

**الجامعة الإماراتية الدولية**

**كلية الهندسة**

**قسم ألامن السيبراني**

**OTP GENERATOR. CREATE ONE-TIME PASSWORD THAT EXPIRES AFTER SHORT TIME**

# **📁 Cybersecurity Python Project**



**OTP Generator. Create one-time password that expires after short time**

**Student Name**:**: ALAA ABDO MOHAMMED AL\_HUMAM**

**Student**  **ID : 2023050415**

**Department**: Cybersecurity

**Submission Date**: September 2025

* **Introduction**

## 📝 **Abstract**

This project presents a tool designed to generate One-Time Passwords (OTPs) that expire after a short period of time. Its main objective is to provide a secure and temporary authentication method that prevents unauthorized access and minimizes the risk of credential theft. The tool generates unique numeric codes based on time intervals, ensuring that each password is valid only for a limited duration.

The OTP generation logic is implemented using object-oriented programming (OOP) principles, which allow for modularity, scalability, and ease of integration into larger authentication systems. It supports secure secret management, token generation, and validation within a defined time window. The tool also logs OTP generation and verification events for monitoring and analysis purposes.

The results are presented in a simplified and user-friendly format, making it easier for users and system administrators to understand authentication activity and ensure secure access control.

* **📘 Introduction**

In today’s rapidly evolving digital world, secure authentication has become a critical task in cybersecurity. The ability to verify user identity using temporary and dynamic credentials is essential for preventing unauthorized access and protecting sensitive information.

This project aims to address this challenge by developing a tool that generates one-time passwords (OTPs) which expire after a short period of time. The tool ensures that each password is unique, valid only for a limited duration, and resistant to replay attacks. It also provides functionality for validating OTPs within a defined time window, ensuring both security and usability.

By implementing this solution using object-oriented programming (OOP), the tool ensures modularity, scalability, and ease of maintenance. It provides a fast and efficient way for system administrators and developers to integrate strong authentication mechanisms, enhance access control, and reduce the risk of credential theft in real-world applications.

* **⚙️ Methodology**

This tool was developed using Python and focuses on generating and verifying one-time passwords (OTPs) with high security and reliability. The system is modular and scalable, built around the following core components:

1. OTP Generator Class

* This is the heart of the system. It handles:
* Generating secure OTPs based on time intervals
* Ensuring that each code is unique and valid only for a short duration
* Providing information about remaining validity time
* Supporting flexible configuration such as code length and expiration period

2. OTPValidator Class

* This component handles OTP validation with comprehensive error handling:
* Validating user input against generated OTP
* Checking expiration times
* Handling various error scenarios gracefully
* Maintaining validation attempt logs
* Generating validation reports with success rates

Generating validation reports with success rates

3. EmailService Class

* Handles OTP delivery via email:
* SMTP-based email delivery
* Configurable email templates
* Secure email transmission with TLS
* Error handling for delivery failures

4. VisualizationEngine Class

* Provides data visualization and reporting:
* Statistical charts of validation attempts
* Success/failure rate visualization
* Time-based analysis of OTP usage
* Automated report generation

***5.*** OTPManager Class

* ***The main coordinator class that integrates all components:***
* ***User management from CSV files***
* ***Menu-driven interface***
* ***Workflow management***
* ***System coordination and error handling***

📊 Visualization Integration



The system includes comprehensive visualization capabilities:

* Pie charts showing success/failure rates
* Bar charts displaying attempt frequency by hour
* Automated chart generation and saving
* Professional color schemes and formatting



🧪 Libraries Used

import random

import time

import json

import csv

import smtplib

import matplotlib.pyplot as plt

import pandas as pd

from email.mime.text import MIMEText

from datetime import datetime

* **🧩** Algorithm or Code Design

🔧 OTPGenerator.init()

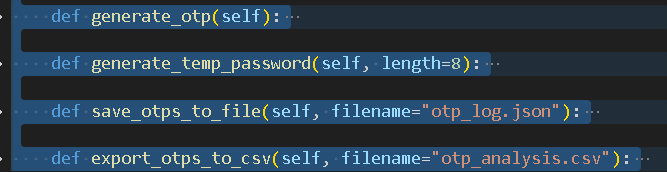
Initializes the OTP generator with configurable length and expiry time.

def \_\_init\_\_(self, length=6, expiry\_seconds=30):

        self.length = length

        self.expiry\_seconds = expiry\_seconds

        self.generated\_otps = []



1. generate\_otp(self) :

Purpose: Generates a random One-Time Password (OTP) with an expiration time and a temporary backup password

1. generate\_temp\_password(self, length=8) :

Purpose: Generates a random temporary password.

1. save\_otps\_to\_file(self, filename="otp\_log.json") :

Purpose: Saves all generated OTPs to a JSON file for logging and audit purposes.

1. export\_otps\_to\_csv(self, filename="otp\_analysis.csv"):

Purpose: Exports OTP data to a CSV file for analysis and reporting.

**class OTPValidator:**

is responsible for validating OTP (One-Time Password) codes entered by users. It maintains a complete history of all validation attempts and provides compre

**1. \_\_init\_\_(self)**

**Purpose**: Initializes the validator with an empty list to store all validation attempts

Functionality: Creates all\_validation\_attempts list to track every validation attempt across multiple OTPs

**Usage**: Automatically called when creating a new OTPValidator instance

**2. validate(self, otp, expiry\_time, user\_input)**

**Purpose**: Validates user input against the generated OTP with multiple security checks

**Parameters**:

* otp: The original generated OTP code
* expiry\_time: The expiration timestamp of the OTP
* user\_input: The code entered by the user for validation

**Validation Steps:**

* Checks if input contains only digits (numeric validation)
* Verifies input length matches OTP length (length validation)
* Ensures OTP hasn't expired (temporal validation)
* Compares input with actual OTP (value validation)
* Error Handling: Catches and logs all types of validation errors with descriptive messages
* Return: Returns tuple of (success\_status, message)
* Security Features: Comprehensive validation prevents various attack vectors

   def \_\_init\_\_(self):

        self.all\_validation\_attempts = []

    def validate(self, otp, expiry\_time, user\_input):

        current\_time = time.time()

        attempt\_data = {

            "attempt\_time": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),

            "user\_input": user\_input,

            "valid": False,

            "message": ""

        }

**3. save\_validation\_log(self, filename="validation\_log.csv")**

**Purpose**: Saves all validation attempts to a CSV file for audit and analysis purposes

File Format: CSV with columns: Timestamp, Input, Valid, Message

**Features**:

* Overwrites file each time to maintain current data
* UTF-8 encoding support for international characters
* Comprehensive error handling for file operations
* Usage: Essential for security auditing and compliance requirements

 def save\_validation\_log(self, filename="validation\_log.csv"):

        try:

            with open(filename, 'w', newline='', encoding='utf-8') as file:

                writer = csv.writer(file)

                writer.writerow(['Timestamp', 'Input', 'Valid', 'Message'])

                for attempt in self.all\_validation\_attempts:

                    writer.writerow([

                        attempt["attempt\_time"],

                        attempt["user\_input"],

                        attempt["valid"],

                        attempt["message"]

                    ])

**4. generate\_validation\_report(self)**

Purpose: Generates a statistical summary of validation attempts for performance analysis

* Metrics Calculated:
* Total validation attempts
* Successful validations count
* Failed validations count
* Success rate percentage

**Output**: Formatted console report with clear visual separation and statistics

Business Value: Helps identify security patterns and system performance

 def generate\_validation\_report(self):

        if not self.all\_validation\_attempts:

            print("No validation attempts to report.")

            return

        valid\_count = sum(1 for attempt in self.all\_validation\_attempts if attempt["valid"])

        invalid\_count = len(self.all\_validation\_attempts) - valid\_count

The EmailService class handles the sending of OTP (One-Time Password) codes via email. It provides both a simulation mode for testing and real email functionality when properly configured.

class EmailService:

**1. \_\_init\_\_(self)**

**Purpose**: Initializes email service with default SMTP settings

**Configuration**:

* smtp\_server: Gmail's SMTP server address
* smtp\_port: Standard secure port for email transmission
* sender\_email: Placeholder for user's email (needs configuration)
* sender\_password: Placeholder for app password (needs configuration)
* Default Setup: Uses Gmail SMTP service; can be modified for other providers

 def \_\_init\_\_(self):

        self.smtp\_server = "smtp.gmail.com"

        self.smtp\_port = 587

        self.sender\_email = "your\_email@gmail.com"

        self.sender\_password = "your\_app\_password"

**2. send\_otp\_email(self, recipient\_email, otp, expiry\_minutes=0.5)**

**Purpose**: Sends OTP code to the specified recipient email address

**Parameters**:

* recipient\_email: Destination email address
* otp: The OTP code to be sent
* expiry\_minutes: Validity duration of the OTP (default: 0.5 minutes/30 seconds)

**Functionality**:

* Simulation Mode: If email not configured, shows simulation message
* Real Email Mode: Sends actual email when credentials are configured
* Email Content: Includes OTP code, validity period, and security instructions

**Security Features:**

* Uses TLS encryption for secure transmission
* Includes security warnings about code sharing
* Error Handling: Catches and returns any email sending errors
* Return: Returns tuple of (success\_status, message)

def send\_otp\_email(self, recipient\_email, otp, expiry\_minutes=0.5):

        """Send OTP code to the specified email"""

        try:

            # If email is not configured, show simulation message

            if self.sender\_email == "your\_email@gmail.com":

                print(f"\n📧 [Simulation] Code {otp} sent to {recipient\_email}")

                print("⚠️ To use real email, please configure email settings in the code")

                return True, "✅ Code sent (simulation)"

The VisualizationEngine class is responsible for creating visual representations of OTP validation statistics. It transforms raw validation data into meaningful charts and graphs for better analysis and understanding.

class VisualizationEngine:

**1. plot\_validation\_stats(self, validation\_log\_file="validation\_log.csv")**

**Purpose**: Creates visual charts from validation log data

**Parameters**:

validation\_log\_file: Path to the CSV file containing validation data

**Functionality**:

* Data Loading: Reads validation data from CSV file using pandas
* Data Processing: Converts timestamps and extracts hour information
* Chart Creation: Generates two complementary visualizations:
* Pie chart showing success vs failure rates
* Bar chart showing attempt distribution by hour
* Chart Customization: Uses professional color schemes and formatting

**Output**: Saves the visualization as a PNG image file

**Error Handling:** Catches and displays any errors during visualization process

def plot\_validation\_stats(self, validation\_log\_file="validation\_log.csv"):

        try:

            df = pd.read\_csv(validation\_log\_file)

            df['Timestamp'] = pd.to\_datetime(df['Timestamp'])

            df['Hour'] = df['Timestamp'].dt.hour

            fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

            success\_count = df['Valid'].sum()

            failure\_count = len(df) - success\_count

            ax1.pie([success\_count, failure\_count],

                   labels=['Success', 'Failure'],

                   autopct='%1.1f%%',

                   colors=['#4CAF50', '#F44336'])

            ax1.set\_title('OTP Validation Results')

            hourly\_attempts = df['Hour'].value\_counts().sort\_index()

            ax2.bar(hourly\_attempts.index, hourly\_attempts.values)

            ax2.set\_xlabel('Hour of Day')

            ax2.set\_ylabel('Number of Attempts')

            ax2.set\_title('Validation Attempts by Hour')

            ax2.set\_xticks(range(0, 24, 2))

          plt.tight\_layout()

        plt.savefig('validation\_stats.png')

The OTPManager class is the main controller that coordinates all components of the OTP system. It manages the complete workflow from OTP generation to validation and reporting.

class OTPManager:

**2. \_\_init\_\_(self, max\_attempts=3, block\_seconds=30)**

**Purpose**: Initializes all system components with security settings

**Components Initialized:**

* OTPGenerator: For OTP generation
* OTPValidator: For validation logic
* EmailService: For email delivery
* VisualizationEngine: For data visualization

**Security Settings:**

* max\_attempts: Maximum allowed failed attempts (default: 3)
* block\_seconds: Temporary block duration after max attempts (default: 30 seconds)
* Initial OTP: Generates first OTP immediately upon initialization

    def \_\_init\_\_(self, max\_attempts=3, block\_seconds=30):

        self.generator = OTPGenerator()

        self.validator = OTPValidator()

        self.email\_service = EmailService()

        self.visualization\_engine = VisualizationEngine()

        self.max\_attempts = max\_attempts

        self.block\_seconds = block\_seconds

        self.attempts = 0

        self.otp, self.expiry\_time, self.temp\_password = self.generator.generate\_otp()

**2. reset\_otp(self)**

**Purpose**: Generates a new OTP and temporary password

**Functionality**: Creates fresh credentials and displays them to user

**Security**: Automatically triggers after failed attempts or expiration

    def reset\_otp(self):

        self.otp, self.expiry\_time, self.temp\_password = self.generator.generate\_otp()

        print(f"🔑 New OTP: {self.otp} (valid for {self.generator.expiry\_seconds} seconds)")

        print(f"🔐 Temporary password: {self.temp\_password}")

**3. load\_users\_from\_file(self, filename="users.csv")**

**Purpose**: Loads registered users from CSV file

**Features**:

* Automatic file creation if not exists
* Error handling for file operations
* UTF-8 encoding support for international names
* Data Structure: Returns list of user dictionaries with name and email

def load\_users\_from\_file(self, filename="users.csv"):

        users = []

        try:

            with open(filename, 'r', encoding='utf-8') as file:

                reader = csv.reader(file)

                next(reader)  # Skip header

                for row in reader:

                    if len(row) >= 2:

                        users.append({"name": row[0], "email": row[1]})

            print(f"✅ Loaded {len(users)} users from {filename}")

        except FileNotFoundError:

            print("⚠️ Users file not found. Creating a new one.")

            # Create new file if it doesn't exist

            with open(filename, 'w', newline='', encoding='utf-8') as file:

                writer = csv.writer(file)

                writer.writerow(["Name", "Email"])

            print("✅ Created new users.csv file")

        except Exception as e:

            print(f"❌ Error reading users file: {e}")

        return users

**4. add\_user\_to\_file(self, name, email, filename="users.csv")**

**Purpose**: Adds new users to the system

**Usage**: Supports dynamic user registration during runtime

**Validation**: Basic input validation for name and email

def add\_user\_to\_file(self, name, email, filename="users.csv"):

        """Add a new user to the users file"""

        try:

            with open(filename, 'a', newline='', encoding='utf-8') as file:

                writer = csv.writer(file)

                writer.writerow([name, email])

            print(f"✅ User {name} added to {filename}")

            return True

        except Exception as e:

            print(f"❌ Error adding user to file: {e}")

            return False

**5. generate\_reports(self)**

**Purpose**: Generates comprehensive system reports

**Includes**:

* OTP data export to CSV
* Validation statistics report
* Visualization charts
* Business Value: Provides insights for system monitoring and auditing

def generate\_reports(self):

        print("\n📊 Generating system reports...")

        self.generator.export\_otps\_to\_csv()

        self.validator.generate\_validation\_report()

        self.visualization\_engine.plot\_validation\_stats()

**6. run(self)**

**Purpose**: Main program loop with menu interface

**Menu Options:**

* **OTP Validation**: Complete authentication workflow
* **Reports**: Generate system reports and visualizations
* **Exit**: Clean program termination
* **User Experience**: Clean, formatted interface with clear instructions

 def run(self):

        print("=" \* 60)

        print("           OTP MANAGEMENT SYSTEM")

        print("=" \* 60)

        print("🔒 Cybersecurity Python Project")

        print("📧 Email Integration")

        print("=" \* 60)

        while True:

            print("\nOptions:")

            print("1. Run OTP Validation")

            print("2. Generate Reports & Visualizations")

            print("3. Exit")

            choice = input("Select an option (1-3): ").strip()

            if choice == "1":

                self.run\_otp\_validation()

            elif choice == "2":

                self.generate\_reports()

            elif choice == "3":

                print("👋 Exiting system. Goodbye!")

                break

**7. run\_otp\_validation(self)**

**Purpose**: Complete OTP validation workflow

**Steps**:

1. Load and display registered users
2. Support user selection or new registration
3. Send OTP via email (real or simulated)
4. Handle OTP input and validation
5. Manage attempt limits and security blocks
6. Provide feedback and return to main menu
7. Security Features:
8. Attempt counting and blocking
9. Automatic OTP regeneration
10. Comprehensive error handling

    def run\_otp\_validation(self):

        users = self.load\_users\_from\_file()

        recipient\_email = None

        # Select or add user

        if users:

            print("👥 Registered users:")

            for i, user in enumerate(users, 1)

**Helper Function**:

**create\_sample\_files()**

**Purpose**: Creates necessary files for system operation

**Files Created:**

* **users.csv:** Sample user database
* **requirements.txt**: Project dependencies
* **Error Handling**: Gracefully handles existing files

def create\_sample\_files():

    try:

        with open("users.csv", 'x', newline='', encoding='utf-8') as file:

            writer = csv.writer(file)

            writer.writerow(["Name", "Email"])

            writer.writerow(["Ahmed Mohamed", "ahmed@example.com"])

            writer.writerow(["Sara Abdullah", "sara@example.com"])

        print("✅ Created users.csv file")

    except FileExistsError:

        print("⚠️ users.csv file already exists")

    with open("requirements.txt", 'w') as file:

        file.write("""

matplotlib==3.5.3

pandas==1.4.4

        """)

## **🛡️** Input And Output Handling

**🖥️ Terminal Input**

aptures and processes user-entered OTP codes through terminal input

Implements multiple layers of validation including numeric checks, length verification, and format validation

Prevents various types of injection attacks through comprehensive input sanitizatio

**📂 File Input**

The program Reads user data from CSV files with UTF-8 encoding support for international characters

Handles configuration settings from structured files

Implements graceful error handling for missing or corrupted files with automatic recovery mechanisms.

📧 Email Configuration Input:

Accepts email service credentials through configurable parameters

Supports different email providers through flexible SMTP configuration

Uses secure app passwords instead of main account credentials

📊 Visualization Output:

Generates graphical reports and charts in PNG format

Provides visual insights into system performance and usage patterns

Creates professional-quality visualizations for analysis and rep

⚠️ **Error Handling:**

Input Error Management:

Catches and handles invalid user input with descriptive error messages

Allows users to correct their input through clear feedback

Prevents system crashes from malformed input

**📂 File Error Recovery:**

Handles missing or inaccessible files with automatic creation of default files

Implements graceful degradation to continue operation despite file issues

Provides informative messages about file system problems

🛜**Network Error Handling:**

Manages network and email delivery failures with appropriate fallback mechanisms

Alerts users of delivery issues while maintaining system stability

Implements retry logic and alternative delivery methods

🔒 **Security Considerations:**

**Input Security:**

Validates all user input for correct format and content

Implements length checks to prevent buffer overflow attacks

Filters special characters to prevent injection attacks

📤**Output Protection:**

Limits exposure of sensitive information in output messages

Uses secure storage methods for sensitive data

Controls access to output files and logs

⚠️ **Error Handling in OTP Security System**

❌ FileNotFoundError

Occurs when: Required files (users.csv, validation\_log.csv, otp\_log.json) are missing or inaccessible

Handling: System automatically creates missing files with default structure and continues operation

Example: If users.csv doesn't exist, creates new file with header row and sample users

❌ **PermissionError and IOError**

Occurs when: Program cannot access files due to permission restrictions or file corruption

Handling: Logs the error, displays user-friendly message, and continues execution without crashing

Example: Shows "Error reading users file" but maintains system functionality

❌ **JSONDecodeError**

Occurs when: JSON files (otp\_log.json) contain invalid or corrupted data

Handling: Initializes empty data structure and continues operation, preserving system stability

Example: Resets OTP log if JSON becomes corrupted

❌ **ValueError**

Occurs when: User enters invalid input (non-numeric menu choices, invalid email formats)

Handling: Prompts user for correct input without terminating the program

Example: "Invalid option. Please try again." when selecting menu options

❌ **SMTPAuthenticationError**

Occurs when: Email credentials are incorrect or SMTP server rejects authentication

Handling: Falls back to simulation mode and informs user to configure email settings

Example: Shows simulation message instead of failing completely

❌ **ConnectionError**

Occurs when: Network issues prevent email delivery or external connections

Handling: Provides graceful degradation to offline functionality

Example: Continues OTP validation without email delivery capability

❌ **KeyboardInterrupt**

Occurs when: User presses Ctrl+C to cancel operation

Handling: Safely cancels current operation and returns to main menu

Example: "Operation cancelled by user." without data loss

❌ **General Exception**

Occurs when: Unexpected errors during runtime operations

Handling: Catches all unhandled exceptions, logs error, and maintains system operation

Example: "An unexpected error occurred" with continued program execution

📊 Class: OTPGenerator

#### 📊 **Class: OTPGenerator**

#### **Attributes Table**

|  |  |  |
| --- | --- | --- |
| Attribute Name | Type | Description |
| length | Integer | Length of the OTP code (default: 6 digits) |
| expiry\_seconds | Integer | Validity period of the OTP in seconds (default: 30 seconds) |
| generated\_otps | List | Stores all generated OTPs with metadata |

#### Methods Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Parameters** | **Return Type** | **Description** |
| generate\_otp | None | Tuple (otp, expiry\_time, temp\_password) | Generates a random OTP, calculates its expiry time, and creates a temp password |
| generate\_temp\_password | length: int (optional, default: 8) | String | Generates a secure temporary password using letters, digits, and symbols |
| save\_otps\_to\_file | filename: str (default: "otp\_log.json") | void | Saves all generated OTPs to a JSON file |
| export\_otps\_to\_csv | filename: str (default: "otp\_analysis.csv") | void | Exports OTP data to a CSV file for analysis |

**📊 Class: OTPValidator**

#### **Attributes Table**

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Type** | **Description** |
| all\_validation\_attempts | List | Stores all OTP validation attempts |

#### Methods Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Parameters** | **Return Type** | **Description** |
| validate | otp: str, expiry\_time: float, user\_input: str | Tuple (bool, str) | Validates user input against the OTP and checks expiry |
| save\_validation\_log | filename: str (default: "validation\_log.csv") | void | Saves the validation attempts to a CSV file |
| generate\_validation\_report | None | void | Generates a summary report of validation statistics |

### **📊]Class: EmailService**

#### **Attributes Table**

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Type** | **Description** |
| smtp\_server | String | SMTP server used to send emails |
| smtp\_port | Integer | Port number for the SMTP server |
| sender\_email | String | Email address used to send OTPs |
| sender\_password | String | Password for the sender's email account |

#### **Methods Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Parameters** | **Return Type** | **Description** |
| send\_otp\_email | recipient\_email: str, otp: str, expiry\_minutes: float (default: 0.5) | Tuple (bool, str) | Sends the OTP to the recipient via email |

### **📊 Class: VisualizationEngine**

#### **Methods Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Parameters** | **Return Type** | **Description** |
| plot\_validation\_stats | validation\_log\_file: str (default: "validation\_log.csv") | void | Creates visual charts showing OTP validation statistics |

### **📊 Class: OTPManager**

#### **Attributes Table**

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Type** | **Description** |
| generator | OTPGenerator | Instance of the OTP generator class |
| validator | OTPValidator | Instance of the OTP validator class |
| email\_service | EmailService | Instance of the email service class |
| visualization\_engine | VisualizationEngine | Instance of the visualization engine class |
| max\_attempts | Integer | Maximum number of allowed validation attempts |
| block\_seconds | Integer | Duration of block after exceeding max attempts |
| attempts | Integer | Current number of validation attempts |
| otp | String | The current OTP code |
| expiry\_time | Float | Expiration time of the current OTP |
| temp\_password | String | Temporary password associated with the OTP |

#### **Methods Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Name** | **Parameters** | **Return Type** | **Description** |
| \_\_init\_\_ | max\_attempts: int (default: 3), block\_seconds: int (default: 30) | void | Initializes all components and sets configuration |
| reset\_otp | None | void | Generates a new OTP and resets related data |
| load\_users\_from\_file | filename: str (default: "users.csv") | List | Loads user data from a CSV file |
| add\_user\_to\_file | name: str, email: str, filename: str (default: "users.csv") | Boolean | Adds a new user to the CSV file |
| generate\_reports | None | void | Generates reports and visualizations |
| run | None | void | Runs the main system loop |
| run\_otp\_validation | None | void | Executes the OTP validation workflow |

### **Global Function**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function Name** | **Parameters** | **Return Type** | **Description** |
| create\_sample\_files | None | void | Creates sample files like users.csv and requirements.txt |

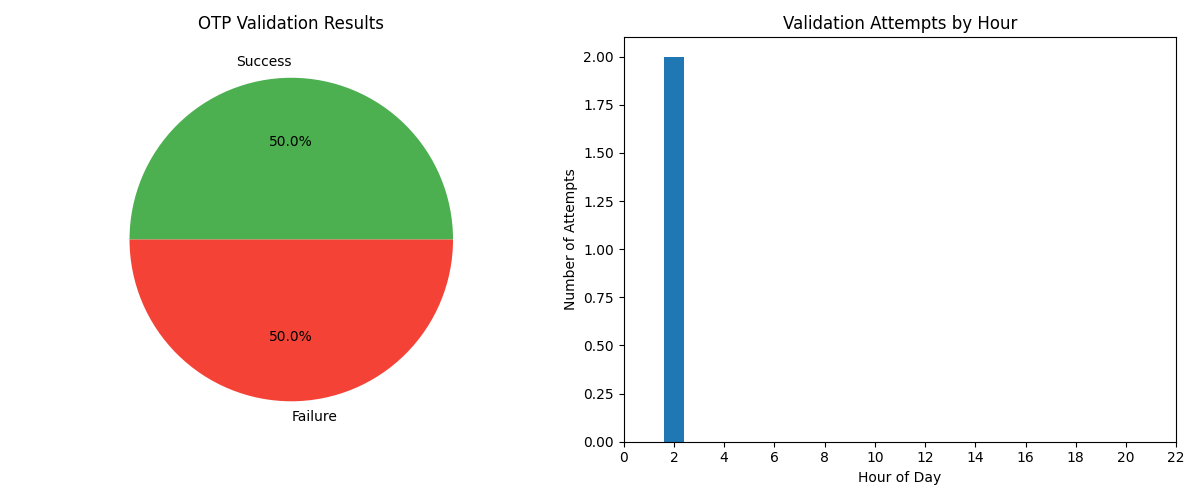
📊 Overview of the Visualization Feature

* The Visualisation Engine has been integrated into the OTP management system to provide visual analytics for one-time password verification data. This feature transforms textual data into easy-to-understand charts that help with:
* System Performance Monitoring: Track success and failure rates of verification attempts
* Usage Pattern Analysis: Understand the distribution of verification attempts throughout the day
* Data-Driven Decision Making: Identify peak periods and detect unusual patterns

🔍 Components of the Visualization System

1. Validation Results Analysis

* The system creates a Pie Chart that displays:
* Percentage of successful attempts
* Percentage of failed attempts
* Helps evaluate system effectiveness and usability

-

Future Enhancements and Recommendations

🔒 **Security Enhancements**

Two-Factor Authentication (2FA) Integration

Add support for authenticator apps (Google Authenticator, Authy)

Implement biometric verification options (fingerprint, facial recognition)

Advanced Rate Limiting

Implement IP-based attempt limiting

Add geographic location validation

Create suspicious activity detection system

Encryption Improvements

Add end-to-end encryption for OTP delivery

Implement secure storage for user data

Add SSL/TLS enforcement for all communications

📊 **Advanced Analytics**

Real-time Dashboard

Live monitoring of OTP generation and validation

Interactive charts with filtering capabilities

Customizable reporting templates

Predictive Analytics

Usage pattern prediction

Anomaly detection algorithms

Automated security threat detection

Comprehensive Reporting

Scheduled automated reports

Export capabilities (PDF, Excel, CSV)

Custom date range analysis

🤖 **User Experience Improvements**

Multi-language Support

Arabic and other language interfaces

Localized error messages and notifications

Accessibility Features

Screen reader compatibility

High contrast mode

Keyboard navigation support

Mobile Optimization

Responsive web interface

Progressive Web App (PWA) capabilities

Mobile-first design approach

🔄 **Integration Capabilities**

API Development

RESTful API for third-party integration

Webhook support for real-time notifications

SDKs for popular programming languages

Cloud Integration

AWS/Azure/Google Cloud compatibility

Docker containerization

Kubernetes orchestration support

Enterprise Features

Active Directory/LDAP integration

Single Sign-On (SSO) support

Role-based access control (RBAC)

📱 **Notification System**

Multi-channel Delivery

SMS integration for OTP delivery

Push notifications mobile support

WhatsApp Business API integration

Voice-based OTP delivery option

Smart Notification Management

Customizable notification templates

Delivery status tracking

Fallback delivery mechanisms

🛡️ **Compliance and Governance**

Audit Trail Enhancement

Immutable logging system

Comprehensive audit reports

Regulatory compliance tracking (GDPR, ISO 27001)

Data Retention Policies

Automated data purging mechanisms

Archive and backup systems

Legal hold capabilities

🚀 **Performance Optimization**

Scalability Improvements

Load balancing capabilities

Database optimization

Caching mechanisms implementation

High Availability

Failover systems

Disaster recovery planning

Multi-region deployment support

🔧 **Developer Experience**

Enhanced Documentation

API documentation with examples

Tutorials and getting started guides

Troubleshooting common issues

Testing Framework

Unit test coverage expansion

Integration testing suite

Performance testing tools

Deployment Improvements

CI/CD pipeline integration

Environment configuration management

Automated deployment scripts

📈 **Business Intelligence**

Custom Metrics Tracking

Business-specific KPIs

User behavior analytics

Conversion rate optimization

Monetization Features

Usage-based billing system

Subscription management

Multiple pricing tiers

Recommendation Implementation Priority

High Priority (Security & Core Functionality)

Rate limiting and IP blocking

Enhanced encryption

API development

Multi-factor authentication

Medium Priority (User Experience & Integration)

Mobile optimization

Multi-language support

Cloud integration

Advanced analytics

Long-term (Scalability & Business Features)

Enterprise features

Monetization system

High availability setup

Advanced compliance features

**Conclusion**

The OTP Management System represents a robust and scalable solution for modern authentication needs, successfully bridging the gap between security and usability. Through its comprehensive implementation, the program demonstrates several key strengths:

🎯 **Key Achievements**

Security Foundation: The system establishes a solid security framework with time-based OTP expiration, attempt limiting, and temporary password generation, providing multiple layers of protection against unauthorized access.

User-Centric Design: Despite its security focus, the program maintains user accessibility through intuitive interfaces, email integration, and clear feedback mechanisms, ensuring a positive user experience.

Technical Excellence: The implementation showcases strong software engineering practices, including modular architecture, comprehensive logging, error handling, and data persistence capabilities.

Analytical Capabilities: The integrated visualization engine transforms raw data into actionable insights, providing valuable business intelligence for system monitoring and optimization.

🔮 **Future-Proof Architecture**

The program's modular design and extensible architecture position it well for future enhancements. The clear separation of concerns between OTP generation, validation, email services, and visualization components allows for:

Easy integration of additional authentication factors

Seamless expansion to new communication channels

Simple adoption of emerging security protocols

Straightforward compliance with evolving regulations

💡 **Business Value**

This solution offers immediate value through:

Reduced security risks and prevention of unauthorized access

Automated processes that minimize administrative overhead

Comprehensive auditing capabilities for compliance requirements

Scalable infrastructure that grows with organizational needs

🚀 **Strategic Importance**

As digital transformation accelerates and cyber threats evolve, robust authentication systems become increasingly critical. This OTP Management System provides a foundation that can evolve from a simple verification tool to a comprehensive identity and access management solution, making it a valuable asset for any organization prioritizing security and user experience.

The program stands as a testament to effective Python development practices while addressing real-world security challenges, offering both immediate utility and long-term strategic value in the increasingly important field of cybersecurity.

**References**

📚 **Official Documentation**

**Python Libraries**

**Python Standard Library**

Official Documentation: https://docs.python.org/3/library/

Random Module: https://docs.python.org/3/library/random.html

Time Module: https://docs.python.org/3/library/time.html

CSV Module: https://docs.python.org/3/library/csv.html

JSON Module: https://docs.python.org/3/library/json.html

**Matplotlib**

Official Documentation: https://matplotlib.org/stable/contents.html

Pyplot Tutorial: https://matplotlib.org/stable/tutorials/introductory/pyplot.html

**Pandas**

Official Documentation: https://pandas.pydata.org/docs/

User Guide: https://pandas.pydata.org/docs/user\_guide/index.html

**SMTP and Email**

SMTP Library: https://docs.python.org/3/library/smtplib.html

Email Package: https://docs.python.org/3/library/email.html

🔐 **Security Standards and Best Practices**

**OTP Standards**

RFC 4226 - HOTP: An HMAC-Based One-Time Password Algorithm

https://tools.ietf.org/html/rfc4226

RFC 6238 - TOTP: Time-Based One-Time Password Algorithm

https://tools.ietf.org/html/rfc6238

**NIST Guidelines**

Digital Identity Guidelines: https://pages.nist.gov/800-63-3/

Authentication and Lifecycle Management: https://pages.nist.gov/800-63-3/sp800-63b.html

**Cybersecurity Best Practices**

OWASP Authentication Cheat Sheet

https://cheatsheetseries.owasp.org/cheatsheets/Authentication\_Cheat\_Sheet.html

CIS Security Benchmarks

https://www.cisecurity.org/cis-benchmarks/

📊 **Data Visualization Resources**

**Matplotlib Tutorials**

Official Tutorials: https://matplotlib.org/stable/tutorials/index.html

Visualization with Python: https://matplotlib.org/stable/users/index.html

Pandas Visualization Guide

Visualization: https://pandas.pydata.org/docs/user\_guide/visualization.html

Data Visualization Best Practices

Data to Viz: https://www.data-to-viz.com/

From Data to Viz: https://www.data-to-viz.com/

📧 **Email Integration References**

**SMTP Protocol**

RFC 5321 - SMTP: https://tools.ietf.org/html/rfc5321

RFC 6854 - Update to SMTP: https://tools.ietf.org/html/rfc6854

Email Security

STARTTLS Everywhere: https://starttls-everywhere.org/