, Location information (city/country), and PaymentMethod (e.g., credit card, bank transfer). The TransactionID is unique for each transaction and is automatically generated by the system. Each transaction must be associated with exactly one user (UserID as foreign key) and must belong to exactly one spending category (CategoryID as foreign key).

The category entity classifies different types of spending such as food, travel, electronics, etc. Each category has a CategoryID and a CategoryName. The CategoryID is unique for each category and is used to link transactions to their appropriate spending classification.

For each user, the system maintains a user profile that contains information about their spending patterns. This includes AvgMonthlySpend, CategoryLimits, TimeWindows (usual transaction times), and FrequentLocations. A user profile is automatically created when a user registers and is continually updated as more transaction data is processed. Each user has exactly one user profile, implemented by using UserID as both primary key and foreign key in the PROFILE entity.

When the system detects unusual activity, it creates an anomaly record. Each anomaly record includes AnomalyID (unique), AnomalyType (e.g., high amount, unusual time, strange location), DetectedDate, and Status (confirmed fraud, false alarm, under investigation). An anomaly record is always associated with exactly one transaction (TransactionID as foreign key), but a transaction may generate multiple anomaly records if it triggers different types of anomaly detection rules.

For each detected anomaly, an alert is generated and sent to the user. Each alert contains AlertID (unique) and Timestamp. Each alert is triggered by exactly one anomaly record (AnomalyID as foreign key) and is sent to exactly one user (UserID as foreign key).

The user profile data directly influences the anomaly detection process, as the system compares incoming transactions against the established patterns in the user profile to identify potential fraud. This relationship is implemented through the UserID foreign key in the ANOMALY_RECORD entity, creating a connection between the user's profile data and the anomaly detection mechanism.

The system allows users to define personalized thresholds for different spending categories within their user profile. These thresholds determine when alerts should be triggered, enabling users to adjust the sensitivity of the fraud detection system according to their preferences. The relationship between PROFILE and ANOMALY_RECORD implements this personalization feature.

The complete flow of the system follows a path from transaction creation to anomaly detection based on user profile patterns, to alert generation and delivery to the appropriate user. This interconnected structure ensures that fraud detection is both personalized and responsive to each user's unique financial behavior.

3.3. ER Diagram

The Entity-Relationship (ER) Diagram illustrates the conceptual model. It describes each entity's attributes, relationships between entities, and cardinality limitations. This visual representation supported discussion among team members and ensured a common knowledge of the system's data structure.

The ER diagram highlights key relationships, such as:

- Each User has one Profile
- Each Transaction is linked to a User and a Category
- Each AnomalyRecord is connected to one Transaction
- Each Alert corresponds to one AnomalyRecord and one Use

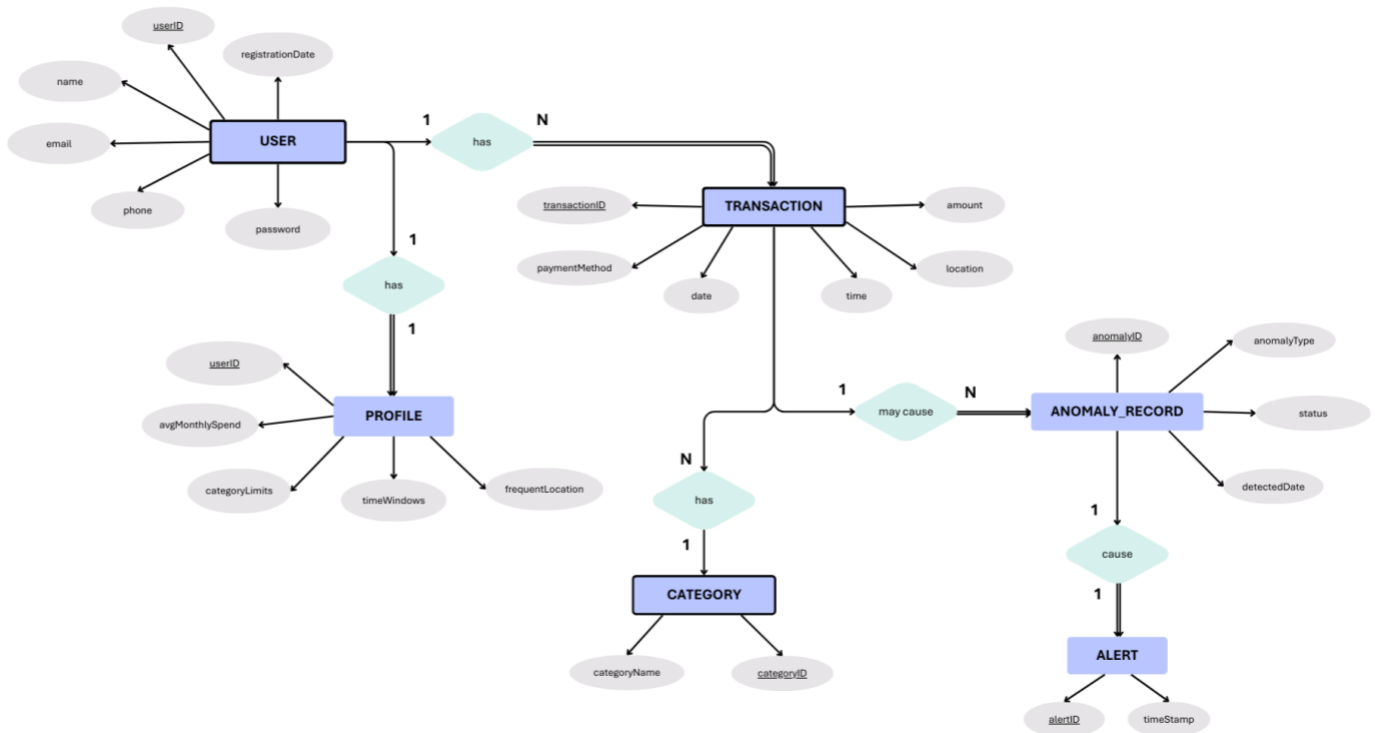


Figure 3.1. ER Diagram of the System

4. Relational Database

4.1. Relational Database Schema

- **User** (UserID, Name, Email, Phone, Password, RegistrationDate)
- **Transaction** (TransactionID, Amount, Date, Time, Location, PaymentMethod, UserID (FK), CategoryID (FK))
- **Category** (CategoryID, CategoryName)
- **Profile** (UserID (PK & FK), AvgMonthlySpend, CategoryLimits, TimeWindows, FrequentLocations)
- **AnomalyRecord** (AnomalyID, AnomalyType, DetectedDate, Status, TransactionID (FK), UserID (FK))
- **Alert** (AlertID, Timestamp, AnomalyID (FK), UserID (FK))

Primary Keys

User	: userID
Profile	: userID
Category	: categoryID
Transaction	: transactionID
Anomaly_Record	: anomalyID
Alert	: alertID

Foreign Keys

Profile.userID → User.userID

Transaction.userID → User.userID

Transaction.categoryID → Category.categoryID

Anomaly_Record.transactionID → Transaction.transactionID

Alert.anomalyID → Anomaly_Record.anomalyID

Alert.userID → User.userID

4.2. Relational Database Design

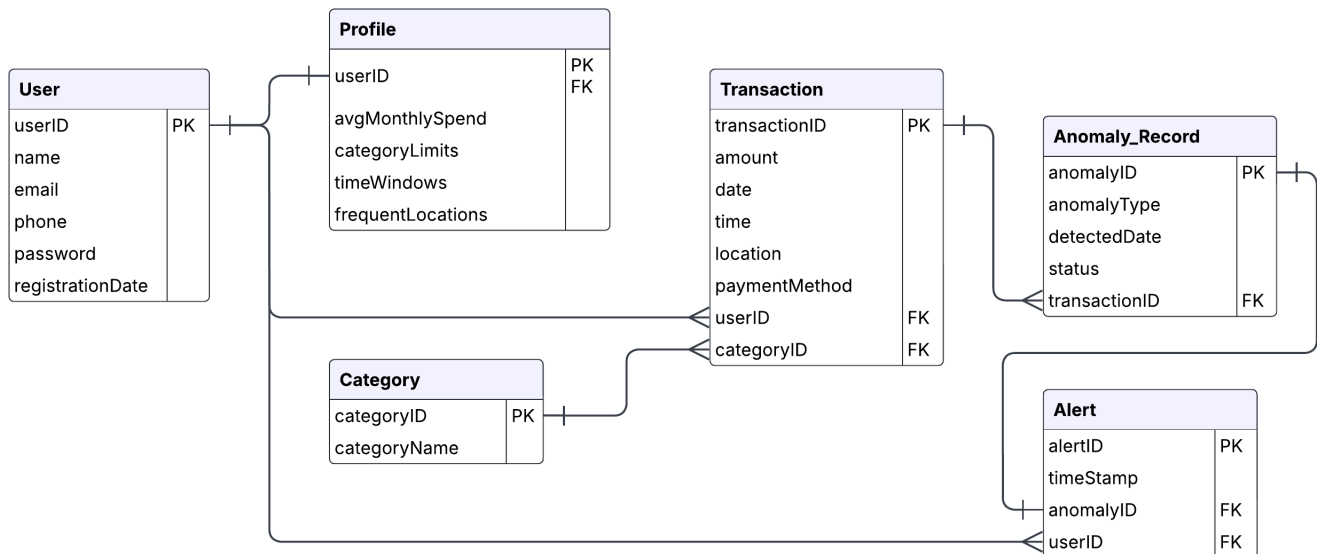


Figure 4.1. Relational Database Design

5. DDL Statements

This section covers the Data Definition Language (DDL) statements required to generate relational database tables. The SQL CREATE TABLE instructions specify each table's structure, including data types, primary keys, and foreign key constraints for referential integrity.

Using well-defined DDL statements makes sure that the database structure is compatible with the schema and supports all system features, including authentication, anomaly detection, and notification.

The DDL statements below are used to create the tables of the database. Each table is defined with its columns, data types, and primary keys to ensure that each record is unique. Foreign keys are used to link related tables together, helping the data stay connected and consistent.

```
CREATE TABLE User (  
  UserID      VARCHAR(20),  
  Name        VARCHAR(50),  
  Email       VARCHAR(50),  
  Phone       VARCHAR(20),  
  Password    VARCHAR(255),  
  RegistrationDate DATE,  
  PRIMARY KEY (UserID)  
);
```

```
CREATE TABLE Category (  
    CategoryID    VARCHAR(20),  
    CategoryName  VARCHAR(50),  
    PRIMARY KEY (CategoryID)  
);
```

```
CREATE TABLE Transaction (  
    TransactionID  VARCHAR(20),  
    UserID         VARCHAR(20),  
    CategoryID    VARCHAR(20),  
    Amount        DECIMAL(10,2),  
    Date          DATE,  
    Time          TIME,  
    Location      VARCHAR(100),  
    PaymentMethod VARCHAR(50),  
    PRIMARY KEY (TransactionID),  
    FOREIGN KEY (UserID) REFERENCES User(UserID),  
    FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID)  
);
```

```
CREATE TABLE Profile (  
    UserID         VARCHAR(20),  
    AvgMonthlySpend DECIMAL(10,2),  
    CategoryLimits VARCHAR(500),  
    TimeWindows   VARCHAR(200),  
    FrequentLocations VARCHAR(300),  
    PRIMARY KEY (UserID),  
    FOREIGN KEY (UserID) REFERENCES User(UserID)  
);
```

```
CREATE TABLE AnomalyRecord (  
    AnomalyID      VARCHAR(20),  
    TransactionID  VARCHAR(20),  
    AnomalyType    VARCHAR(50),  
    DetectedDate   DATE,  
    Status         VARCHAR(50),  
    PRIMARY KEY (AnomalyID),  
    FOREIGN KEY (TransactionID) REFERENCES Transaction(TransactionID)  
);
```

```
CREATE TABLE Alert (  
    AlertID        VARCHAR(20),  
    AnomalyID      VARCHAR(20),  
    UserID         VARCHAR(20),  
    Timestamp      DATETIME,  
    PRIMARY KEY (AlertID),  
    FOREIGN KEY (AnomalyID) REFERENCES AnomalyRecord(AnomalyID),  
    FOREIGN KEY (UserID) REFERENCES User(UserID)  
);
```


6. General Hierarchical Structure

The system follows a two-tier hierarchical flow:

1. Authentication Layer

- Users log in (Figure 7.1) or register (Figure 7.2), with credentials securely stored in the User table (Section 4.1).

2. Functional Layer

- Dashboard (Figure 7.3): Displays transactions, security status, and alerts.
- Transaction Management (Figure 7.5): Users add transactions, which are checked against Profile rules (e.g., spending limits).
- Alerts (Figure 7.4): Triggered by anomalies (Section 3.2) and stored in the AnomalyRecord and Alert tables.
- Profile Settings (Figure 7.6): Customizable fraud detection thresholds (e.g., time windows, locations).

Backend: PHP processes data via PDO (Section 8.2), while the frontend uses AJAX for real-time updates. The relational database (Section 4) ensures data integrity.

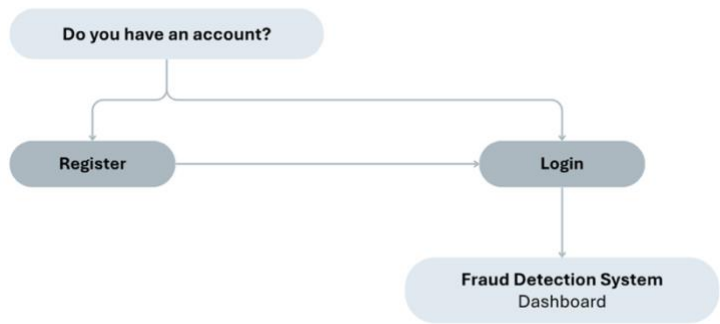


Figure 6.1. Account Access Flow

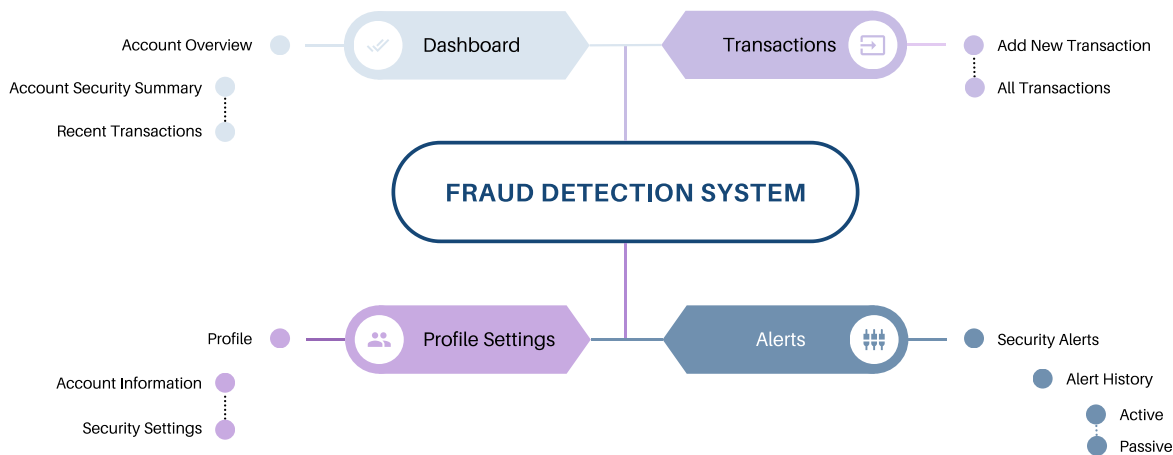
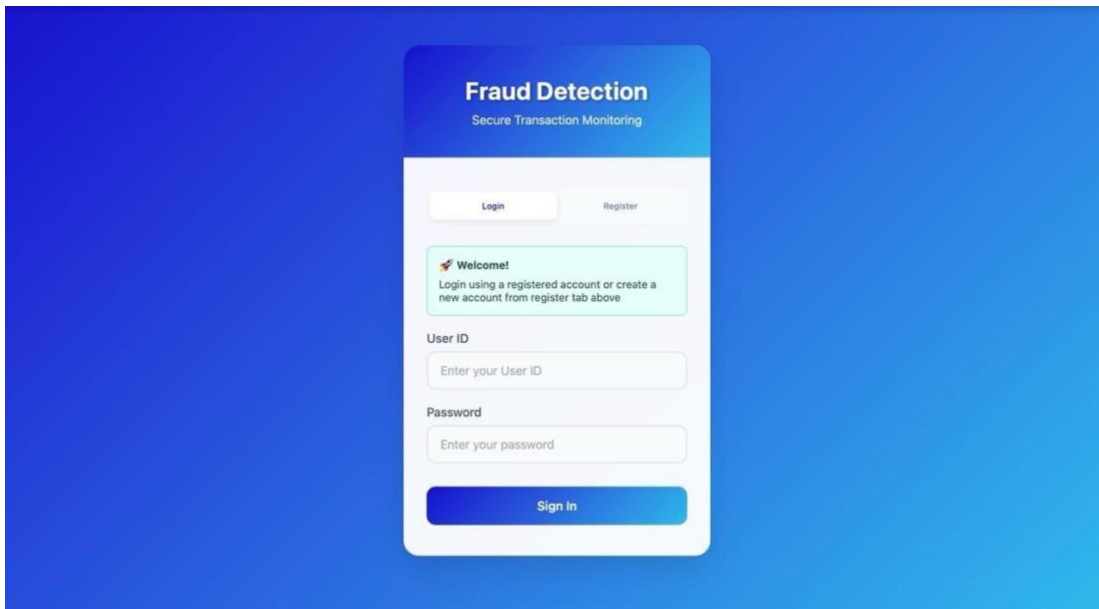


Figure 6.2. Functional Interface Structure

7. Professional User Manual

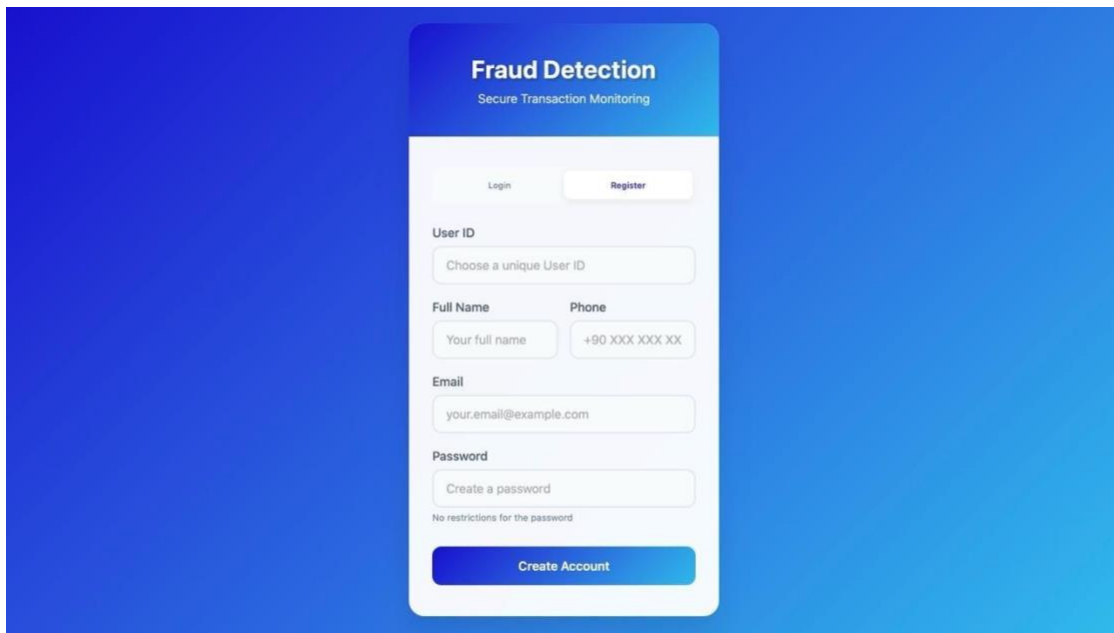
Enter your User ID and password to log in to the system.



The login page features a blue gradient background. At the top, a white box contains the text "Fraud Detection" and "Secure Transaction Monitoring". Below this, there are two tabs: "Login" (selected) and "Register". A green box with a checkmark icon and the text "Welcome!" is displayed, followed by the instruction "Login using a registered account or create a new account from register tab above". The login form includes a "User ID" field with the placeholder "Enter your User ID", a "Password" field with the placeholder "Enter your password", and a blue "Sign In" button.

Figure 7.1. Login page of the website

If you don't have an account, fill in your details to create a new account.



The registration page features a blue gradient background. At the top, a white box contains the text "Fraud Detection" and "Secure Transaction Monitoring". Below this, there are two tabs: "Login" and "Register" (selected). The registration form includes a "User ID" field with the placeholder "Choose a unique User ID", a "Full Name" field with the placeholder "Your full name", a "Phone" field with the placeholder "+90 XXX XXX XX", an "Email" field with the placeholder "your.email@example.com", and a "Password" field with the placeholder "Create a password". A note below the password field states "No restrictions for the password". A blue "Create Account" button is at the bottom.

Figure 7.2. Registration page of the website

View your transaction history and account security summary.

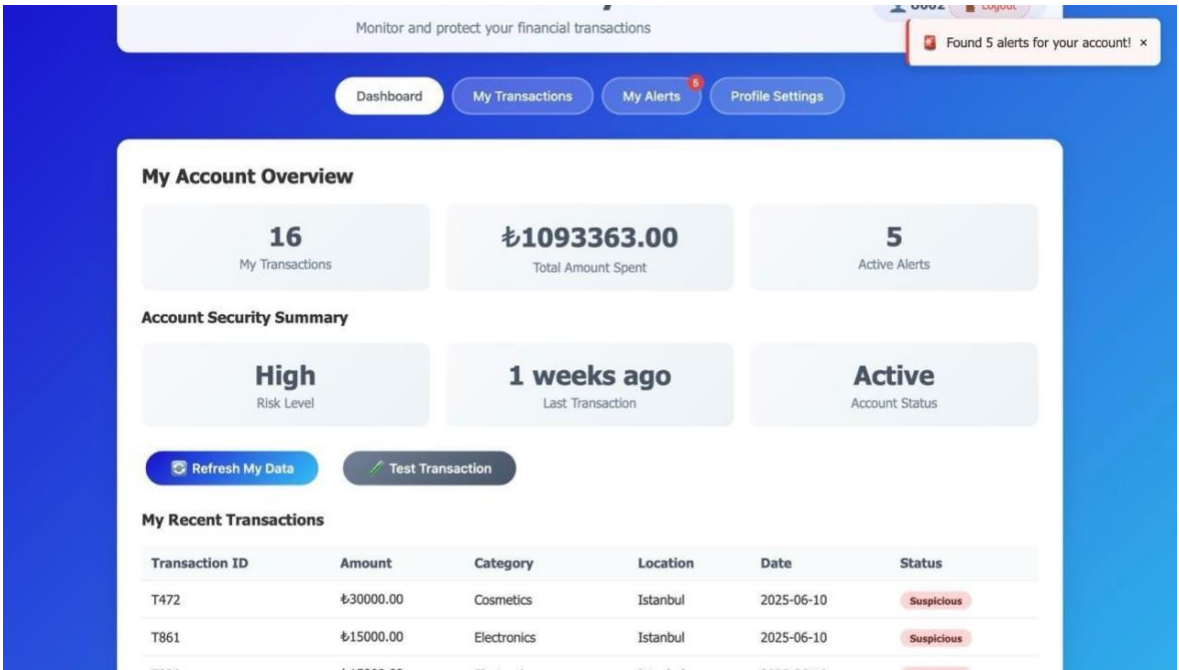


Figure 7.3. Dashboard Tab

Check active alerts for suspicious activities.

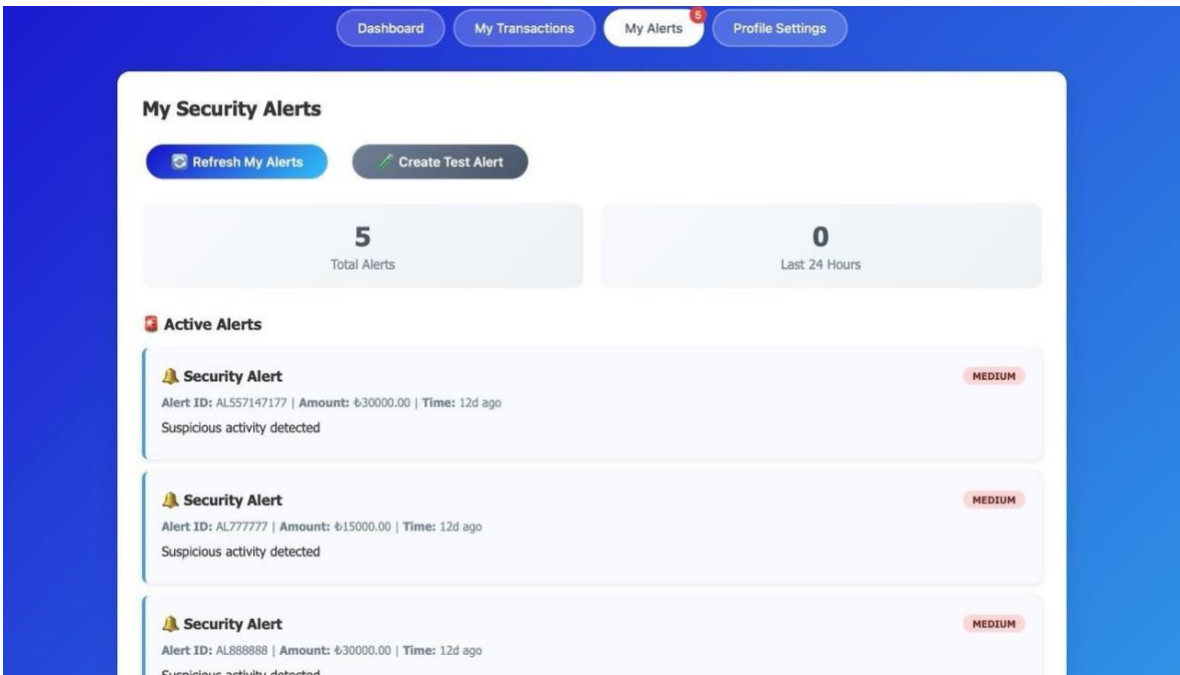


Figure 7.4. My Alerts section overview

Add new transactions or review past transactions.

Transaction ID	Amount	Category	Location	Date	Status
T472	₺30000.00	Cosmetics	Istanbul	2025-06-10	Suspicious
T861	₺15000.00	Electronics	Istanbul	2025-06-10	Suspicious
T820	₺15000.00	Electronics	Istanbul	2025-06-10	Suspicious
T370	₺15000.00	Food & Dining	İzmir	2025-06-09	Suspicious
T851	₺30000.00	Food & Dining	Istanbul	2025-06-09	Suspicious
T475	₺3000.00	Food & Dining	İzmir	2025-06-09	Normal

Figure 7.5. My Transactions tab

Customize your security preferences, such as spending limits and frequent locations.

My Profile & Security Settings

Account Information

User ID: U002
Total Transactions: 16
Account Status: Active

Security Settings

Average Monthly Spend: 0.00
Usual Transaction Times: 09:00-22:00

Category Spending Limits:

- Food & Dining: Set your limit
- Electronics: Set your limit
- Cosmetics: Set your limit
- Bills: Set your limit
- Entertainment: Set your limit

Frequent Locations:

Istanbul, Ankara

Update Security Settings Reset

Figure 7.6. Profile Settings Tab

8. SQL Database and Coding

8.1. SQL Database and Tables

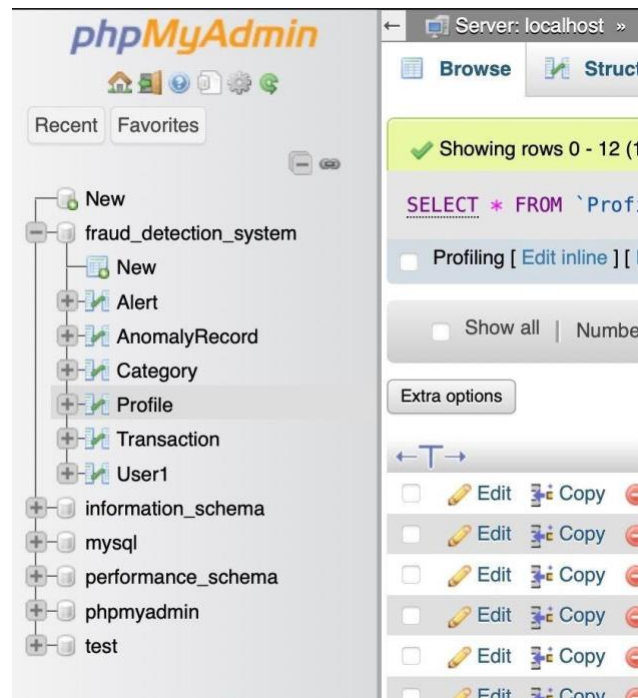


Figure 8.1. Fraud Detection System Database and Tables

The screenshot shows the phpMyAdmin interface with the 'Transaction' table selected. The table structure is visible on the left, and the main area displays the table data. The table has columns: TransactionID, UserID, CategoryID, Amount, Date, Time, Location, and PaymentMethod. The data is displayed in a grid format with edit and delete icons for each row.

TransactionID	UserID	CategoryID	Amount	Date	Time	Location	PaymentMethod
T001	U001	C001	45.75	2025-05-01	12:30:00	Istanbul	Credit Card
T002	U001	C002	299.99	2025-05-05	15:00:00	Istanbul	Debit Card
T003	U001	C003	80.00	2025-05-10	18:45:00	Istanbul	Cash
T004	U002	C001	30.50	2025-05-03	13:10:00	Istanbul	Credit Card
T005	U002	C002	150.00	2025-05-07	16:00:00	Istanbul	Credit Card
T006	U002	C003	55.00	2025-05-12	19:30:00	Istanbul	Cash
T007	U003	C001	22.20	2025-05-02	11:20:00	Istanbul	Debit Card
T008	U003	C002	420.00	2025-05-06	14:50:00	Istanbul	Credit Card
T009	U003	C003	90.00	2025-05-09	17:00:00	Istanbul	Cash
T200	U001	C001	100.00	2025-06-08	20:24:47	Istanbul	Credit Card
T206	U6725	C003	500.00	2025-05-30	15:10:10	Istanbul	Credit Card
T234	U2496	C001	200.00	2025-06-10	15:18:17	Ankara	Credit Card
T293	U2496	C002	4000.00	2025-06-10	15:18:03	Istanbul	Bank Transfer
T304	U2496	C001	372.00	2025-06-14	18:24:45	Istanbul	Credit Card
T335	U005	C001	500.00	2025-05-28	15:50:23	Istanbul	Credit Card
T369	U7867	C003	20000.00	2025-06-09	11:31:05	Bursa	Bank Transfer
T370	U002	C001	15000.00	2025-06-09	11:42:19	Izmir	Credit Card
T378	U2496	C005	20000.00	2025-06-10	15:19:44	Istanbul	Debit Card
T398	U6725	C002	800000.00	2025-05-30	15:13:00	Istanbul	Credit Card
T427	U002	C002	15000.00	2025-06-09	10:53:32	Istanbul	Credit Card
T432	U2496	C001	209.00	2025-06-14	18:24:38	Istanbul	Credit Card
T463	U6725	C004	200.00	2025-05-30	15:11:10	Istanbul	Credit Card
T472	U002	C003	30000.00	2025-06-10	15:05:47	Istanbul	Bank Transfer
T475	U002	C001	3000.00	2025-06-09	11:34:15	Izmir	Bank Transfer
T523	U6725	C001	1000.00	2025-05-30	15:09:44	Istanbul	Credit Card

Figure 8.2. 'Transaction' Table

The screenshot shows the phpMyAdmin interface for the 'fraud_detection_system' database. The 'Category' table is selected, and its structure and data are displayed. The table has two columns: 'CategoryID' and 'CategoryName'.

CategoryID	CategoryName
C001	Food & Dining
C002	Electronics
C003	Cosmetics
C004	Bills
C005	Entertainment

Figure 8.3. 'Category' Table

The screenshot shows the phpMyAdmin interface for the 'fraud_detection_system' database. The 'Profile' table is selected, and its structure and data are displayed. The table has five columns: 'UserID', 'AvgMonthlySpend', 'CategoryLimits', 'TimeWindows', and 'FrequentLocations'.

UserID	AvgMonthlySpend	CategoryLimits	TimeWindows	FrequentLocations
U001	40000.00	("C001":2000,"C002":4000,"C003":500,"C004":20000,"...	09:00-23:00	Istanbul,Ankara
U002	20000.00	("C001":3000,"C002":4000,"C003":5000,"C004":5000,"...	09:00-21:00	Istanbul
U003	50000.00	("C001":3000,"C002":4000,"C003":5000,"C004":5000,"...	10:00-22:00	Istanbul
U004	50000.00	("C001":3000,"C002":4000,"C003":5000,"C004":5000,"...	09:00-22:00	Istanbul,Ankara
U005	60000.00	("C001":5000,"C002":6000,"C003":10000,"C004":4000,"...	10:00-22:00	Istanbul,Ankara
U006	50000.00	("C001":5000,"C002":10000,"C003":2000,"C004":3000,"...	10:00-23:00	Istanbul,Ankara
U007	40000.00	("C001":4000,"C002":5000,"C003":2000,"C004":90000,"...	10:00-22:00	Istanbul,Ankara
U008	50000.00	("C001":3000,"C002":4000,"C003":5000,"C004":6000,"...	10:00-22:00	Istanbul,Ankara
U009	20000.00	("C001":4000,"C002":5000,"C003":6000,"C004":10000,"...	10:00-22:00	Istanbul,Ankara
U2496	40000.00	("C001":5000,"C002":2000,"C003":5000,"C004":20000,"...	09:00-22:00	Istanbul,Izmir
U3218	30000.00	("C001":4000,"C002":5000,"C003":2000,"C004":7000,"...	10:00-22:00	Istanbul,Ankara
U6725	40000.00	("C001":4000,"C002":5000,"C003":1000,"C004":3000,"...	10:00-22:00	Istanbul,Ankara
U7867	40000.00	("C001":2000,"C002":3000,"C003":5000,"C004":10000,"...	09:00-23:00	Istanbul,Ankara

Figure 8.4. 'Profile' Table

8.2. SQL Connection

A database named "fraud_detection_system" and six tables as "User", "Transaction", "Category", "Profile", "AnomalyRecord" and "Alert" were created in MySQL. The SQL Database and website connection is established through a comprehensive PHP configuration system that ensures secure and reliable communication between the web application and the MySQL database. The connection logic is implemented using a DatabaseConfig class that handles all database interactions through PHP's PDO (PHP Data Objects) extension, which provides a consistent interface for accessing databases and effectively protects against SQL injection attacks through prepared statements. The database connection is configured with specific parameters including the localhost server, the fraud_detection_system database name, and default XAMPP credentials (root user with empty password). The connection string uses a Data Source Name (DSN) format that specifies the MySQL host, port 3306, database name, and UTF-8 character encoding to ensure proper handling of international characters. PDO connection options are carefully configured to enable exception mode for error handling, set the default fetch mode to associative arrays, disable prepared statement emulation for better security, and initialize the connection with proper UTF-8 encoding.

The system implements a robust error handling mechanism that logs database connection failures without exposing sensitive information to end users, maintaining security while providing diagnostic capabilities for developers. A connection testing function validates the database setup by checking for the existence of all required tables and providing detailed feedback about the database structure. The DatabaseConfig class includes a helper function that can be easily imported into other PHP files, creating a centralized and consistent approach to database access throughout the application.

PHP handles all server-side logic including user authentication through secure password hashing, transaction processing with real-time fraud detection, session management for maintaining user states, and alert generation based on anomaly detection algorithms. The backend communicates with the frontend through a RESTful API architecture that processes JSON requests and responses, enabling seamless data exchange between the user interface and the database. All database operations use prepared statements with parameterized queries, ensuring that user input is properly sanitized and preventing SQL injection vulnerabilities. The modular design allows for easy maintenance and scalability, with the database connection logic separated from business logic, making it straightforward to modify database configurations or extend functionality without affecting other system components.

8.3. Coding Logic

The development language used for backend development was PHP, which handled server-side logic such as user authentication, transaction management, fraud analysis, and alert generation. PHP's compatibility with MySQL and its wide adoption made it an ideal choice for this project. All communication between the frontend and backend was managed through a RESTful API built with PHP.

```

<?php
// Fraud detection algorithm example
public function analyzeTransaction($transactionData) {
    $riskScore = 0;
    $flags = [];
    $alertCreated = false;

    $amount = floatval($transactionData['amount']);
    $userID = $transactionData['transactionUserID'];
    $transactionID = $transactionData['transactionID'];

    // Check 1: High amount transactions
    if ($amount >= 15000) {
        $riskScore += 60;
        $flags[] = "Very high amount transaction (฿" . number_format($amount, 2) . ")";
        $isSuspicious = true;
    } elseif ($amount >= 10000) {
        $riskScore += 40;
        $flags[] = "High amount transaction (฿" . number_format($amount, 2) . ")";
        $isSuspicious = true;
    }

    // Check 2: Get user profile and check against limits
    $userProfile = $this->getUserProfile($userID);
    if ($userProfile && !empty($userProfile['CategoryLimits'])) {
        $categoryLimits = json_decode($userProfile['CategoryLimits'], true);
        $categoryID = $transactionData['categoryID'] ?? 'C001';

        if ($categoryLimits && isset($categoryLimits[$categoryID])) {
            $limit = floatval($categoryLimits[$categoryID]);
            if ($amount > $limit) {
                $riskScore += 30;
                $flags[] = "Amount exceeds category limit";
                $isSuspicious = true;
            }
        }
    }

    return [
        'risk_score' => $riskScore,
        'risk_level' => $this->calculateRiskLevel($riskScore),
        'flags' => $flags,
        'alert_created' => $alertCreated
    ];
}
?>

```

In the above example, the fraud detection system analyzes each transaction by checking multiple parameters. The system assigns risk scores based on transaction amounts, with transactions over ฿15,000 receiving 60 points and those over ฿10,000 receiving 40 points. The code also retrieves the user's profile to check against personalized spending limits stored in JSON format. If any suspicious patterns are detected, the system creates appropriate flags and calculates an overall risk level for the transaction.