Advanced Authentication Security Lab

Complete Features Documentation

A-Z Comprehensive Guide

Generated: January 21, 2026

Version 1.0 - Full Stack Security Education Platform

Table of Contents

A. Architecture Overview

System Architecture

The Advanced Authentication Security Lab is a full-stack web application demonstrating password security concepts through practical implementation. It features a hybrid architecture combining Flask backend API with client-side JavaScript fallback.

Layer	Technology	Purpose
Frontend	HTML5, CSS3, JavaScript	User interface and client-side logic
Backend	Python Flask 3.0	REST API and business logic
Database	SQLite	Data persistence
Styling	Bootstrap 5.3 + Custom CSS	Responsive design
Crypto	Argon2, BCrypt, MD5, SHA	Password hashing

Three-Tier Architecture

1. Presentation Layer (Frontend): Browser-based interface serving 8 distinct pages with persistent sidebar navigation. Uses modern glassmorphism design with responsive layouts.

2. Application Layer (Backend): Flask REST API handling authentication, password hashing with multiple algorithms, and database operations. Includes rate limiting and migration capabilities.

3. Data Layer (Database): SQLite database with three tables (users, login_attempts, password_history) providing full ACID compliance and relationship integrity.

B. Backend Components

Flask Application Structure

Location: `backend/app.py` (811 lines)

The backend is built with Flask 3.0 and provides 15+ RESTful API endpoints. It implements professional security practices including CORS configuration, rate limiting, and comprehensive error handling.

Component	File	Lines	Purpose
Main Application	app.py	811	API routes and business logic
Database Models	models.py	~150	SQLAlchemy ORM models
Dependencies	requirements.txt	4	Python package list
Database Instance	instance/*.db	N/A	SQLite database file

Key Backend Features

• Multi-algorithm password hashing (MD5, SHA-1, BCrypt, Argon2id)

• Automatic migration from weak to strong hashing algorithms

• Rate limiting (5 attempts per minute) with lockout mechanism

• Password history tracking (prevents reuse of last 5 passwords)

• Salt generation and pepper implementation

• Hashcat export functionality for security testing

• Comprehensive logging and debugging output

• CORS configuration for cross-origin requests

• SQLAlchemy ORM with relationship management

• Environment variable configuration support

C. Core Features - All 8 Pages Explained

1. Login System

File: index.html

Location: Root directory

URL: `http://localhost:8000/index.html`

Size: 482 lines

Main entry point providing user authentication with hybrid backend/localStorage support.

Key Features:

• Username and password authentication

• Automatic backend connectivity check

• Falls back to LocalStorage if backend unavailable

• Session management

• Redirect to dashboard on successful login

• Error handling with user-friendly messages

• Password visibility toggle

• Remember me functionality (LocalStorage)

• Responsive mobile design

Technologies: HTML5, JavaScript (loginUser function), API: POST /api/login

2. User Registration

File: register.html

Location: pages/register.html

URL: `http://localhost:8000/pages/register.html`

Size: ~350 lines

Comprehensive registration system with algorithm selection and password security features.

Key Features:

• 4 hash algorithms: MD5, SHA-1, BCrypt, Argon2id

• Real-time password strength meter

• Have I Been Pwned API integration

• Configurable cost factors (BCrypt rounds, Argon2 memory)

• Client-side password validation

• Duplicate username detection

• Visual algorithm comparison

• Educational tooltips explaining each algorithm

• Export registration data

Technologies: CryptoJS, BCrypt.js, Argon2-browser, API: POST /api/register

3. Admin Dashboard

File: dashboard.html

Location: pages/dashboard.html

URL: `http://localhost:8000/pages/dashboard.html`

Size: 340 lines

Central management interface displaying all users and system statistics.

Key Features:

• User table with sortable columns

• Algorithm distribution statistics

• Security badge indicators (Vulnerable/Weak/Secure)

• Hash display with copy-to-clipboard

• User deletion capability

• Bulk operations (export, clear all)

• Upgraded user indicators

• Real-time user count

• Responsive table design

Technologies: JavaScript (loadDashboard, renderUserTable), API: GET /api/users

4. Breach Time Calculator

File: breach.html

Location: pages/breach.html

URL: `http://localhost:8000/pages/breach.html`

Size: ~280 lines

Educational tool calculating password cracking time across different hardware.

Key Features:

• GPU-based cracking time estimates

• Multiple GPU models (RTX 4090, RTX 3090, GTX 1080)

• Algorithm comparison (MD5 vs Argon2)

• Password complexity analysis

• Keyspace calculation

• Visual strength meter

• Real-world cracking scenarios

• Educational recommendations

Technologies: JavaScript (calculateBreachTime), API: POST /api/breach-time

5. Hash Tools

File: hash-tools.html

Location: pages/hash-tools.html

URL: `http://localhost:8000/pages/hash-tools.html`

Size: ~200 lines

Multi-purpose hashing utility for generating and verifying cryptographic hashes.

Key Features:

• Generate hashes: MD5, SHA-1, SHA-256

• Real-time hash generation

• Hash comparison tool

• Verify hash against plaintext

• Copy hash to clipboard

• Batch hashing capability

• Hash length display

• Educational hash explanations

Technologies: CryptoJS library, JavaScript

6. Security Testing Lab

File: security-testing.html

Location: pages/security-testing.html

URL: `http://localhost:8000/pages/security-testing.html`

Size: ~220 lines

Interactive security vulnerability demonstration platform.

Key Features:

• SQL Injection demonstrations

• XSS (Cross-Site Scripting) tests

• CSRF token validation

• Input sanitization examples

• Safe vs unsafe code comparisons

• Educational vulnerability explanations

• Interactive exploit examples

• Mitigation strategies

Technologies: JavaScript, educational demonstrations

7. Security Guide

File: security-guide.html

Location: pages/security-guide.html

URL: `http://localhost:8000/pages/security-guide.html`

Size: ~350 lines

Comprehensive educational resource for password security best practices.

Key Features:

• Password creation guidelines

• Algorithm comparison charts

• Security best practices

• Common vulnerabilities explained

• Code examples for each algorithm

• Industry standards (NIST, OWASP)

• Migration strategies

• Real-world case studies

Technologies: HTML5, educational content

8. All Features Overview

File: all-features.html

Location: pages/all-features.html

URL: `http://localhost:8000/pages/all-features.html`

Size: ~370 lines

Landing page showcasing all platform features with quick navigation.

Key Features:

• Feature cards grid layout

• Platform statistics

• Quick navigation to all features

• Feature descriptions

• Visual feature icons

• Responsive card design

• Category organization

• Search functionality

Technologies: HTML5, CSS Grid, Bootstrap 5

D. Database Structure

SQLite Database Schema

Location: `backend/instance/auth_security_lab.db`

Table	Columns	Purpose
users	id, username, algorithm, hash, salt, timestamp, upgraded	Store user accounts and hashed passwords
login_attempts	id, username, timestamp, success, ip_address	Track login attempts for rate limiting
password_history	id, user_id, hash, timestamp	Prevent password reuse (last 5)

E. Encryption & Hashing Algorithms

Algorithm	Security	Speed	Use Case	Salt
MD5	■■ Broken	Very Fast	Legacy demonstration only	Optional
SHA-1	■■ Weak	Fast	Educational purposes	Optional
BCrypt	■ Secure	Slow (tunable)	Good for passwords	Built-in
Argon2id	■ Best	Slow (tunable)	OWASP recommended	Built-in

F. Complete File Structure

```
Computer-Security/
■■■ index.html (Login page - 482 lines)
■■■ pages/ (Feature pages)
■   ■■■ register.html (User registration)
■   ■■■ dashboard.html (Admin dashboard)
■   ■■■ breach.html (Breach calculator)
■   ■■■ hash-tools.html (Hash utilities)
■   ■■■ security-testing.html (Vulnerability demos)
■   ■■■ security-guide.html (Educational content)
■   ■■■ all-features.html (Feature overview)
■■■ assets/
■   ■■■ css/
■   ■   ■■■ style.css (38KB - main styles)
■   ■   ■■■ nav-styles.css (Navigation styles)
■   ■■■ js/
■       ■■■ script.js (48KB - core functions)
■       ■■■ api-client.js (4.5KB - API integration)
■■■ backend/
■   ■■■ app.py (811 lines - Flask API)
■   ■■■ models.py (Database models)
■   ■■■ requirements.txt (Dependencies)
■   ■■■ instance/
■       ■■■ auth_security_lab.db (SQLite)
■■■ docs/ (Documentation)
■■■ scripts/ (Utility scripts)
■■■ START_APP.bat (One-click startup)
```

H. How It Works - Complete Flow

1. User Opens Browser: Types http://localhost:8000/index.html

2. Frontend Server Response: HTTP server (port 8000) serves index.html

3. Browser Loads Assets: Downloads style.css, script.js, api-client.js from /assets/

4. JavaScript Initialization: api-client.js checks backend availability

5. Backend Connection Test: Fetch request to http://127.0.0.1:5000/api/test

6a. Backend Available: All operations use Flask API + database

6b. Backend Unavailable: Falls back to LocalStorage (client-side only)

7. User Registration: POST /api/register with username, password, algorithm

8. Password Hashing: Backend hashes password with selected algorithm + salt

9. Database Storage: User record saved to SQLite with hash and metadata

10. Login Attempt: POST /api/login with credentials

11. Hash Verification: Backend compares submitted password hash with stored hash

12. Session Creation: JWT token or session cookie issued on success

13. Dashboard Access: Authenticated user sees all users and statistics

14. Feature Usage: User explores breach calculator, hash tools, etc.

R. REST API Endpoints - Complete Reference

Method	Endpoint	Purpose	Request Body
GET	/api/test	Health check	None
GET	/api/health	Server status	None
POST	/api/register	Create user	username, password, algorithm
POST	/api/login	Authenticate	username, password
GET	/api/users	Get all users	None
DELETE	/api/user/<username>	Delete user	None
POST	/api/hash	Generate hash	password, algorithm
POST	/api/verify	Verify password	password, hash, algorithm
POST	/api/breach-time	Calculate breach	password, algorithm
POST	/api/migrate	Upgrade hash	username, password
GET	/api/export/hashcat	Export hashes	None
DELETE	/api/clear-all	Clear database	None
GET	/api/stats	Get statistics	None

Y. Your Next Steps - Getting Started

Step 1: Start Servers: Run START_APP.bat or manually start backend and frontend

Step 2: Open Application: Navigate to http://localhost:8000/index.html

Step 3: Register User: Go to Register page and create account with Argon2

Step 4: Test Login: Return to login page and authenticate

Step 5: Explore Dashboard: View user statistics and management

Step 6: Try Breach Calculator: Test password strength estimation

Step 7: Use Hash Tools: Generate and compare hashes

Step 8: Read Security Guide: Learn best practices

Step 9: Test Vulnerabilities: Explore security testing lab

Step 10: Customize: Modify code to add your own features

Quick Start Commands

```
cd d:\Computer-Security
START_APP.bat
```

Z. Zero to Hero - Complete Learning Path

Level 1: Beginner (Week 1)

✓ Understand the difference between hashing and encryption

✓ Learn why MD5 and SHA-1 are insecure

✓ Register users with different algorithms

✓ Compare hash outputs in the dashboard

✓ Use the breach time calculator

Level 2: Intermediate (Week 2-3)

✓ Explore the Flask backend code (app.py)

✓ Understand SQLAlchemy ORM and database models

✓ Test API endpoints using Postman or curl

✓ Modify cost factors (BCrypt rounds, Argon2 memory)

✓ Export database for Hashcat testing

Level 3: Advanced (Week 4+)

✓ Implement additional hash algorithms (PBKDF2, scrypt)

✓ Add JWT token authentication

✓ Create custom rate limiting logic

✓ Deploy to cloud (Heroku, Railway, AWS)

✓ Implement password breach detection API

✓ Add 2FA (Two-Factor Authentication)

✓ Create comprehensive unit tests

✓ Optimize database queries and indexing

Congratulations!

You now have complete documentation of the Advanced Authentication Security Lab. This platform demonstrates real-world security concepts through hands-on implementation. Use it to learn, teach, and experiment with password security safely.

Remember: This is an educational platform. Always follow security best practices in production applications. Use Argon2id for new applications, implement proper rate limiting, and never store passwords in plain text.