# FingerScanner

## Advanced Biometric Authentication System

Projet de Fin d'Année

2024-2025


**Presented by:**

**Hafedh GUENICHI** **Ibtihel SALHI**

**Amel MABROUKI** **Sarra KHADHRAOUI**

**Nesrine GANNOUNI**

**Supervised by:**

Dr. Makrem BELDI

TEK-UP Private College of Engineering Technology

# Dedication

*With profound gratitude and deep respect, we dedicate this work:*

*To our beloved parents, whose unwavering support, endless sacrifices, and unconditional love have been the foundation of our academic journey. Your belief in our capabilities has been our greatest motivation.*

*To our esteemed professors and mentors, who have illuminated our path with knowledge, wisdom, and guidance. Your expertise and dedication to education have inspired us to pursue excellence.*

*To our siblings and extended family members, who have provided emotional support and encouragement during challenging times. Your understanding and patience have been invaluable.*

*To our friends and colleagues, who have shared this academic journey with us, offering collaboration, constructive criticism, and camaraderie. The bonds we have formed will endure beyond these university walls.*

*To the technical community whose open-source contributions and shared knowledge have made this project possible. Your commitment to innovation and collaboration continues to advance our field.*

*This achievement is not ours alone but belongs to all who have contributed to our growth and development.*

***With sincere appreciation and heartfelt thanks.***

# Acknowledgments

We would like to express our sincere gratitude to our supervisor, Dr. Makreb Beldi, for their guidance, expertise, and continuous support throughout this project. Their insights and feedback have been invaluable to our work.

We also extend our thanks to the faculty and staff of the Higher Institute of TEK-UP University for providing us with the resources and knowledge necessary to complete this project.

Special thanks to our families and friends for their unwavering support and encouragement throughout our academic journey.

# Abstract

This report presents FingerScanner, an advanced biometric authentication system that leverages fingerprint recognition technology to provide secure, reliable, and user-friendly identity verification. The system processes fingerprint images, extracts unique minutiae points, and creates secure hash templates for authentication purposes.

The project implements a modern technology stack including Flask and FastAPI for backend services, MongoDB for data storage, and Next.js with shadcn/ui for the frontend interface. The system architecture follows a microservices approach, separating concerns between authentication, fingerprint processing, and user management.

Key features include high-quality fingerprint capture and processing, secure authentication mechanisms, comprehensive user management, and API integration capabilities. The system is designed with security and compliance in mind, implementing encryption, secure hashing, and privacy-preserving techniques.

This report details the implementation, architecture, and functionality of the FingerScanner system, providing insights into the technologies used and the solutions developed to address the challenges of biometric authentication.

**Keywords:** Biometric Authentication, Fingerprint Recognition, Flask, FastAPI, MongoDB, Next.js, Microservices

# Contents

# List of Figures

# List of Abbreviations

| | |
|---|---|
| **API** | Application Programming Interface |
| **CRUD** | Create, Read, Update, Delete |
| **CSS** | Cascading Style Sheets |
| **HTML** | HyperText Markup Language |
| **HTTP** | HyperText Transfer Protocol |
| **JSON** | JavaScript Object Notation |
| **JWT** | JSON Web Token |
| **MVC** | Model-View-Controller |
| **REST** | Representational State Transfer |
| **UI** | User Interface |
| **UX** | User Experience |

# Chapter 1

# Introduction

## 1.1 Project Overview

FingerScanner is an advanced biometric authentication system designed to provide secure and reliable identity verification through fingerprint recognition. In an era where digital security is paramount, biometric authentication offers a robust solution that combines security with user convenience. This project implements a comprehensive system that captures, processes, and verifies fingerprints for authentication purposes.

## 1.2 Problem Statement

Traditional authentication methods such as passwords and PINs suffer from several limitations, including vulnerability to theft, forgetfulness, and sharing. Biometric authentication addresses these issues by using unique physical characteristics that cannot be easily forgotten, shared, or stolen. However, implementing an effective biometric system presents challenges in terms of accuracy, security, and user experience.

## 1.3 Objectives

The primary objectives of the FingerScanner project are:

- To develop a secure and reliable fingerprint authentication system
- To implement a modern, scalable architecture using industry-standard technologies
- To create an intuitive and responsive user interface
- To ensure data privacy and security through encryption and secure hashing
- To provide comprehensive API integration capabilities for third-party applications

## 1.4 Scope

The scope of this project encompasses:

- Fingerprint capture and processing

- User registration and authentication
- Secure storage of biometric templates
- Administrative interface for user management
- API endpoints for integration with other systems

## 1.5 Report Structure

This report is structured as follows:

- Chapter 2 provides an overview of the technology stack used in the project
- Chapter 3 details the system architecture and design
- Chapter 4 explains the implementation of the backend services
- Chapter 5 covers the frontend implementation
- Chapter 6 discusses the fingerprint processing algorithms
- Chapter 7 presents the security measures implemented
- Chapter 8 concludes the report and suggests future enhancements
- Appendices provide additional information, including screenshots and code samples

# Chapter 2

# Technology Stack

**Technology Stack Overview**

The FingerScanner project utilizes a modern technology stack to deliver a robust and scalable solution

## 2.1 Backend Technologies

### 2.1.1 Flask


Figure 2.1: Flask Logo

Flask is a lightweight WSGI web application framework in Python. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. In the FingerScanner project, Flask is used for handling authentication routes and user management. Its simplicity and flexibility make it an ideal choice for these specific components.

Key features of Flask utilized in this project:

- Blueprint system for organizing routes
- Flask-JWT-Extended for token-based authentication
- Flask-Bcrypt for password hashing
- Flask-Cors for handling cross-origin resource sharing

### 2.1.2 FastAPI


Figure 2.2: FastAPI Logo

FastAPI is a modern, high-performance web framework for building APIs with Python. It is based on standard Python type hints and provides automatic interactive documentation. In the FingerScanner project, FastAPI is used for the main API endpoints, particularly those related to fingerprint processing and analysis.

Key features of FastAPI utilized in this project:

- Automatic data validation using Pydantic models
- Asynchronous request handling for improved performance
- Interactive API documentation with Swagger UI
- Dependency injection system for clean code organization

### 2.1.3 MongoDB


Figure 2.3: MongoDB Logo

MongoDB is a document-oriented NoSQL database that provides high performance, high availability, and easy scalability. In the FingerScanner project, MongoDB is used for storing user profiles, encrypted fingerprint templates, and system logs.

Key features of MongoDB utilized in this project:

- Document-based storage for flexible data modeling
- Indexing for efficient queries
- MongoDB Atlas for cloud-based database management
- PyMongo and Motor for Python integration

## 2.2 Frontend Technologies

### 2.2.1 Next.JS


Figure 2.4: Next.js Logo

Next.js is a React framework that enables server-side rendering, static site generation, and other advanced features. In the FingerScanner project, Next.js is used for building the user interface, providing a fast and responsive experience.

Key features of Next.js utilized in this project:

- App Router for simplified routing and layouts
- Server Components for improved performance
- API Routes for backend functionality
- Image optimization for efficient loading

### 2.2.2 Shadcn/UI


Figure 2.5: Shadcn/UI Components

Shadcn/UI is a collection of reusable components built with Radix UI and styled with Tailwind CSS. In the FingerScanner project, it is used to create a consistent and accessible user interface.

Key features of Shadcn/UI utilized in this project:

- Accessible UI components
- Customizable design system
- Integration with Tailwind CSS
- Dark mode support

## 2.3 Additional Technologies

### 2.3.1 OpenCV & NumPy

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices. In the FingerScanner project, these libraries are used for fingerprint image processing and feature extraction.

### 2.3.2 JWT Authentication

JSON Web Tokens (JWT) provide a secure method for transmitting information between parties as a JSON object. In the FingerScanner project, JWT is used for authentication and authorization, ensuring secure access to protected resources.

### 2.3.3 Docker & Docker Compose

Docker is a platform for developing, shipping, and running applications in containers. Docker Compose is a tool for defining and running multi-container Docker applications. In the FingerScanner project, Docker and Docker Compose are used for containerization and deployment.

# Chapter 3

# System Architecture

## 3.1 Overall Architecture

The FingerScanner system follows a microservices architecture, separating concerns between different components of the application. This approach allows for better scalability, maintainability, and deployment flexibility.

The main components of the system are:

- **Authentication Service (Flask)**: Handles user registration, login, and session management
- **Fingerprint Processing Service (FastAPI)**: Processes fingerprint images and performs matching
- **User Management Service (FastAPI)**: Manages user profiles and permissions
- **Database Layer (MongoDB)**: Stores user data, fingerprint templates, and system logs
- **Frontend Application (Next.js)**: Provides the user interface for interacting with the system

## 3.2 Data Flow

The data flow in the FingerScanner system follows a typical client-server pattern, with the frontend application communicating with the backend services through RESTful API calls.

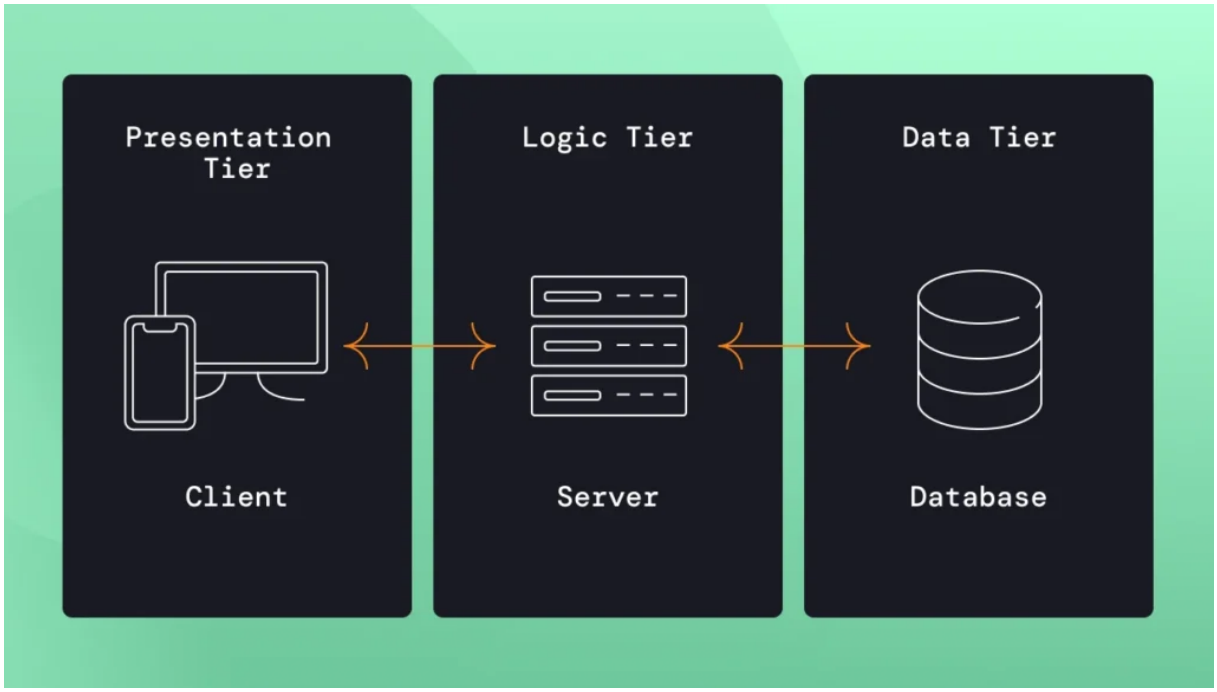


Figure 3.1: Data Flow Diagram

The general flow of data is as follows:

1. User interacts with the frontend application
2. Frontend sends requests to the appropriate backend service
3. Backend service processes the request and interacts with the database if necessary
4. Backend service returns a response to the frontend
5. Frontend updates the user interface based on the response

## 3.3 Database Schema

The MongoDB database schema is designed to store user profiles, fingerprint templates, and system logs in separate collections.

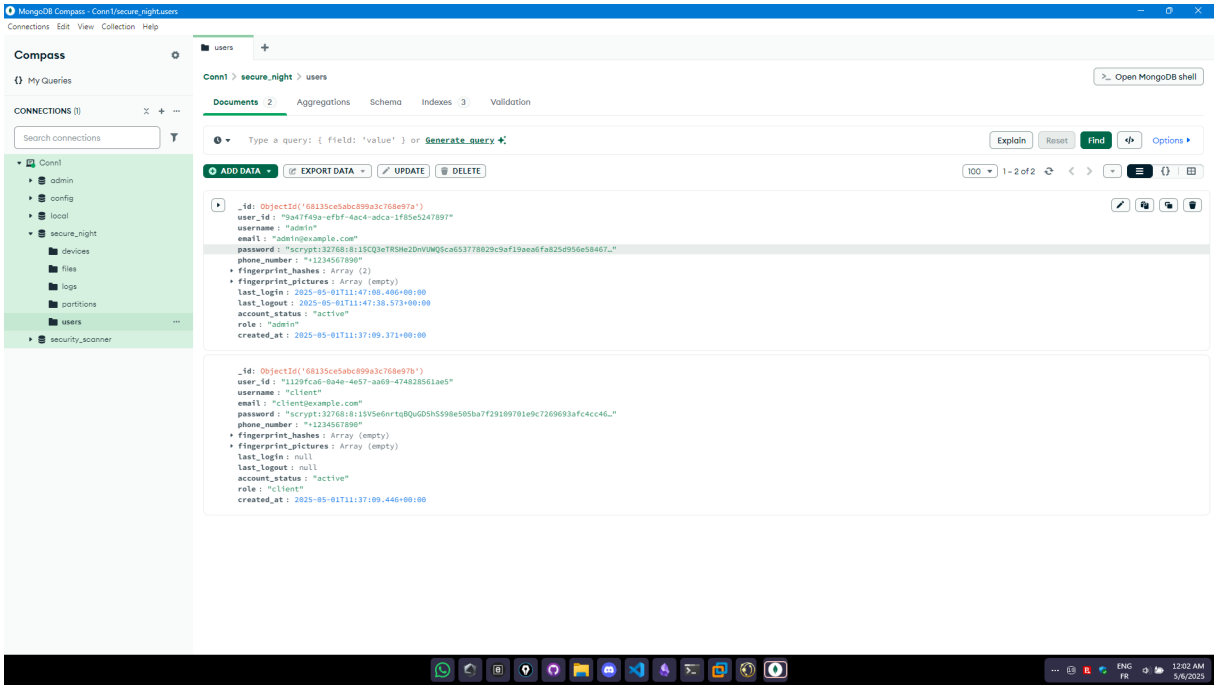


Figure 3.2: Database Schema

The main collections in the database are:

- **Users**: Stores user profile information
- **FingerprintTemplates**: Stores encrypted fingerprint templates
- **AuthLogs**: Stores authentication attempts and results
- **SystemLogs**: Stores system-level logs for monitoring and debugging

# Chapter 4

# Backend Implementation

## 4.1 Flask Authentication Service

The Flask Authentication Service is responsible for user registration, login, and session management. It provides endpoints for these operations and handles the generation and validation of JWT tokens.

### 4.1.1 Key Components

- **User Model**: Defines the structure of user data
- **Authentication Controller**: Handles registration, login, and token refresh
- **JWT Manager**: Manages the creation and validation of JWT tokens
- **Password Hasher**: Securely hashes and verifies passwords

### 4.1.2 Authentication Flow

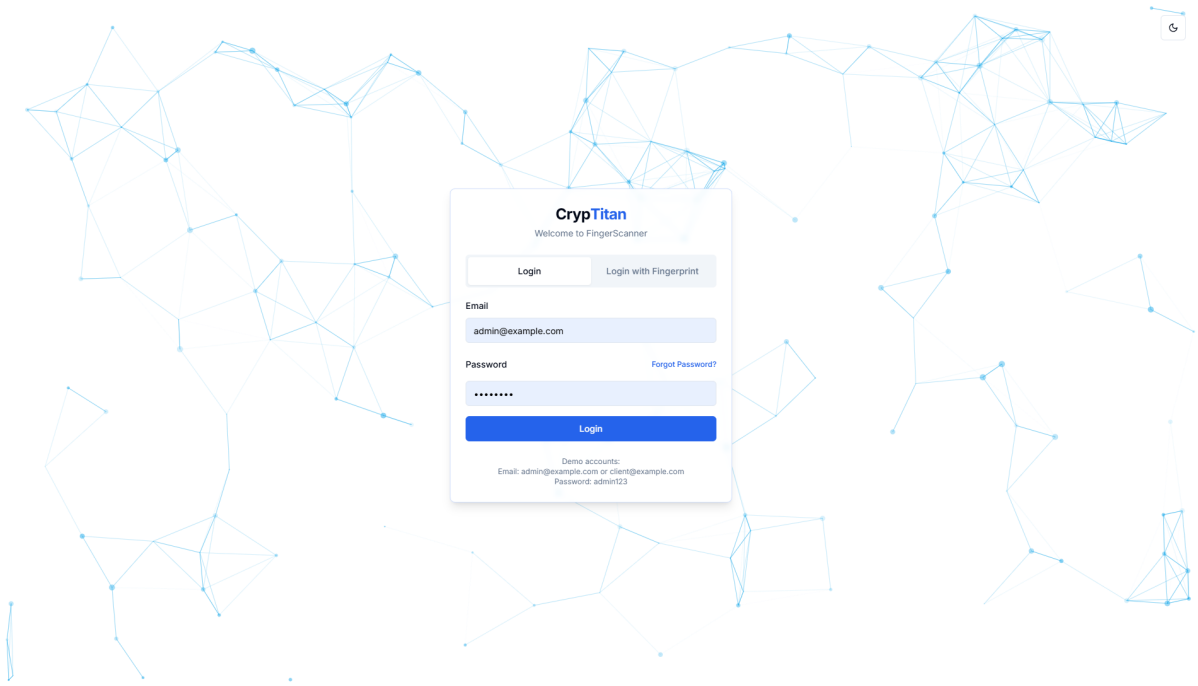


Figure 4.1: Authentication Flow

The authentication flow is as follows:

1. User submits registration or login credentials
2. Server validates the credentials
3. If valid, server generates JWT tokens (access and refresh)
4. Tokens are returned to the client
5. Client includes the access token in subsequent requests
6. Server validates the token for each protected request

## 4.2 FastAPI Fingerprint Service

The FastAPI Fingerprint Service is responsible for processing fingerprint images and performing matching operations. It provides endpoints for uploading fingerprint images, extracting features, and verifying fingerprints.

### 4.2.1 Key Components

- **Image Processor**: Enhances and normalizes fingerprint images
- **Feature Extractor**: Identifies minutiae points and other features

- **Template Generator**: Creates secure templates from extracted features
- **Matcher**: Compares templates for verification

### 4.2.2 Fingerprint Processing Pipeline

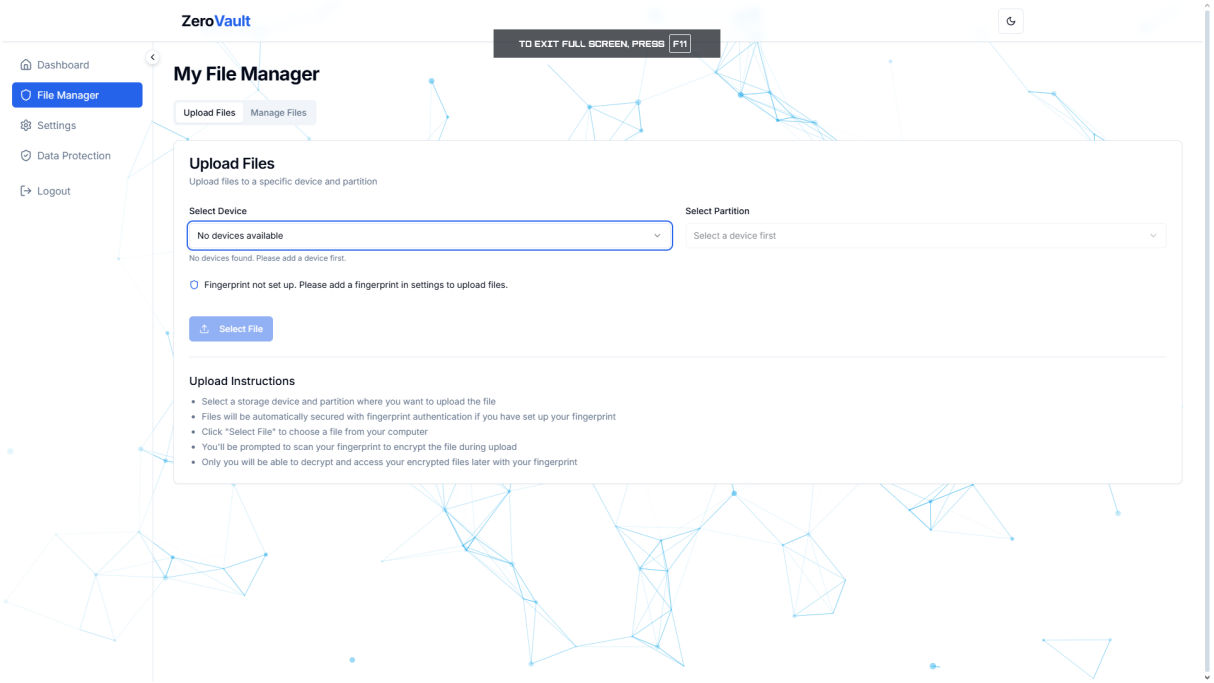


Figure 4.2: Fingerprint Processing Pipeline

The fingerprint processing pipeline consists of the following steps:

1. Image acquisition from the client
2. Preprocessing (enhancement, normalization)
3. Segmentation (isolating the fingerprint region)
4. Feature extraction (identifying minutiae points)
5. Template generation (creating a secure template)
6. Template storage or matching

## 4.3 API Endpoints

The FingerScanner system provides a comprehensive set of API endpoints for interacting with the system. Below are some of the key endpoints:

**POST** `/api/auth/register`

Creates a new user account in the system

**POST** `/api/auth/login`

Authenticates a user and issues JWT tokens

**POST** `/api/auth/update-fingerprint`

Updates a user's fingerprint template

**GET** `/api/auth/verify-fingerprint`

Verifies a fingerprint against a stored template

**GET** `/api/users/profile`

Retrieves the current user's profile information

PUT /api/users/profile

Updates the current user's profile information

## 4.4 Database Integration

The backend services integrate with MongoDB using PyMongo for Flask and Motor for FastAPI. This allows for efficient storage and retrieval of user data, fingerprint templates, and system logs.

```
# Example of MongoDB integration with Flask
from flask import Flask
from flask_pymongo import PyMongo

app = Flask(__name__)
app.config["MONGO_URI"] = "mongodb://localhost:27017/fingerscanner"
mongo = PyMongo(app)

@app.route("/users")
def get_users():
    users = mongo.db.users.find({}, {"password": 0})
    return jsonify([user for user in users])

# Example of MongoDB integration with FastAPI
from fastapi import FastAPI
from motor.motor_asyncio import AsyncIOMotorClient

app = FastAPI()

@app.on_event("startup")
async def startup_db_client():
    app.mongodb_client = AsyncIOMotorClient("mongodb://localhost:27017")
    app.mongodb = app.mongodb_client["fingerscanner"]

@app.on_event("shutdown")
async def shutdown_db_client():
    app.mongodb_client.close()

@app.get("/users")
async def get_users():
    users = await app.mongodb["users"].find({}, {"password": 0}).to_list(1000)
    return users
```

# Chapter 5

# Frontend Implementation

## 5.1 Next.js Application Structure

The frontend application is built using Next.js with the App Router, which provides a file-based routing system and support for React Server Components.

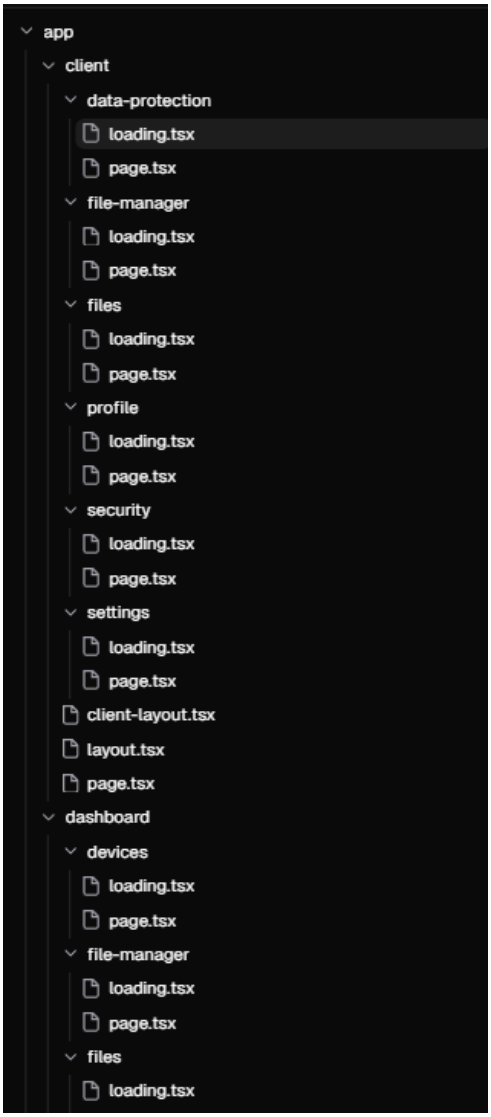


Figure 5.1: Next.js Application Structure

The application structure follows the Next.js App Router conventions:

- **app/**: Contains the application routes and layouts
- **components/**: Contains reusable UI components
- **lib/**: Contains utility functions and hooks
- **public/**: Contains static assets

## 5.2 User Interface Components

The user interface is built using shadcn/ui components, which provide a consistent and accessible design system.

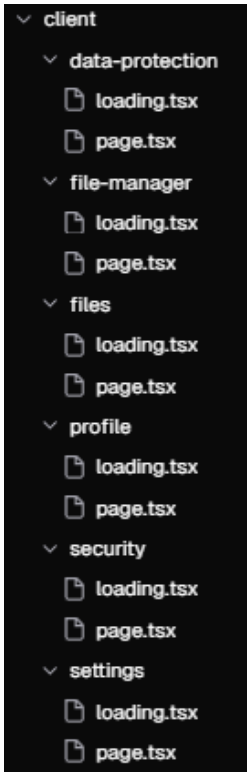


Figure 5.2: User Interface Components

Key UI components include:

- **Authentication Forms**: Registration and login forms
- **Fingerprint Capture**: Interface for capturing fingerprint images
- **User Dashboard**: Overview of user activity and settings
- **Admin Panel**: Interface for managing users and system settings

## 5.3 State Management

State management in the frontend application is handled using React's built-in hooks and context API, with additional support from libraries like SWR for data fetching.

```
// Example of state management with React hooks and context
import { createContext, useContext, useState } from 'react';

const AuthContext = createContext();

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  const login = async (credentials) => {
    setLoading(true);
```

```
    try {
      const response = await fetch('/api/auth/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(credentials),
      });
      const data = await response.json();
      if (response.ok) {
        setUser(data.user);
        localStorage.setItem('token', data.access_token);
        return { success: true };
      } else {
        return { success: false, error: data.error };
      }
    } catch (error) {
      return { success: false, error: error.message };
    } finally {
      setLoading(false);
    }
  };

  const logout = () => {
    setUser(null);
    localStorage.removeItem('token');
  };

  return (
    <AuthContext.Provider value={{ user, loading, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
}

export function useAuth() {
  return useContext(AuthContext);
}
```

## 5.4 API Integration

The frontend application communicates with the backend services through RESTful API calls. This is handled using the Fetch API or libraries like Axios.

```
// Example of API integration with the Fetch API
async function updateFingerprint(fingerprintData) {
  const token = localStorage.getItem('token');

  try {
    const response = await fetch('/api/auth/update-fingerprint', {
```

```
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${token}`,
      },
      body: JSON.stringify({ fingerprint: fingerprintData }),
    });

    const data = await response.json();

    if (response.ok) {
      return { success: true, data };
    } else {
      return { success: false, error: data.error };
    }
  } catch (error) {
    return { success: false, error: error.message };
  }
}
```

# Chapter 6

# Fingerprint Processing

## 6.1 Image Preprocessing

Fingerprint image preprocessing is a critical step in the fingerprint recognition process. It enhances the quality of the image and prepares it for feature extraction.

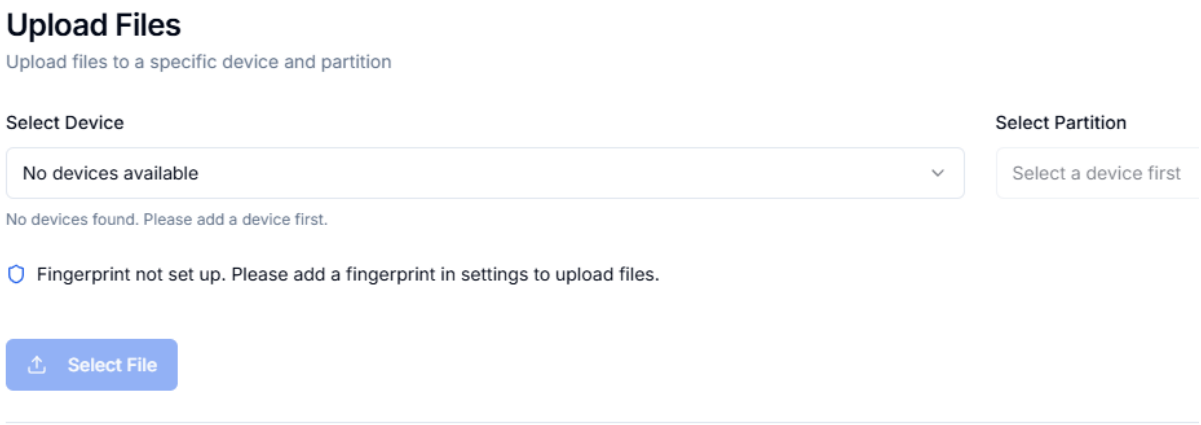


Figure 6.1: Fingerprint Image Preprocessing

The preprocessing steps include:

- **Grayscale Conversion**: Converting color images to grayscale
- **Normalization**: Adjusting the intensity values
- **Enhancement**: Improving the clarity of ridge structures
- **Binarization**: Converting to a binary image
- **Thinning**: Reducing ridge thickness to one pixel

## 6.2 Feature Extraction

Feature extraction involves identifying the unique characteristics of a fingerprint, primarily minutiae points such as ridge endings and bifurcations.

```python
def process_fingerprint(fingerprint_binary):
    """
    Process a fingerprint image to extract minutiae and generate a hash.

    Args:
        fingerprint_binary (bytes): Binary data of the fingerprint image

    Returns:
        str: Hash of the fingerprint minutiae
    """
    try:
        # Convert binary to image
        image = Image.open(io.BytesIO(fingerprint_binary))

        # Convert to numpy array for OpenCV processing
        img_array = np.array(image)

        # Convert to grayscale if it's a color image
        if len(img_array.shape) > 2 and img_array.shape[2] > 1:
            gray = cv2.cvtColor(img_array, cv2.COLOR_BGR2GRAY)
        else:
            gray = img_array

        # STEP 1: FINGERPRINT REGION DETECTION
        # 1.1 Apply Gaussian blur to reduce noise
        blurred = cv2.GaussianBlur(gray, (5, 5), 0)

        # 1.2 Apply Otsu's thresholding to get a binary image
        _, binary = cv2.threshold(
            blurred, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU
        )

        # 1.3 Apply morphological operations to clean the binary image
        kernel = np.ones((5, 5), np.uint8)
        cleaned = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)
        cleaned = cv2.morphologyEx(cleaned, cv2.MORPH_OPEN, kernel)

        # 1.4 Find contours to identify the fingerprint region
        contours, _ = cv2.findContours(
            cleaned, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
        )

        # 1.5 Find the largest contour (assumed to be the fingerprint)
        if contours:
            largest_contour = max(contours, key=cv2.contourArea)

            # 1.6 Create a mask for the fingerprint region
            mask = np.zeros_like(gray)
            cv2.drawContours(mask, [largest_contour], 0, 255, -1)

            # 1.7 Get bounding rectangle of the fingerprint
            x, y, w, h = cv2.boundingRect(largest_contour)

            # 1.8 Add a margin to the bounding rectangle
```

Figure 6.2: Fingerprint Feature Extraction

The feature extraction process includes:

- **Minutiae Detection**: Identifying ridge endings and bifurcations
- **Minutiae Filtering**: Removing false minutiae
- **Minutiae Characterization**: Determining type, position, and orientation

## 6.3 Template Generation

Template generation involves creating a compact and secure representation of the fingerprint features for storage and matching.

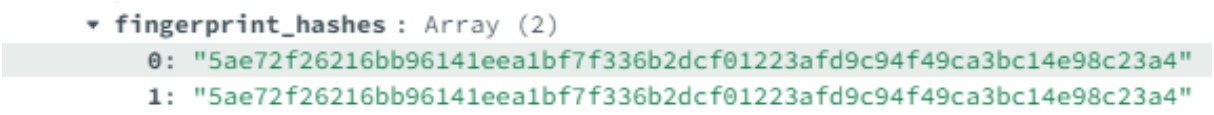


Figure 6.3: Fingerprint Template Generation

The template generation process includes:

- **Feature Selection**: Selecting the most distinctive features
- **Template Encoding**: Converting features into a compact format
- **Template Encryption**: Securing the template with encryption

# Chapter 7

# Security Measures

## 7.1 Data Encryption

Data encryption is a critical aspect of the FingerScanner system, ensuring that sensitive information is protected both in transit and at rest.

- **Transport Layer Security (TLS)**: All communication between the client and server is encrypted using TLS
- **Database Encryption**: Sensitive data in the database is encrypted using AES-256
- **Template Encryption**: Fingerprint templates are encrypted before storage

## 7.2 Authentication and Authorization

The system implements robust authentication and authorization mechanisms to ensure that only authorized users can access protected resources.

- **JWT-based Authentication**: Secure token-based authentication
- **Role-based Access Control**: Different access levels for different user roles
- **Multi-factor Authentication**: Option to require multiple forms of authentication

## 7.3 Privacy Considerations

The FingerScanner system is designed with privacy in mind, implementing measures to protect user data and comply with privacy regulations.

- **Data Minimization**: Only collecting necessary data
- **Purpose Limitation**: Using data only for specified purposes
- **User Consent**: Obtaining explicit consent for data collection
- **Data Deletion**: Providing mechanisms for users to delete their data

# Chapter 8

# Conclusion and Future Work

## 8.1 Summary

The FingerScanner project has successfully implemented an advanced biometric authentication system using fingerprint recognition technology. The system provides a secure, reliable, and user-friendly solution for identity verification, leveraging modern technologies such as Flask, FastAPI, MongoDB, and Next.js.

## 8.2 Achievements

The key achievements of the project include:

- Development of a robust fingerprint processing pipeline
- Implementation of secure authentication mechanisms
- Creation of an intuitive and responsive user interface
- Integration of comprehensive security measures
- Establishment of a scalable and maintainable architecture

## 8.3 Limitations

Despite the achievements, the project has some limitations:

- Dependency on the quality of fingerprint images
- Potential for false positives or false negatives in matching
- Limited support for different types of fingerprint sensors
- Scalability challenges with large numbers of users

## 8.4 Future Enhancements

Future enhancements to the system could include:

- Integration with other biometric modalities (e.g., face, iris)
- Implementation of liveness detection to prevent spoofing
- Enhancement of the matching algorithm for improved accuracy
- Development of mobile applications for wider accessibility
- Integration with existing identity management systems

# Bibliography


[1] Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.

[2] Ramírez, S. (2020). *FastAPI: Modern Python Web Development*. Packt Publishing.

[3] Chodorow, K. (2013). *MongoDB: The Definitive Guide*. O'Reilly Media.

[4] Wieruch, R. (2021). *The Road to React with Next.js*. Self-published.

[5] Jain, A. K., Ross, A., Nandakumar, K. (2011). *Introduction to Biometrics*. Springer.

[6] Maltoni, D., Maio, D., Jain, A. K., Prabhakar, S. (2009). *Handbook of Fingerprint Recognition*. Springer.

[7] Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice*. Pearson.

[8] Bradski, G., Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media.

# Appendix A

# User Interface Screenshots

## A.1 Login and Registration

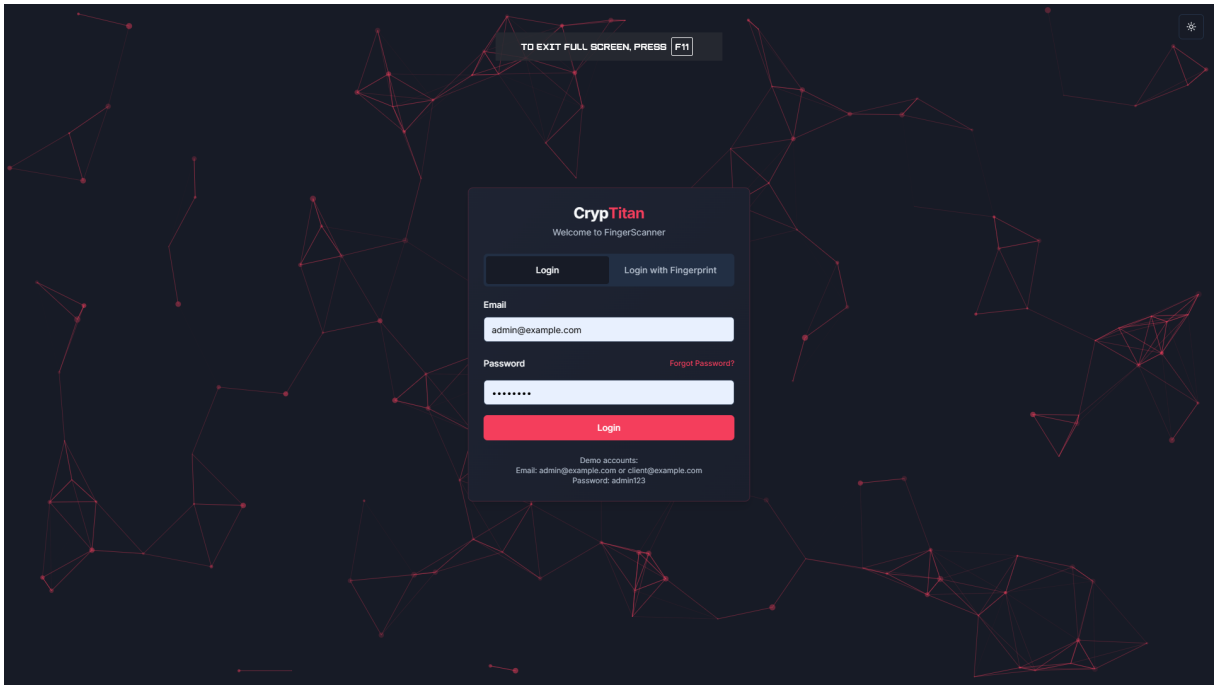


Figure A.1: Login Screen

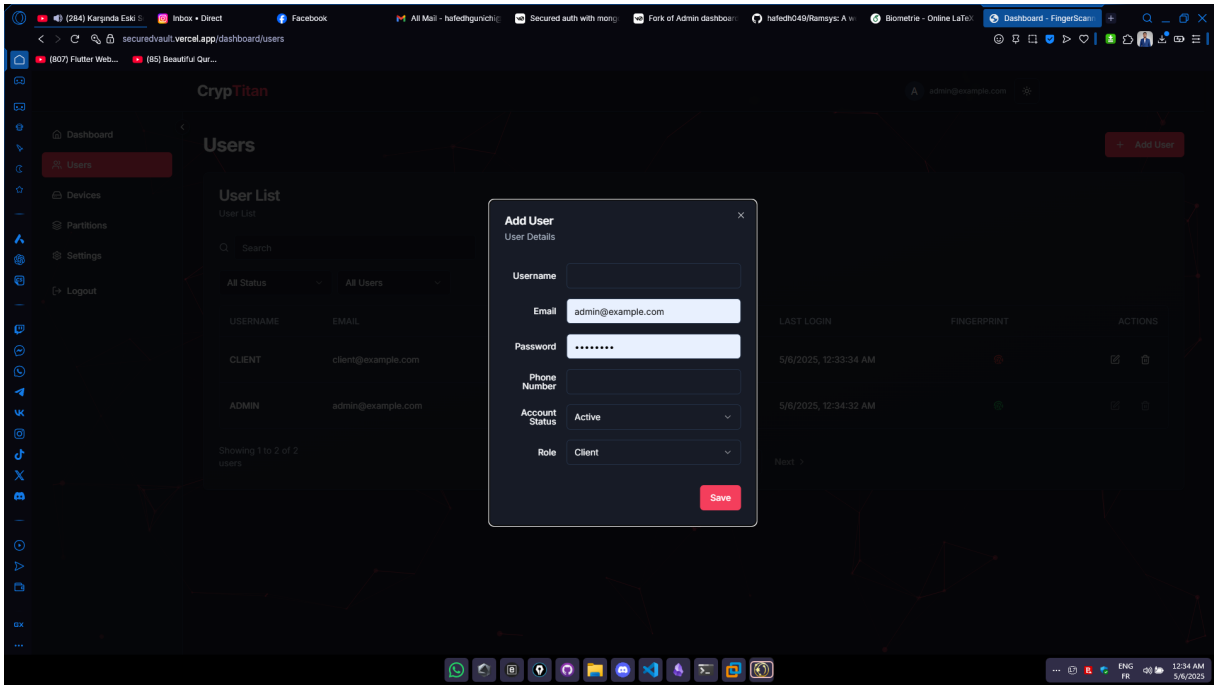


Figure A.2: Registration Screen

## A.2 User Dashboard

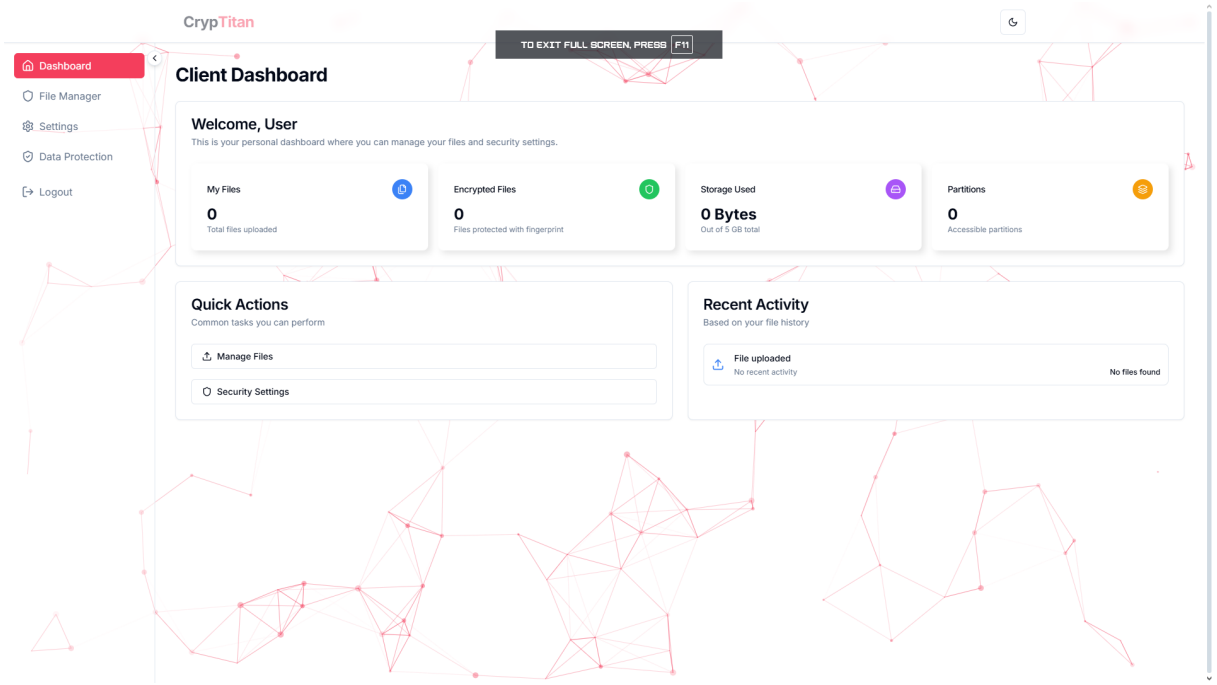


Figure A.3: User Dashboard

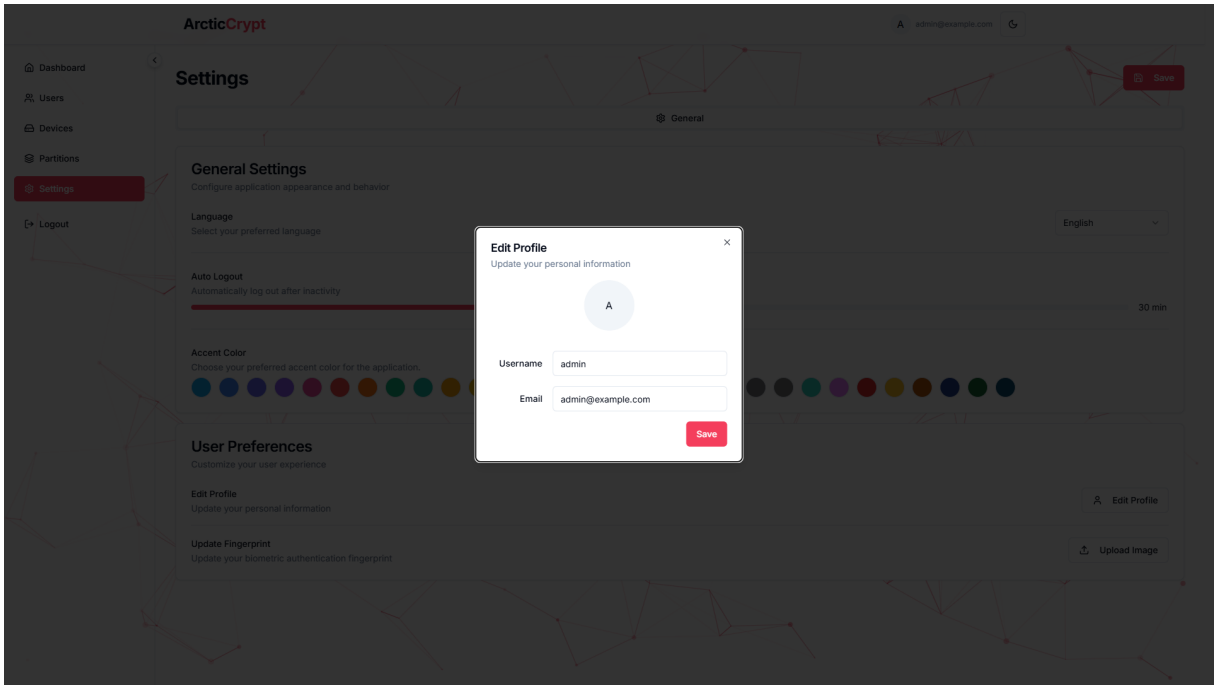


Figure A.4: User Profile Settings

## A.3 Fingerprint Management

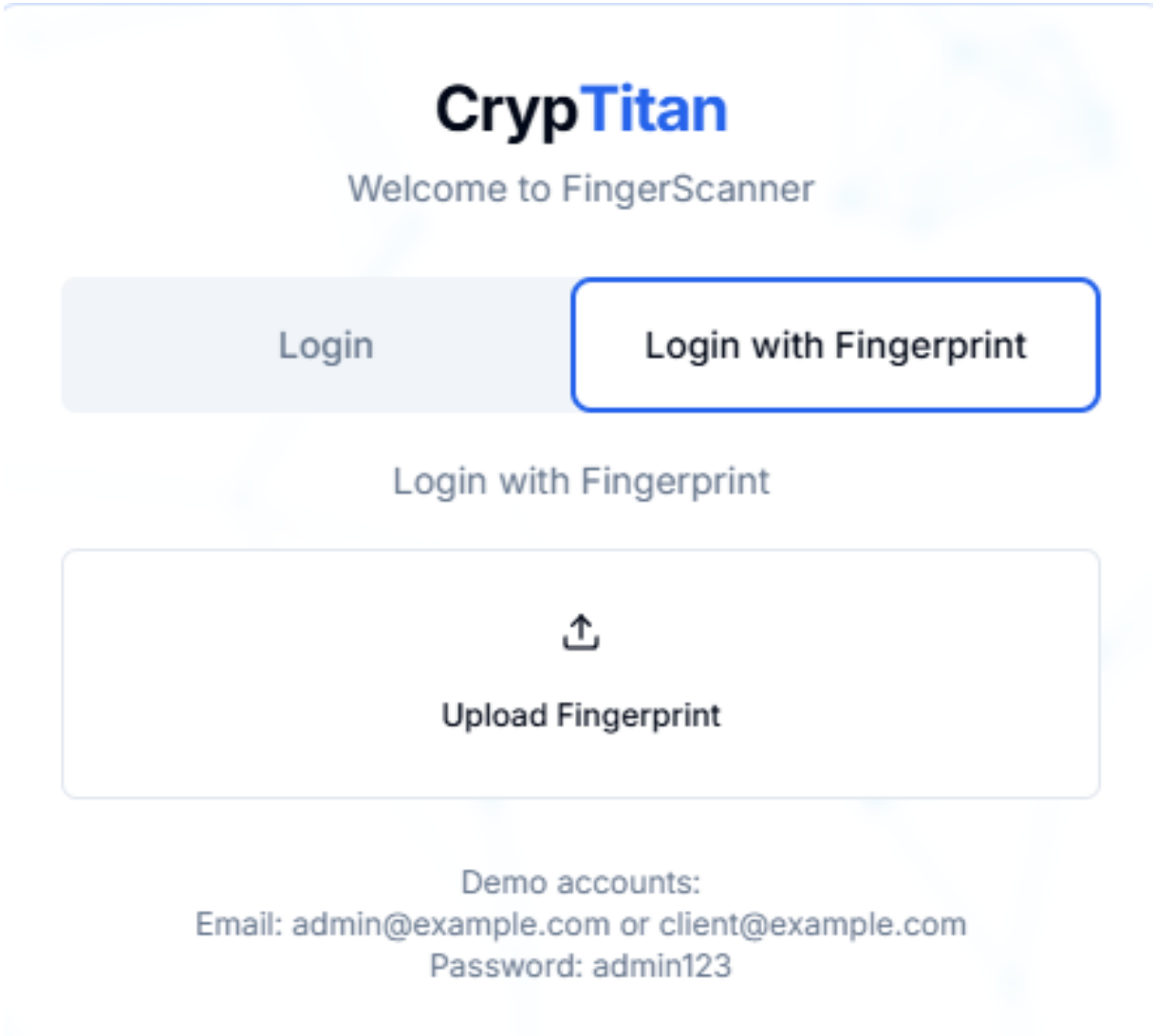


Figure A.5: Fingerprint Capture Interface

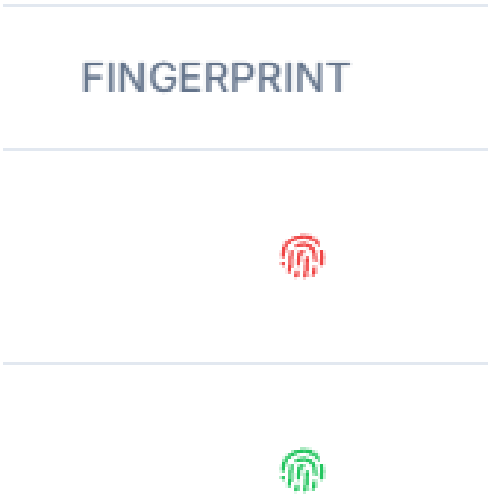


Figure A.6: Fingerprint Verification Result

## A.4 Admin Interface

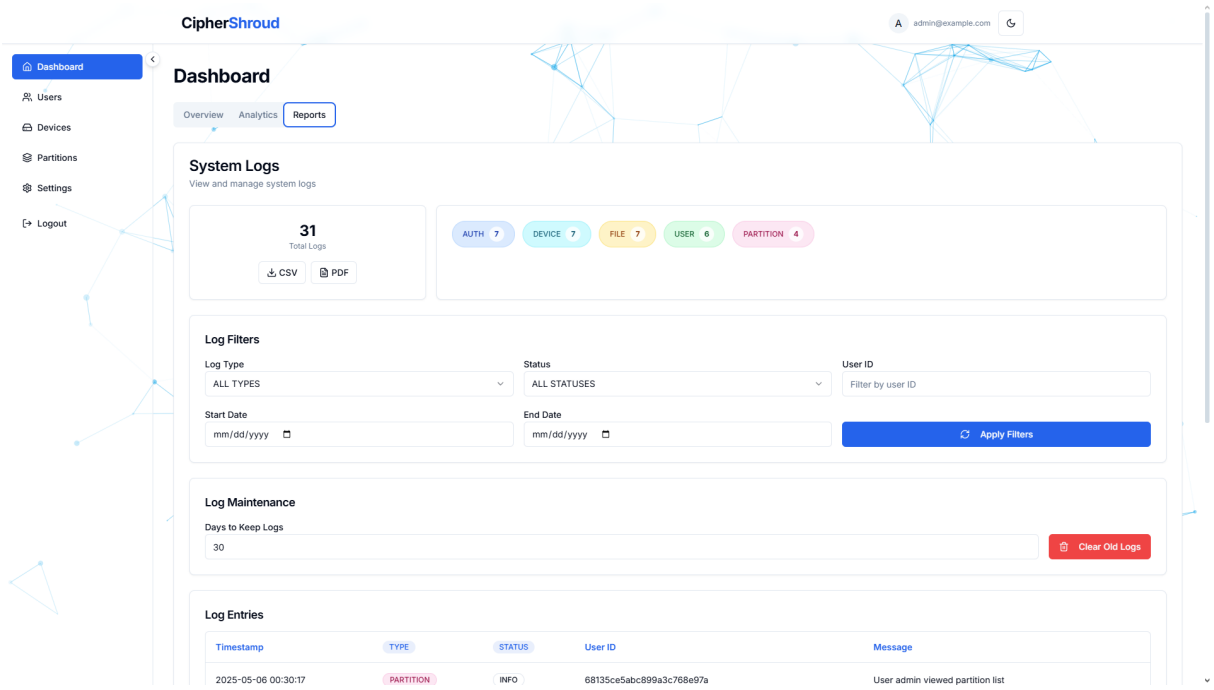


Figure A.7: Admin Dashboard

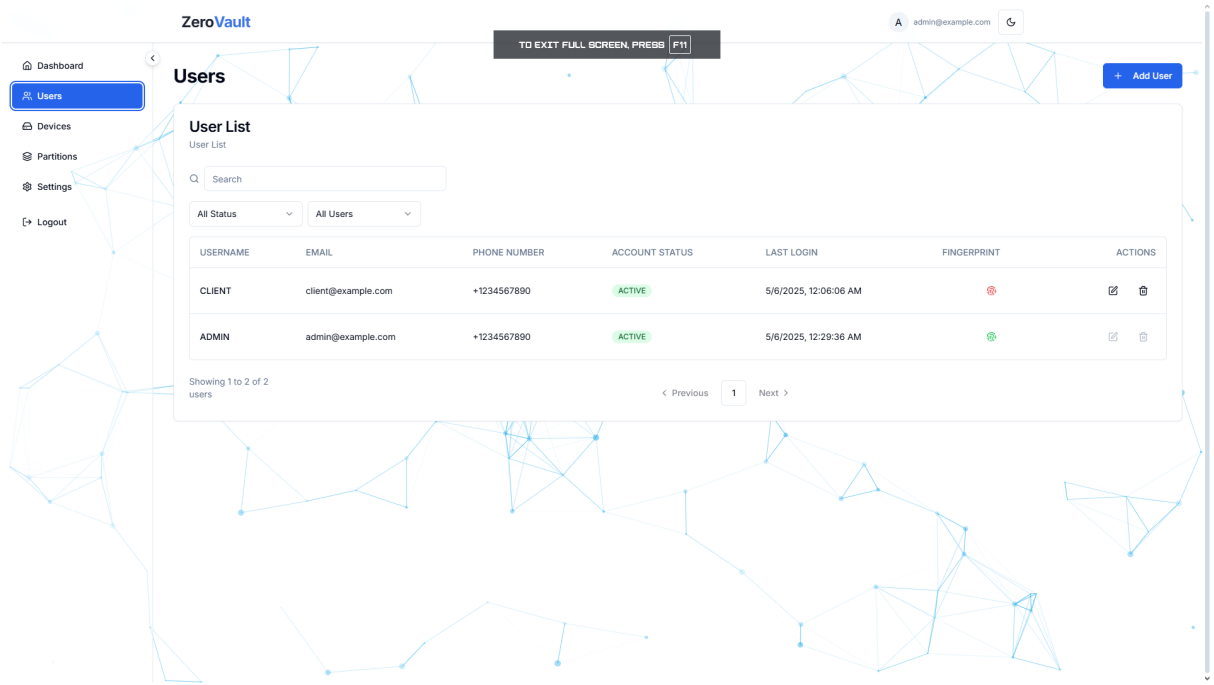


Figure A.8: User Management Interface

# Appendix B

# API Documentation

## B.1 Authentication API

```
// POST /api/auth/register
// Request
{
  "username": "john_doe",
  "email": "john@example.com",
  "password": "SecureP@ss123",
  "phone_number": "+1234567890"
}

// Response (201 Created)
{
  "message": "User registered successfully",
  "user": {
    "user_id": "550e8400-e29b-41d4-a716-446655440000",
    "username": "john_doe",
    "email": "john@example.com",
    "phone_number": "+1234567890",
    "account_status": "active",
    "role": "client",
    "created_at": "2023-04-15T14:30:45.123Z"
  }
}

// POST /api/auth/login
// Request
{
  "email": "john@example.com",
  "password": "SecureP@ss123"
}

// Response (200 OK)
{
  "message": "Login successful",
```

```
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "user_id": "550e8400-e29b-41d4-a716-446655440000",
    "username": "john_doe",
    "email": "john@example.com",
    "role": "client",
    "account_status": "active"
  }
}
```

## B.2 Fingerprint API

```
// POST /api/auth/update-fingerprint
// Request
{
  "fingerprint": "base64_encoded_fingerprint_image"
}

// Response (200 OK)
{
  "message": "Fingerprint updated successfully"
}

// GET /api/auth/verify-fingerprint
// Request
{
  "fingerprint": "base64_encoded_fingerprint_image"
}

// Response (200 OK)
{
  "message": "Fingerprint verified successfully",
  "match": true,
  "confidence": 0.95
}
```

# Appendix C

# Code Samples

## C.1 Fingerprint Processing

```
import cv2
import numpy as np

def preprocess_fingerprint(image):
    """
    Preprocess a fingerprint image for feature extraction.

    Args:
        image: The input fingerprint image

    Returns:
        The preprocessed image
    """
    # Convert to grayscale if needed
    if len(image.shape) > 2:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    else:
        gray = image

    # Normalize the image
    normalized = cv2.normalize(gray, None, 0, 255, cv2.NORM_MINMAX)

    # Apply Gaussian blur to reduce noise
    blurred = cv2.GaussianBlur(normalized, (5, 5), 0)

    # Apply adaptive thresholding
    binary = cv2.adaptiveThreshold(
        blurred, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
        cv2.THRESH_BINARY_INV, 11, 2
    )

    # Apply morphological operations to clean up the image
    kernel = np.ones((3, 3), np.uint8)
```

```
    cleaned = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)

    return cleaned

def extract_minutiae(image):
    """
    Extract minutiae points from a preprocessed fingerprint image.

    Args:
        image: The preprocessed fingerprint image

    Returns:
        A list of minutiae points (x, y, type, orientation)
    """
    # Thin the ridges to one pixel width
    thinned = cv2.ximgproc.thinning(image)

    # Find minutiae points
    minutiae = []
    for y in range(1, thinned.shape[0] - 1):
        for x in range(1, thinned.shape[1] - 1):
            if thinned[y, x] == 0:
                continue

            # Get 8-neighborhood
            neighbors = [
                thinned[y-1, x-1], thinned[y-1, x], thinned[y-1, x+1],
                thinned[y, x-1],                    thinned[y, x+1],
                thinned[y+1, x-1], thinned[y+1, x], thinned[y+1, x+1]
            ]

            # Count transitions from 0 to 1 in the neighborhood
            transitions = 0
            for i in range(len(neighbors)):
                if neighbors[i] == 0 and neighbors[(i + 1) % 8] == 1:
                    transitions += 1

            # Count neighbors that are 1
            ridge_pixels = sum(1 for n in neighbors if n == 1)

            # Ridge ending: one neighbor
            if ridge_pixels == 1:
                minutiae.append((x, y, 'ending', 0))

            # Ridge bifurcation: three or more neighbors
            elif ridge_pixels >= 3:
                minutiae.append((x, y, 'bifurcation', 0))
```

```
    return minutiae

def generate_template(minutiae):
    """
    Generate a template from minutiae points.

    Args:
        minutiae: A list of minutiae points

    Returns:
        A template representation of the fingerprint
    """
    # Simple template: list of minutiae with relative positions
    if not minutiae:
        return None

    # Find the center of mass
    center_x = sum(m[0] for m in minutiae) / len(minutiae)
    center_y = sum(m[1] for m in minutiae) / len(minutiae)

    # Create template with relative positions
    template = [
        (m[0] - center_x, m[1] - center_y, m[2], m[3])
        for m in minutiae
    ]

    return template

def match_templates(template1, template2, threshold=0.7):
    """
    Match two fingerprint templates.

    Args:
        template1: The first template
        template2: The second template
        threshold: The matching threshold

    Returns:
        A tuple of (match, confidence)
    """
    if not template1 or not template2:
        return False, 0.0

    # Count matching minutiae
    matches = 0
    total = min(len(template1), len(template2))

    for m1 in template1:
```

```
        for m2 in template2:
            # If minutiae are of the same type and close enough
            if m1[2] == m2[2] and \
               abs(m1[0] - m2[0]) < 10 and \
               abs(m1[1] - m2[1]) < 10:
                matches += 1
                break

    confidence = matches / total if total > 0 else 0.0
    match = confidence >= threshold

    return match, confidence
```

## C.2 Frontend Authentication

```
// auth.js
import { createContext, useContext, useState, useEffect } from 'react';
import { useRouter } from 'next/navigation';

const AuthContext = createContext();

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const router = useRouter();

  useEffect(() => {
    // Check if user is logged in
    const checkAuth = async () => {
      const token = localStorage.getItem('token');
      if (!token) {
        setLoading(false);
        return;
      }

      try {
        const response = await fetch('/api/auth/me', {
          headers: {
            'Authorization': `Bearer ${token}`
          }
        });

        if (response.ok) {
          const data = await response.json();
          setUser(data.user);
        } else {
          // Token is invalid or expired
          localStorage.removeItem('token');
```

```
      }
    } catch (error) {
      console.error('Authentication check failed:', error);
    } finally {
      setLoading(false);
    }
  };

  checkAuth();
}, []);

const login = async (credentials) => {
  setLoading(true);
  try {
    const response = await fetch('/api/auth/login', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(credentials),
    });

    const data = await response.json();

    if (response.ok) {
      setUser(data.user);
      localStorage.setItem('token', data.access_token);
      router.push('/dashboard');
      return { success: true };
    } else {
      return { success: false, error: data.error };
    }
  } catch (error) {
    return { success: false, error: error.message };
  } finally {
    setLoading(false);
  }
};

const register = async (userData) => {
  setLoading(true);
  try {
    const response = await fetch('/api/auth/register', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(userData),
    });

    const data = await response.json();
```

```
      if (response.ok) {
        router.push('/login');
        return { success: true };
      } else {
        return { success: false, error: data.error };
      }
    } catch (error) {
      return { success: false, error: error.message };
    } finally {
      setLoading(false);
    }
  };

  const logout = () => {
    setUser(null);
    localStorage.removeItem('token');
    router.push('/login');
  };

  return (
    <AuthContext.Provider value={{ user, loading, login, register, logout }}>
      {children}
    </AuthContext.Provider>
  );
}

export function useAuth() {
  return useContext(AuthContext);
}
```