CIPHERCASCADE: A MULTI-ENCRYPTION FRAMEWORK

PHASE-I PROJECT REPORT

SUBMITTED BY

MAYANK ANAND [RA2411030010067]
SANSKAR GUPTA [RA2411030010068]
NAMAN TYAGI [RA2411030010073]

UNDER THE GUIDANCE OF
DR. MANICKAM M

21CSC203P – ADVANCED PROGRAMMING PRACTICE

DEPARTMENT OF NETWORKING AND COMMUNICATION

FACULTY OF ENGINEERING AND TECHNOLOGY
SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR
OCTOBER 2025

MARK RUBRICS

ABSTRACT ,INTRODUCTION (1)

USE CASE DIAGRAM(2)

ARCHITECTURE DIAGRAM(2)

COMPLETE PROJECT EXECUTION STEP (5)

TOTAL (10)

ABSTRACT

CipherCascade is a Java-based desktop application that implements a multi-layer encryption framework supporting AES, DES, and Blowfish algorithms in cascade mode. The system features a comprehensive user authentication mechanism integrated with MySQL database through JDBC, allowing users to securely encrypt sensitive data using single or multiple encryption algorithms sequentially. The application provides an intuitive graphical user interface built with Java Swing, enabling users to register, login, encrypt/decrypt data, and store encrypted records in a personal database. The cascade encryption approach enhances security by applying multiple encryption layers, making it significantly harder for unauthorized parties to decrypt sensitive information. Each layer of encryption adds an additional computational barrier, exponentially increasing the complexity for brute-force attacks. The encrypted output is encoded using Base64 encoding scheme to ensure safe storage and transmission as text format. This project demonstrates practical implementation of cryptographic algorithms, database connectivity, secure software design principles, and the importance of layered security in modern information systems.

1. INTRODUCTION / BACKGROUND

In today's digital age, data security has become paramount as sensitive information is constantly transmitted and stored electronically. Most existing encryption systems rely on a single encryption algorithm, which creates a single point of failure in security architecture. If that one algorithm is compromised through cryptanalysis, advanced computing techniques, or quantum computing threats, the entire data becomes vulnerable. Multi-layer or cascade encryption provides enhanced security by applying multiple encryption algorithms sequentially, creating multiple independent barriers for potential attackers. Each encryption layer uses different mathematical foundations, key structures, and cipher mechanisms, ensuring that compromise of one algorithm does not expose the entire data. This defense-in-depth approach aligns with modern cybersecurity principles where multiple independent security controls are employed. Additionally, the encoded output must be represented in a text-safe format for database storage and network transmission, which is achieved through Base64 encoding. The integration of database systems for secure credential storage and encrypted data management further strengthens the security architecture. This project addresses the need for a user-friendly encryption tool that combines the strengths of multiple encryption standards while providing secure user authentication and persistent storage of encrypted data through robust database connectivity.

2. PROBLEM STATEMENT

Most current encryption systems implement only a single layer of encryption, creating a vulnerability where breaking one algorithm compromises the entire security. There is a need for an accessible multi-layer encryption system that applies multiple algorithms in cascade to enhance security, while providing user-friendly interface, proper authentication, and database integration for managing encrypted records.

3. OBJECTIVES

- To develop a desktop application implementing cascade encryption using AES, DES, and Blowfish algorithms
- To overcome the limitation of single-layer encryption by providing multi-algorithm cascade functionality
- To create a secure user authentication system with registration and login functionality
- To integrate MySQL database using JDBC for persistent storage of user credentials and encrypted data
- To implement Base64 encoding for safe text representation of encrypted binary data
- To design an intuitive graphical user interface using Java Swing for ease of use
- To implement proper encryption key management and secure data handling practices
- To provide users with the ability to view, manage, and delete their encrypted records

4. SCOPE & LIMITATIONS

Scope:

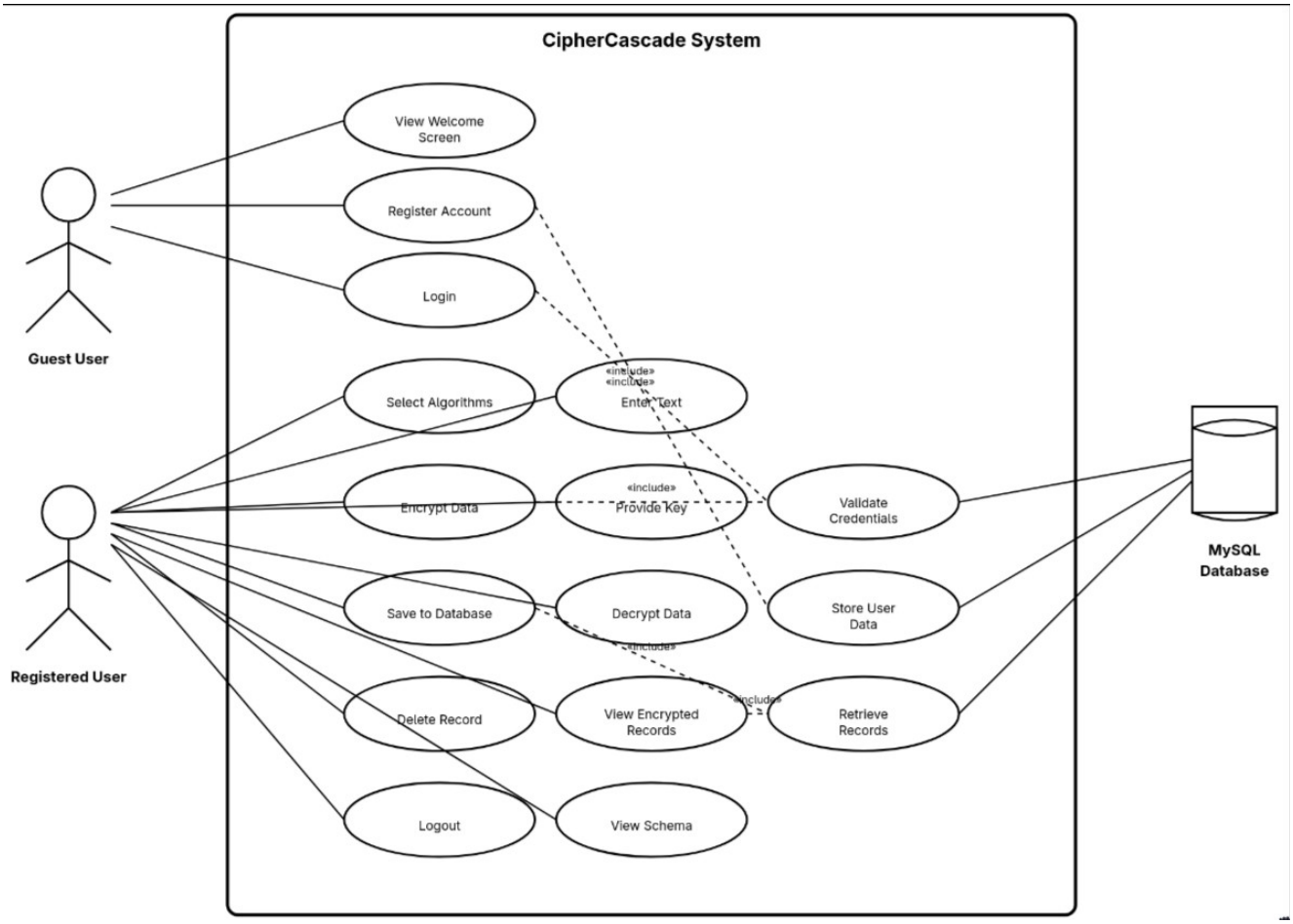
- Desktop application compatible with Windows, Linux, and macOS
- Support for three encryption algorithms: AES (128-bit), DES (56-bit), and Blowfish
- Multi-layer cascade encryption with user-selectable algorithm combinations

- User authentication with username/password protection
- Personal encrypted data storage with timestamp tracking
- Base64 encoding for encrypted data representation
- Database schema visualization and data management interface

Limitations:

- Passwords stored in plain text in database (not hashed - for educational purposes)
- Fixed encryption modes (ECB mode) without IV (Initialization Vector)
- No password recovery mechanism implemented
- Single-user desktop application (not a multi-user server application)

5. USE CASE DIAGRAM



5. TECHNOLOGY STACK

Programming Language & Framework:

- Java SE 11 (or higher)
- Java Swing for GUI development
- Java Cryptography Extension (JCE) for encryption algorithms
- Java Base64 utility for encoding/decoding

Database:

- MySQL Database Server (compatible with MariaDB)
- JDBC (Java Database Connectivity) for database integration
- MySQL Connector/J driver (version 8.x)

Development Tools:

- IDE: IntelliJ IDEA / Eclipse / NetBeans
- Database Management: phpMyAdmin / MySQL Workbench / DBeaver
- Build Tool: Manual compilation or Maven

External Libraries:

- javax.crypto - for encryption/decryption operations
- java.sql - for database connectivity
- javax.swing - for GUI components
- java.util.Base64 - for Base64 encoding/decoding

6. SYSTEM ARCHITECTURE

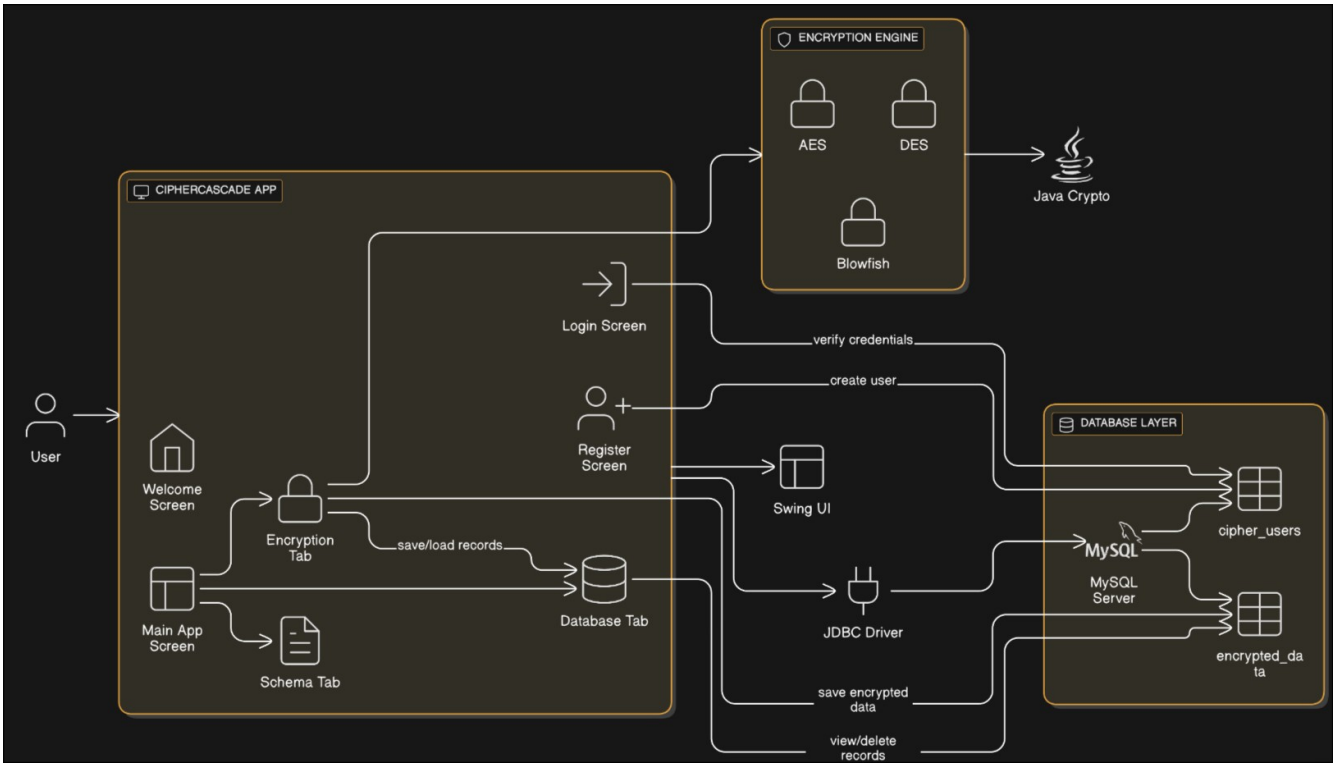
The CipherCascade application follows a layered architecture pattern with three primary layers implementing the Model-View-Controller (MVC) design philosophy:

Presentation Layer: Java Swing-based GUI with CardLayout for screen navigation (Welcome, Login, Register, Main Application screens). Provides user interaction through text areas, buttons, checkboxes, and tables. This layer handles all user input validation and displays appropriate feedback messages.

Business Logic Layer: Implements encryption/decryption algorithms (AES, DES, Blowfish) with cascade functionality where data passes through multiple encryption stages sequentially. Each algorithm encrypts the output of the previous one, creating layers of protection. Handles user authentication logic, session management, key padding mechanisms for fixed-length cipher requirements, and Base64 encoding/decoding to convert binary encrypted data into text format suitable for database storage and display.

Data Layer: MySQL database integration using JDBC API for persistent storage and retrieval operations. Manages two main tables: cipher_users (authentication) and encrypted_data (encrypted records with foreign key relationships). Implements prepared statements to prevent SQL injection attacks and ensure data integrity.

The architecture ensures separation of concerns, making the application maintainable, scalable, and secure. JDBC acts as the bridge between Java application logic and the relational database, providing platform-independent database operations. Base64 encoding is applied after encryption to ensure the binary cipher output can be safely stored as TEXT in the database and displayed in the GUI without character encoding issues.



7. DATA MODEL / KEY TABLES / INPUTS & OUTPUTS

Database Name: cipherdb

Table 1: cipher_users

Column Name	Data Type	Constraints	Description
id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique user identifier
username	VARCHAR(100)	UNIQUE, NOT NULL	Login username
password	VARCHAR(100)	NOT NULL	User password
email	VARCHAR(100)	UNIQUE, NOT NULL	User email address
full_name	VARCHAR(100)	-	User's full name
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Registration timestamp

Table 2: encrypted_data

Column Name	Data Type	Constraints	Description
id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique record identifier
user_id	INT	FOREIGN KEY → cipher_users(id)	Owner of encrypted data
original_text	TEXT	-	Plain text before encryption
encrypted_text	TEXT	-	Base64 encoded encrypted output

Column Name	Data Type	Constraints	Description
algorithms_used	VARCHAR(255)	-	Algorithms applied (e.g., "AES DES")
encryption_date	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Encryption timestamp

Entity Relationship:

- One user (cipher_users) can have many encrypted records (encrypted_data)
- Relationship: 1:N (One-to-Many)
- Foreign Key: encrypted_data.user_id references cipher_users.id

Key Inputs:

- User registration: username, password, email, full name
- Login: username, password
- Encryption: plain text, encryption key, selected algorithms (single or multiple)
- Decryption: Base64 encoded encrypted text, encryption key, selected algorithms in same order

Key Outputs:

- Encrypted text (Base64 encoded string for safe text representation)
- Decrypted text (original plain text recovered from cascade)
- Database records with timestamps
- Success/error messages

Encryption Flow:

- Plain Text → Algorithm 1 → Binary Data → Algorithm 2 → Binary Data → ... → Final Binary → Base64 Encoding → Stored Text

Decryption Flow:

- Base64 Text → Base64 Decoding → Binary Data → Reverse Algorithm N → ... → Reverse Algorithm 1 → Plain Text

STEP-BY-STEP PROCEDURE

1. Database Setup and Configuration

1.1 Database Prerequisites

- MySQL/MariaDB server must be installed and running
- Default connection parameters:
    - Host: localhost
    - Port: 3306
    - Database name: cipherdb
    - Username: root
    - Password: (empty)

1.2 Creating the Database

Execute the following SQL command in MySQL console or workbench:

```
CREATE DATABASE cipherdb;

USE cipherdb;
```

1.3 Database Schema

The application automatically creates two tables:

Table 1: cipher_users

Stores user authentication information.

```
CREATE TABLE IF NOT EXISTS cipher_users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    full_name VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Columns Explanation:

- `id`: Auto-incrementing primary key for unique user identification

- `username`: Unique username for login (cannot be duplicated)
- `password`: User password (stored in plain text - note: in production, this should be hashed)
- `email`: Unique email address
- `full_name`: User's full display name
- `created_at`: Timestamp of account creation

Table 2: encrypted_data

Stores encrypted text records associated with users.

```
CREATE TABLE IF NOT EXISTS encrypted_data (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    original_text TEXT,
    encrypted_text TEXT,
    algorithms_used VARCHAR(255),
    encryption_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES cipher_users(id)
);
```

Columns Explanation:

- `id`: Auto-incrementing primary key for each encryption record
- `user_id`: Foreign key linking to cipher_users table
- `original_text`: Original unencrypted text (stored for reference)
- `encrypted_text`: Encrypted output after applying selected algorithms
- `algorithms_used`: Space-separated list of algorithms applied (e.g., "AES DES")
- `encryption_date`: Timestamp when encryption was performed

```
[kiri@archlinux ~]$ mysql -u root
mysql: Deprecated program name. It will be removed in a future release, use '/usr/bin/mariadb' instead
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 14
Server version: 12.0.2-MariaDB Arch Linux

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> USE cipherdb;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [cipherdb]> SHOW TABLES;
+--------------------+
| Tables_in_cipherdb |
+--------------------+
| cipher_users       |
| encrypted_data     |
+--------------------+
2 rows in set (0.001 sec)

MariaDB [cipherdb]> DESC cipher_users;
+------------+--------------+------+-----+---------------------+----------------+
| Field      | Type         | Null | Key | Default             | Extra          |
+------------+--------------+------+-----+---------------------+----------------+
| id         | int(11)      | NO   | PRI | NULL                | auto_increment |
| username   | varchar(100) | NO   | UNI | NULL                |                |
| password   | varchar(100) | NO   |     | NULL                |                |
| email      | varchar(100) | NO   | UNI | NULL                |                |
| full_name  | varchar(100) | YES  |     | NULL                |                |
| created_at | timestamp    | YES  |     | current_timestamp() |                |
+------------+--------------+------+-----+---------------------+----------------+
6 rows in set (0.001 sec)

MariaDB [cipherdb]> DESC encrypted_data;
+-----------------+--------------+------+-----+---------------------+----------------+
| Field           | Type         | Null | Key | Default             | Extra          |
+-----------------+--------------+------+-----+---------------------+----------------+
| id              | int(11)      | NO   | PRI | NULL                | auto_increment |
| user_id         | int(11)      | YES  | MUL | NULL                |                |
| original_text   | text         | YES  |     | NULL                |                |
| encrypted_text  | text         | YES  |     | NULL                |                |
| algorithms_used | varchar(255) | YES  |     | NULL                |                |
| encryption_date | timestamp    | YES  |     | current_timestamp() |                |
+-----------------+--------------+------+-----+---------------------+----------------+
6 rows in set (0.001 sec)
```

1.4 Database Connectivity Setup

The application uses JDBC (Java Database Connectivity) to connect to MySQL:

```
private static final String DB_URL = "jdbc:mysql://localhost:3306/cipherdb";
private static final String DB_USER = "root";
private static final String DB_PASS = "";

// Loading MySQL driver and establishing connection
Class.forName("com.mysql.cj.jdbc.Driver");
conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASS);
```

Required JDBC Driver:

- MySQL Connector/J (mysql-connector-java-8.x.x.jar)
- Must be added to project classpath/build path

Connection Flow:

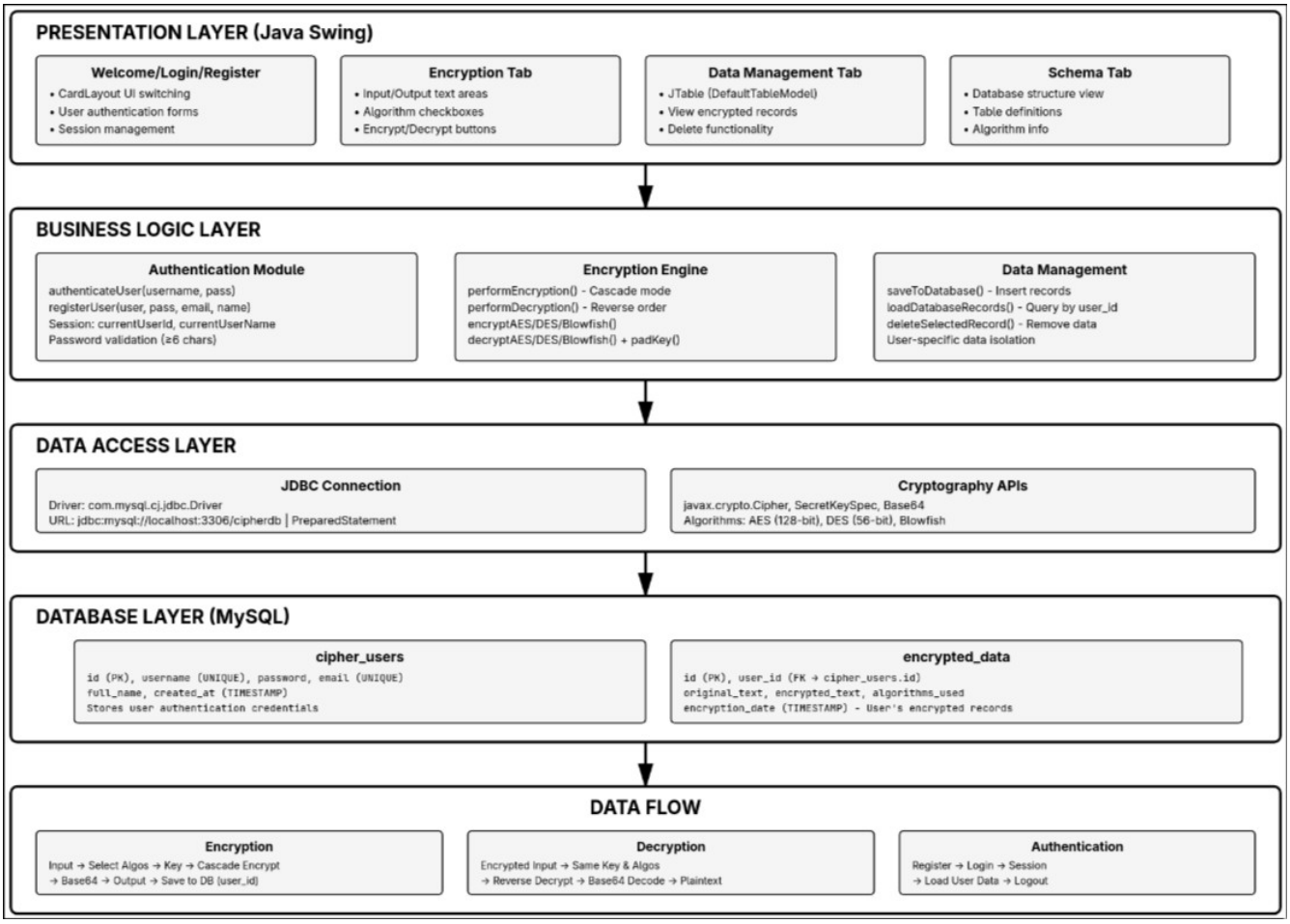
1. Load JDBC driver class
2. Establish connection using URL, username, password
3. Create Statement/PreparedStatement objects for SQL execution
4. Execute queries and process results
5. Close connections properly

2. Front-End Architecture

2.1 Application Flow

The application uses a CardLayout to manage multiple screens:

1. **Welcome Screen** → Login or Register
2. **Login Screen** → Authenticate existing user
3. **Register Screen** → Create new account
4. **Main Application Screen** → Three tabs:
    - Encryption/Decryption operations
    - My Encrypted Data (database view)
    - Database Schema documentation



3. Screen-by-Screen Walkthrough

3.1 Welcome Screen

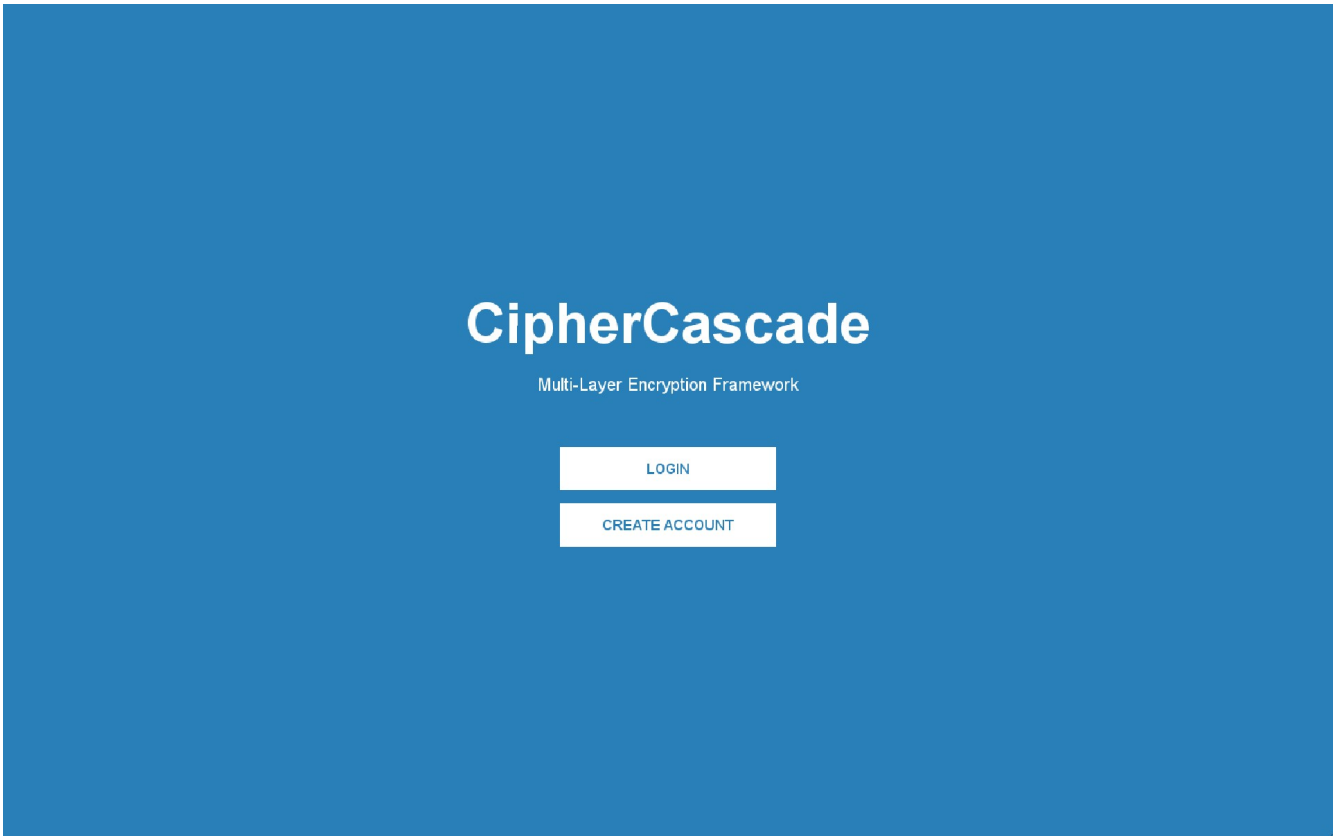
Purpose: Initial landing page with branding and navigation options.

Components:

- Title: "CipherCascade" in large bold font (64pt)
- Subtitle: "Multi-Layer Encryption Framework" (20pt)
- Two action buttons:
    - LOGIN: Directs to login screen
    - CREATE ACCOUNT: Directs to registration screen

User Actions:

- Click LOGIN → Navigate to Login Screen
- Click CREATE ACCOUNT → Navigate to Register Screen



3.2 Registration Screen

Purpose: New user account creation.

Form Fields:

1. **Full Name** (JTextField): User's display name

2. **Username** (JTextField): Unique login identifier
3. **Email** (JTextField): Unique email address
4. **Password** (JPasswordField): Secure password input
5. **Confirm Password** (JPasswordField): Password verification

Validation Rules:

- All fields must be filled
- Passwords must match
- Password must be at least 6 characters
- Username and email must be unique (database constraint)

Action Buttons:

- CREATE ACCOUNT: Submits registration
- BACK: Returns to welcome screen

Registration Process:

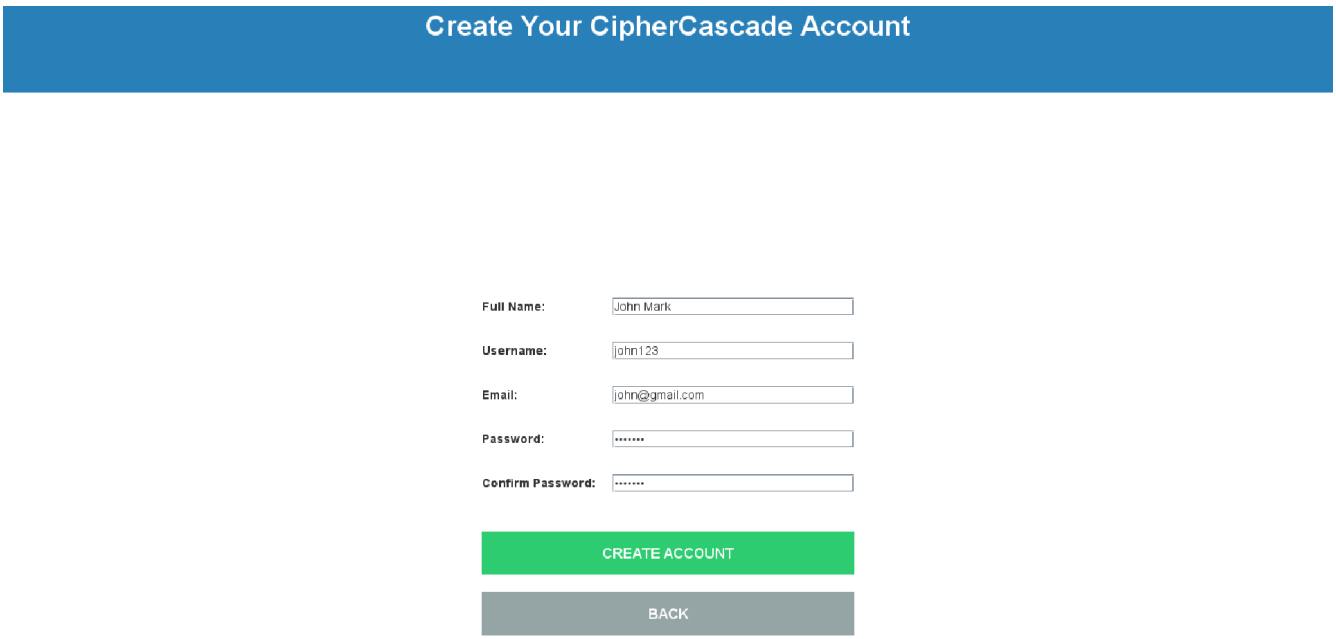
1. User fills all form fields
2. Click CREATE ACCOUNT button
3. Application validates input:
    - Checks for empty fields
    - Verifies password match
    - Validates password length (≥6 characters)
4. If valid, executes SQL INSERT:

```
INSERT INTO cipher_users (username, password, email, full_name) VALUES (?, ?, ?, ?)
```

5. Success: Shows confirmation dialog, redirects to login
6. Failure: Shows error (duplicate username/email)

Error Handling:

- Empty fields → "Please fill all fields!"
- Password mismatch → "Passwords do not match!"
- Short password → "Password must be at least 6 characters!"
- Duplicate entry → "Username or email may already exist"



3.3 Login Screen

Purpose: User authentication for existing accounts.

Form Fields:

1. **Username** (JTextField): Account username
2. **Password** (JPasswordField): Account password

Action Buttons:

- LOGIN: Authenticate and enter application
- BACK: Return to welcome screen

Authentication Process:

1. User enters credentials
2. Click LOGIN button
3. Application validates input (non-empty fields)
4. Executes SQL query:

```
SELECT id, full_name FROM cipher_users WHERE username = ? AND password = ?
```

5. If record found:
    - Store `currentUserId` and `currentUserName` in session
    - Show welcome message
    - Navigate to main application screen

6. If not found:
    - Show "Invalid credentials" error

Session Management:

- `currentUserId`: Stores logged-in user's database ID
- `currentUserName`: Stores user's full name for display
- These variables persist throughout the session

3.4 Main Application Screen - Encryption Tab

Purpose: Core encryption/decryption functionality.

Layout Sections:

Header Bar

- Application title: "CipherCascade Multi-Layer Encryption"
- User information: "User: [Full Name]"
- Logout button (red, right-aligned)

Input/Output Area (Split View)

Left Panel - Input Text:

- Large text area (JTextArea) for entering plain text or encrypted text
- Scrollable

- Line wrapping enabled
- Monospaced font

Right Panel - Output Text:

- Large text area for displaying results
- Read-only (non-editable)
- Shows encrypted output after encryption
- Shows decrypted output after decryption

Control Panel (Bottom Section)

Algorithm Selection:

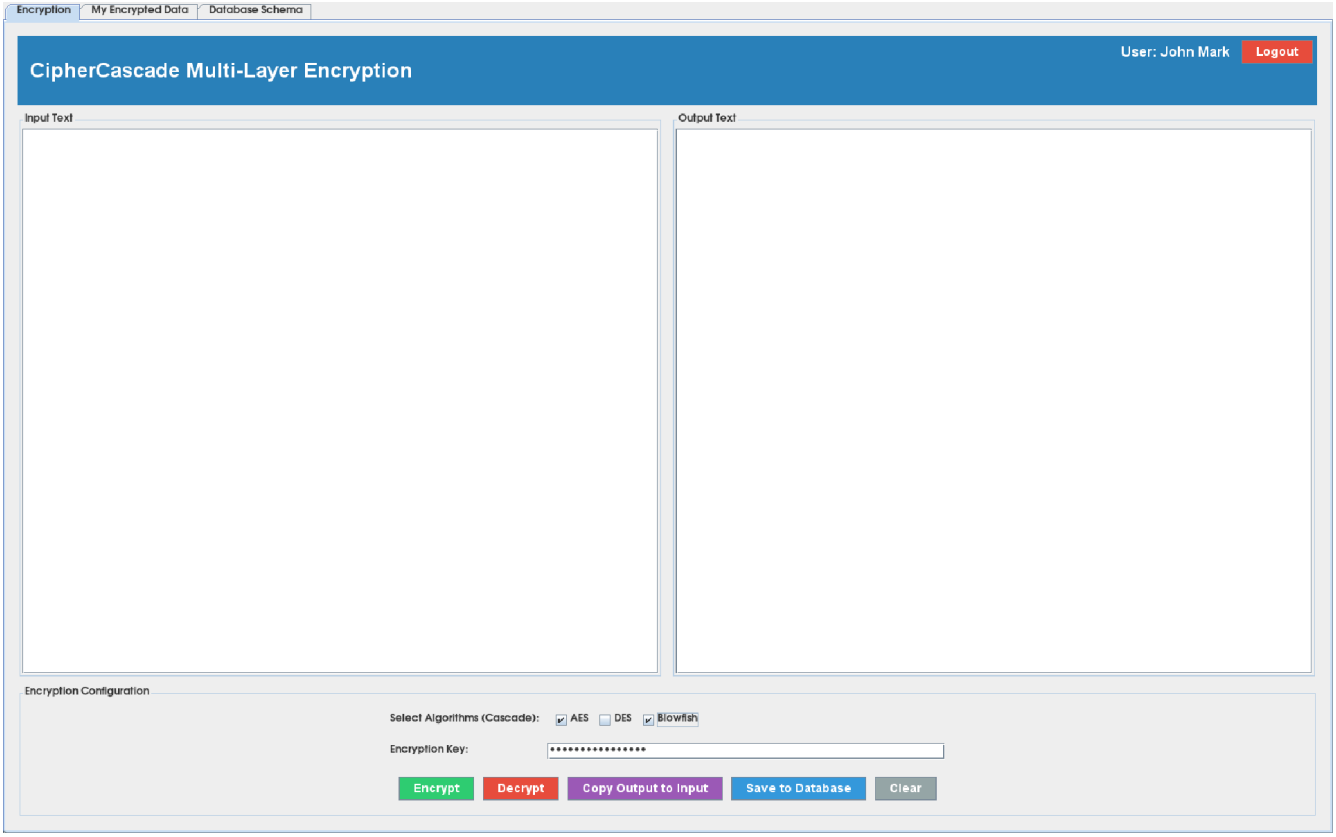
- Three checkboxes for cascade encryption:
  - AES (Advanced Encryption Standard - 128 bit)
  - DES (Data Encryption Standard - 56 bit)
  - Blowfish (Variable key length)
- Multiple algorithms can be selected simultaneously
- Encryption applies in order: AES → DES → Blowfish
- Decryption applies in reverse: Blowfish → DES → AES

Encryption Key:

- Password field (JPasswordField)
- Default value: "MySecretKey12345"
- Same key must be used for encryption and decryption
- Key is padded/truncated to match algorithm requirements

Action Buttons:

1. **ENCRYPT (Green)**: Encrypts input text
2. **DECRYPT (Red)**: Decrypts input text
3. **Copy Output to Input (Purple)**: Transfers output to input for decryption
4. **Save to Database (Blue)**: Saves current encryption to database
5. **CLEAR (Gray)**: Clears both input and output areas



3.5 Encryption Operation

Step-by-Step Process:

1. **User enters text** in Input area (left panel)
2. **Select algorithms** (one or more checkboxes)
3. **Enter encryption key** (or use default)
4. **Click ENCRYPT button**

Backend Process:

```
Original Text
      ↓
[If AES selected] → Apply AES Encryption
      ↓
[If DES selected] → Apply DES Encryption
      ↓
[If Blowfish selected] → Apply Blowfish Encryption
      ↓
Base64 Encoded Output → Display in Output area
```

Encryption Algorithm Details:

AES Encryption:

- Key size: 16 bytes (128 bits)
- Mode: Default ECB mode
- Key preparation: Pad or truncate key to 16 characters
- Output: Base64-encoded string

DES Encryption:

- Key size: 8 bytes (56 bits usable)
- Mode: Default ECB mode
- Key preparation: Pad or truncate key to 8 characters
- Output: Base64-encoded string

Blowfish Encryption:

- Key size: Variable (1-56 bytes)
- Mode: Default ECB mode
- Uses key as-is without padding
- Output: Base64-encoded string

Cascade Effect: When multiple algorithms are selected (e.g., AES + DES):

1. Original text is encrypted with AES → Result1
2. Result1 is encrypted with DES → Final output
3. Creates layered encryption (more secure)

Success Dialog: Shows confirmation with algorithms used and instructions for decryption.

3.6 Decryption Operation

Step-by-Step Process:

1. **Click "Copy Output to Input"** to move encrypted text to input area
    - OR manually paste encrypted text into Input area
2. **Select SAME algorithms** used for encryption (checkboxes)
3. **Enter SAME encryption key** used originally
4. **Click DECRYPT button**

Backend Process:

```
                    Encrypted Text (Base64)
                                ↓
[If Blowfish was used] → Apply Blowfish Decryption
                                ↓
     [If DES was used] → Apply DES Decryption
                                ↓
     [If AES was used] → Apply AES Decryption
```

↓

`Original Text → Display in Output area`

Important Notes:

- Decryption order is **reverse** of encryption order
- Must use exact same key
- Must select exact same algorithms
- Incorrect key/algorithms will produce garbage output or error

Error Handling:

- Wrong key → Decryption error with helpful message
- Wrong algorithms → Incorrect output
- Invalid encrypted text → Base64 decoding error

3.7 Save to Database Operation

Purpose: Persist encrypted data to MySQL database for future reference.

Prerequisites:

- User must be logged in
- Database connection must be active
- Input and Output areas must have content
- At least one algorithm must be selected

Save Process:

1. **Click "Save to Database" button**
2. Application validates:
    - User is authenticated (currentUserId != -1)
    - Input text is not empty
    - Output text is not empty
    - At least one algorithm was selected
3. **Prepares data:**
    - Original text: From Input area
    - Encrypted text: From Output area
    - Algorithms used: Concatenated string (e.g., "AES DES")
    - User ID: From current session
4. **Executes SQL INSERT:**

```
INSERT INTO encrypted_data
(user_id, original_text, encrypted_text, algorithms_used)
VALUES (?, ?, ?, ?)
```

5. **Success:**

- Shows confirmation dialog with algorithm info
- Automatically refreshes "My Encrypted Data" tab
- Record includes timestamp (auto-generated)

Error Scenarios:

- Not logged in → "Please login first!"
- No content → "Please encrypt some data first!"
- No algorithms selected → "No algorithms were selected!"
- Database error → Shows detailed SQL error message

3.8 My Encrypted Data Tab

Purpose: View and manage all encrypted records saved by current user.

Components:

Data Table (JTable):

- **Columns:**
  1. ID: Record identifier (auto-increment)
  2. Original Text: First 50 characters + "..." if longer
  3. Encrypted Text: First 50 characters + "..." if longer
  4. Algorithms: Space-separated list (e.g., "AES Blowfish")
  5. Date: Timestamp of encryption (YYYY-MM-DD HH:MM:SS)

Features:

- Read-only table (no direct editing)
- Single row selection mode
- Scrollable for many records
- Sorted by ID descending (newest first)

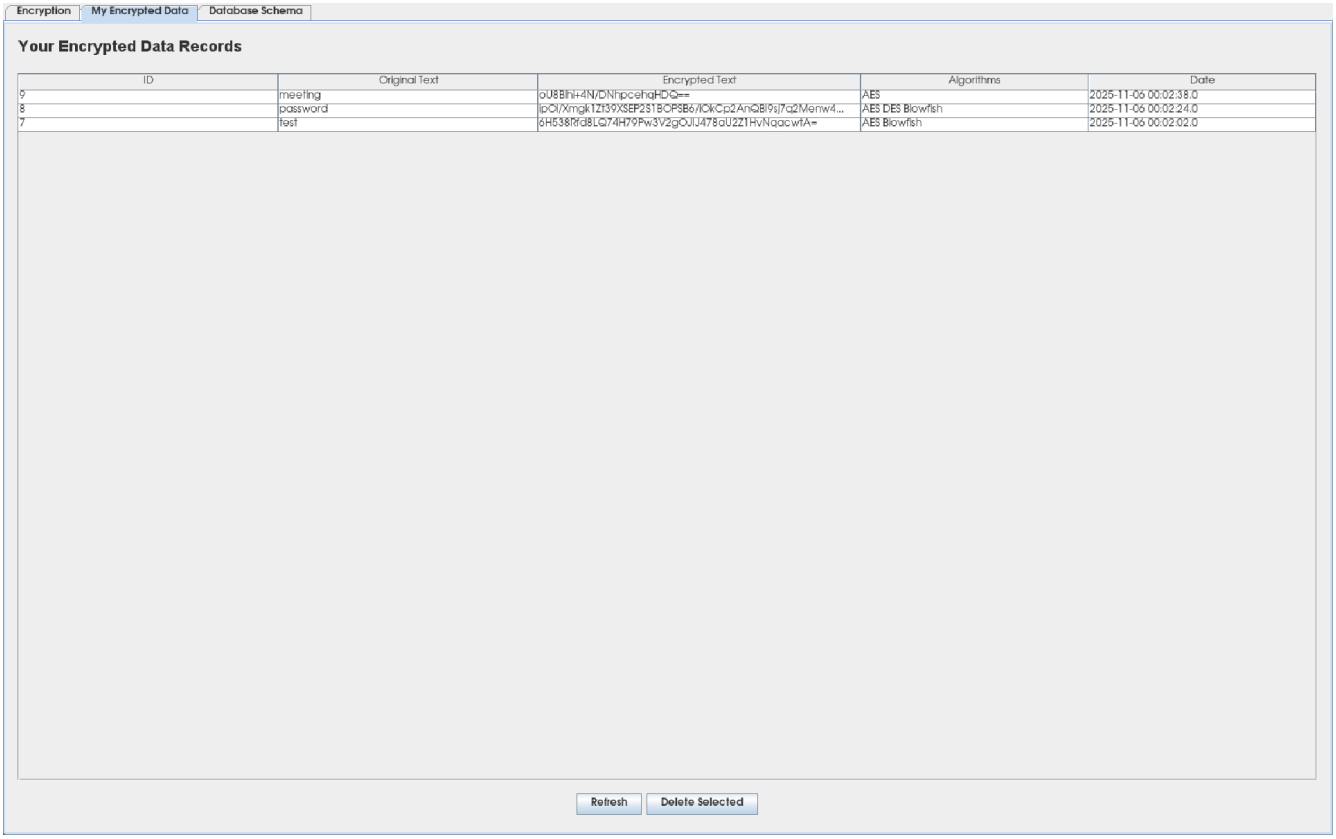
Action Buttons:

1. **REFRESH:**
   - Reloads data from database
   - Queries only current user's records
   - SQL: `SELECT * FROM encrypted_data WHERE user_id = ? ORDER BY id DESC`
2. **DELETE SELECTED:**
   - Deletes selected record from database
   - Shows confirmation dialog before deletion
   - SQL: `DELETE FROM encrypted_data WHERE id = ? AND user_id = ?`
   - Only deletes if record belongs to current user (security)

Data Loading:

- Automatically loads when user logs in

- Refreshes after saving new record
- Shows only records belonging to logged-in user (filtered by user_id)



3.9 Database Schema Tab

Purpose: Documentation and reference for database structure.

Content:

- Text-based schema documentation
- Displays in non-editable text area
- Monospaced font for alignment

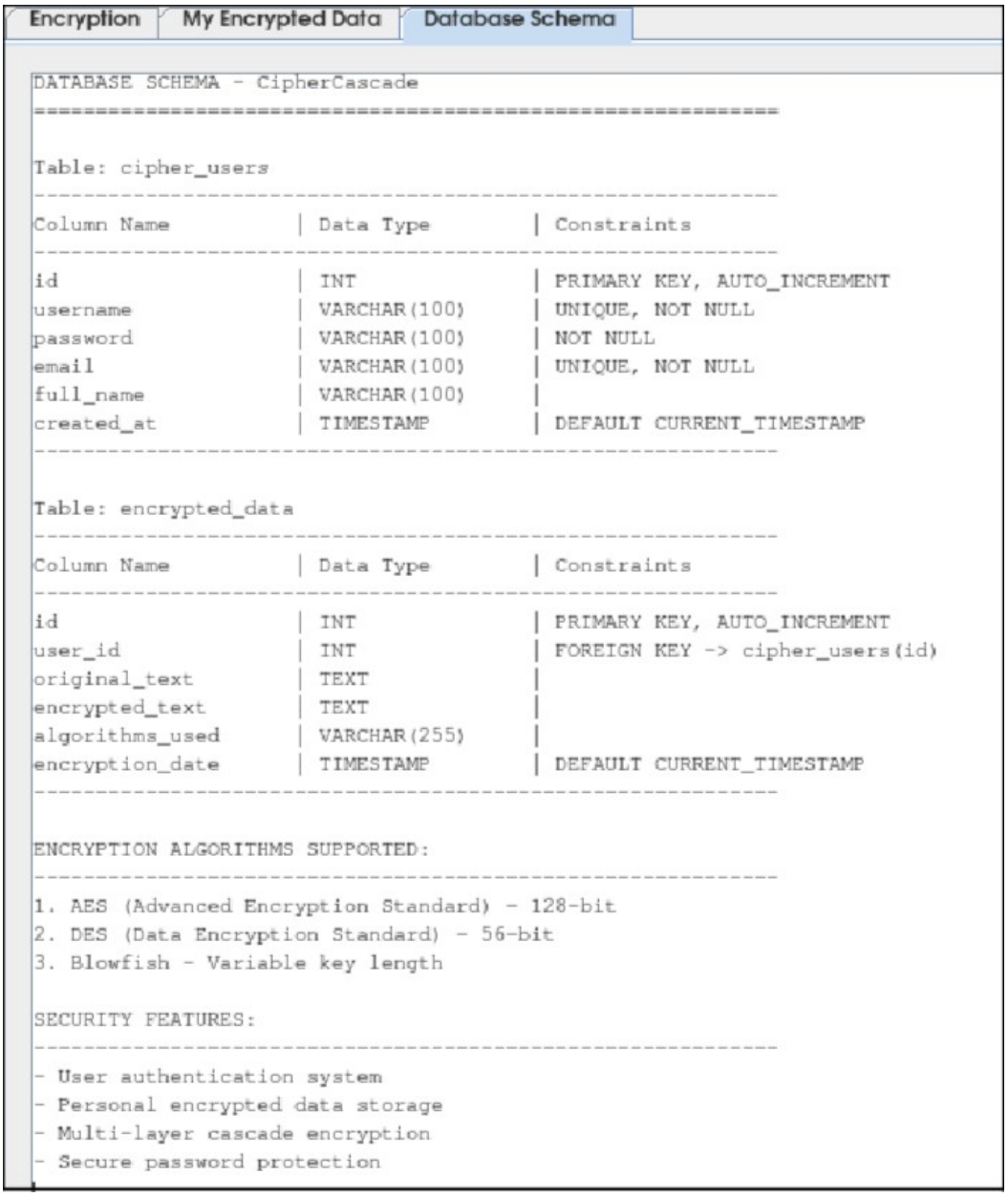
Information Shown:

1. **Table Structures:**
    - cipher_users table definition
    - encrypted_data table definition
    - Column names, data types, constraints
2. **Supported Algorithms:**
    - AES (Advanced Encryption Standard) - 128-bit
    - DES (Data Encryption Standard) - 56-bit
    - Blowfish - Variable key length
3. **Security Features:**

- User authentication system
- Personal encrypted data storage
- Multi-layer cascade encryption
- Secure password protection

Purpose:

- Quick reference for developers
- Understanding data structure
- No user interaction required (read-only)



References

1. Tutorialspoint. (2024). *Java Cryptography - Encrypting Data*. Tutorialspoint Java Cryptography Tutorial. https://www.tutorialspoint.com/java_cryptography/java_cryptography_encrypting_data.htm

2. National Institute of Standards and Technology. (2001). *Advanced Encryption Standard (AES)* (FIPS PUB 197). U.S. Department of Commerce. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

3. Oracle Corporation. (2024). *Java Cryptography Architecture (JCA) Reference Guide*. Oracle Java Documentation. https://docs.oracle.com/en/java/javase/17/security/java-cryptography-architecture-jca-reference-guide.html

4. Baeldung. (2024). *Java AES Encryption and Decryption*. Baeldung Java Tutorials. https://www.baeldung.com/java-aes-encryption-decryption

5. Schneier, B. (1993). Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish). *Fast Software Encryption: Cambridge Security Workshop Proceedings*, 191-204. https://www.schneier.com/academic/blowfish/