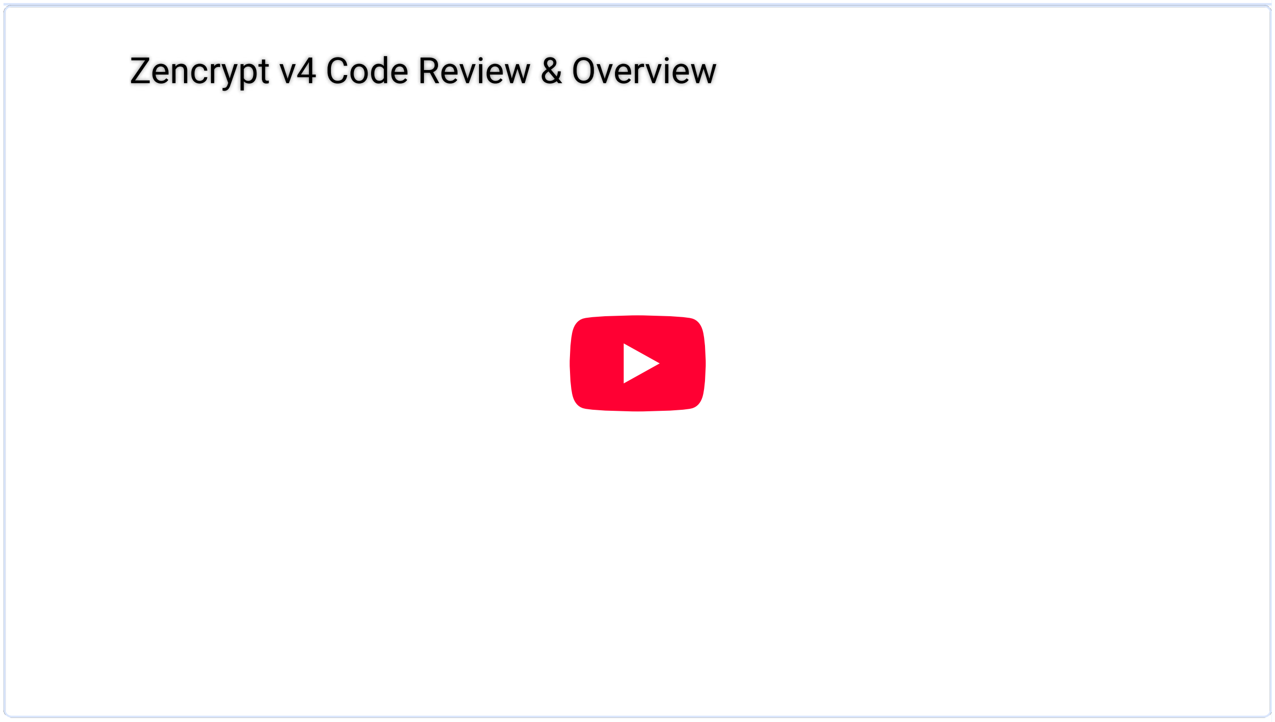
Zencrypt

Web-Application Narratives

Zencrypt Web-App

Zencrypt CLI and Update Plans

Overview and Whitepapers:



Code Review and Cipher Run and Explanations

Welcome to Zencrypts Whitepapers! Here you'll find all the documentation you need to get up and running with the Zencrypt software along with all of its updates and my progress.

Want to go ahead and download and use the Cipher CLI?

Feeling like an eager beaver?
Jump in to the Github and skip all of the technical non-sense

> The ***Zencrypt CLI*** source code currently is hosted on `version 5`

GitHub - ryanshatch/zencrypt at v5
GitHub

> The ***webapp*** source code is hosted on the final branches "`v2.6.6-alpha`" and is also on the "`main`"

GitHub - ryanshatch/zencrypt: Zencrypt is a tool deisgned for cryptographic operations such as hashing, encryption, and decryption. It allows users to generate and validate hashes with salt, encrypt and decrypt text along with file management.
GitHub

Want to Continue to read the Documentation on the Narratives?

Dive a little deeper and start exploring my Zencrypt reference to get an idea of everything that's possible with it, from encrypting text and files to full blown PGP functionalities.

This idea started as a single CLI script and is now a full fledged web application.

Narratives

> Or you can read more about my overall approach from the beginning and see how it adapted and changed over time:

Zencrypt Documentation

Want to Continue to Read More About My Overall Approach For Migrating Zencrypt into a Web Application:

Merging Zencrypt CLI Into A Web-App

Narratives

Merging Zencrypt CLI Into A Web-App

Zencrypt is a cryptographic project designed to enhance data security through encryption and hashing. The application is evolving from a command-line interface (CLI) to a modular, scalable, and user-friendly application featuring GUI and web-service capabilities. This document provides an overview of the project, enhancement plans, and the future roadmap.

Enhancement Plan for Zencrypt CLI

Personal Reflection Initial Enhancement Plans

Zencrypt aims to address modern cryptographic needs by incorporating a modular structure, advanced encryption techniques, and user-friendly interfaces. The enhancement plan focuses on the following key improvements:

Core Enhancements

- **UI/UX Integration:** Add a graphical user interface (GUI) using frameworks like Tkinter or PyQt.
- **Modular Structure:** Transition from a single-file CLI script to a modular architecture.
- **Web-Service Expansion:** Introduce a Flask/Django-based web-service to allow remote encryption/decryption.
- **Best Practices:** Adopt industry standards such as configuration files, logging, and environment variables for secure secret handling.

Software Engineering and Design

Algorithms and Data Structures

- **Advanced Cryptography:** Integrate elliptic-curve cryptography (ECC) and Argon2 hashing.
- **Concurrency:** Optimize large file encryption using multithreading or multiprocessing.
- **Data Handling:** Use sophisticated data structures like queues and Merkle trees to enhance performance.

Database Integration

- **Secure Storage:** Store keys, logs, and user information in a SQL or NoSQL database.
- **Authentication:** Implement user authentication and role-based access controls.
- **Key Management:** Add key rotation and expiry functionality for enhanced security.

Architecture and Workflow

High-Level Flowchart

1. **Startup:** Load configuration and environment variables.
2. **Logging & Initialization:** Set up modular architecture and log handlers.
3. **Interface Selection:** Choose between GUI, web-service, or CLI mode.
4. **Execution:**
    - **GUI Mode:** Launch a Tkinter/PyQt interface for user interaction.
    - **Web-Service Mode:** Run a Flask/Django server for remote operations.
    - **CLI Mode:** Present an updated menu with enhanced features.
5. **Database Operations:** Manage keys, logs, and encrypted data securely.
6. **Shutdown:** Close resources and save logs.

Modular Components

- `config.py`: Handles configuration and environment variables.
- `utils.py`: Contains encryption and hashing algorithms.
- `cli.py`: Provides command-line functionality.
- `webapp.py`: Manages the web-service functionality.

Updating Zencrypt Algorithms and Data Structures:

Database Flow

1. **Configuration Loading:** Retrieve database credentials from secure sources.
2. **Connection:** Establish a secure connection to MySQL, PostgreSQL, or MongoDB.
3. **Authentication:** Validate user credentials and roles.
4. **Operations:** Perform actions like storing keys, logging events, and managing key expiry.
5. **Response Handling:** Return results or errors to the main application.

Updating Zencrypt Databases

Planned Features for Zencrypt v5

- **Scalability:** Built for long-term maintainability with a focus on modular design.
- **Performance:** Optimize large file handling and encryption tasks.
- **Security:** Incorporate advanced cryptographic algorithms and database-level security.
- **User Experience:** Simplify user interaction through a polished GUI and robust web interface.

Skills and Illustrated Outcomes

Professional Assessment

Zencrypt reflects a commitment to secure software development, combining theoretical knowledge with practical implementation. By addressing real-world cybersecurity challenges, Zencrypt highlights skills in:

- **Full-Stack Development:** Expertise in building scalable applications.
- **Cybersecurity:** Proficiency in cryptography, secure data handling, and compliance.
- **Project Management:** Following Agile methodologies to meet client expectations.

Zencrypt's documentation and enhancement roadmap demonstrate a dedication to quality, innovation, and practicality in cybersecurity solutions.

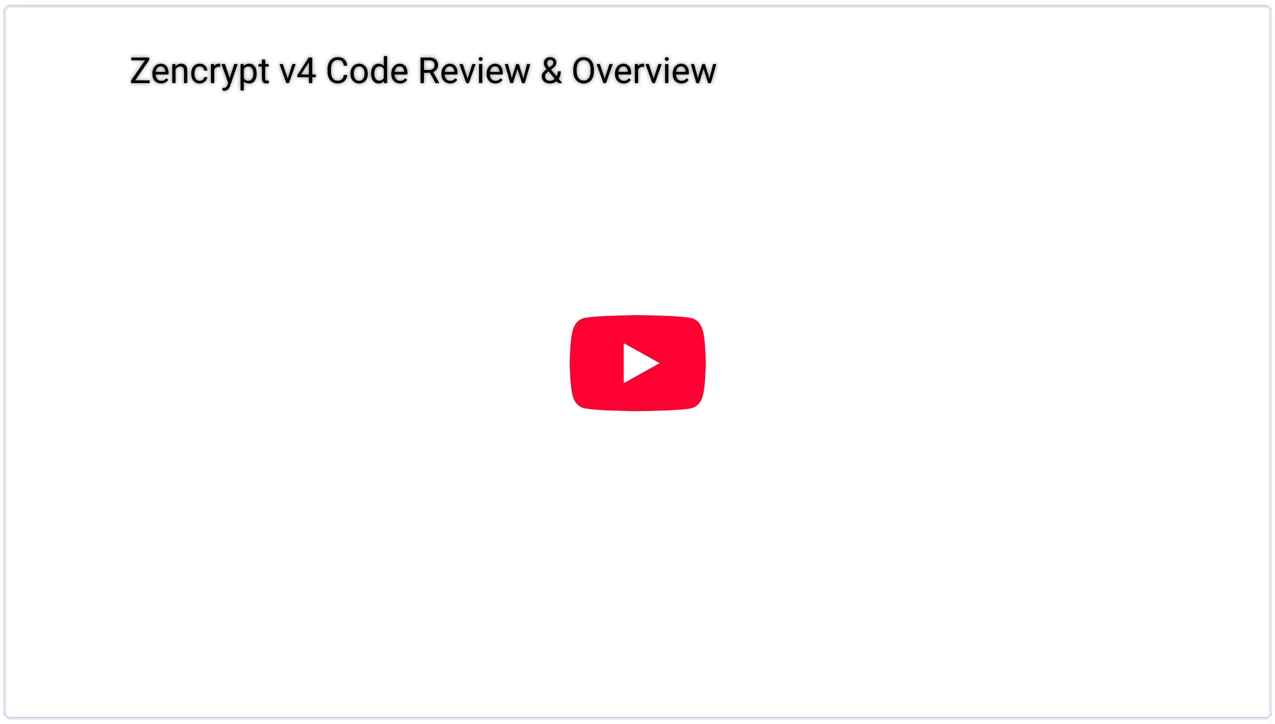
ePortfolio in current state

ERROR: Banana404 NOT FOUND

Code Review: What Is Productive Code?

What Is the Definition of Productive Code?

Code Review:



CLI (Version 4.2)

A good code review is the very foundation of developing solid software. Productive code review helps keep the code clean, scalable, and consistent from the start. With that said, by making it a habit to take the extra time to review my code before merging it into the main branch, I am able to catch bugs early and fix them. This practice helps to optimize the strength of the codebase and keep everything secure. None the less, sticking to consistent coding standards through regular reviews is what helps me make sure that each project turns out well.

To do reviews as effectively as possible, I use checklists, automated tools that catch basic issues automatically, and to keep a steady focus on small and manageable changes. With that said, comments are a big part of this process as they need to be clear and helpful for future development. Feedback should also be kind, on time, and tailored towards the project’s goals. None the less, in my personal experience the best time to review code is right after any changes are made by pushing them to GitHub. This helps catch problems early and fix them before merging them into the main branch.

Recording the Process for Zencrypt v4 and Planning Updates:

Note: The review was tailored towards structure, documentation, variables, and functionality.

To effectively review my Zencrypt CLI, I recorded a screencast using OBS Studio. I picked OBS because it's easy to use and produces great results. During the recording, I ran the program in Visual Studio Code to show how the cipher works and to spot any issues in the code. Once the video was done, I uploaded it to YouTube. This made it easy to share since the review was over 30 minutes long, and I wanted to explain Zencrypt properly.

For the structure, I checked the flow of the menu, how files were organized, and whether the code followed good practices. Since Zencrypt v4 is just one Python script, it was simple to go through everything in detail. For the variables, I made sure the names of the variables were clear and that they didn't create any conflicts. If I found bugs, I handled them and added comments where needed. For Zencrypt v5, I plan to make the code more modular and organized, which means grouping constants better for scalability. As for the documentation, I added clear comments throughout the script to explain the code and summarize its parts. I also used GitBooks to create a markdown repository to track edits and document Zencrypt's transition from v4 to v5.

The screencast helped me review Zencrypt CLI v4 as a whole. By running the code and showing examples of the cipher, I spotted areas where I could improve its functionality, add more detailed comments, and refine the documentation. My goal is to make Zencrypt modular and scalable, which will make it easier to work with as it evolves into v5.

The screen capture that I recorded for Zencrypt CLI v4 was to effectively give an overall view of the project. In the video, I walked through the code step by step, running it to show how the cipher works and pointing out areas I want to improve or change. For me personally, this code review was a good way to think about what I want to add next for the cipher. As I went through it, I found places where I could enhance in terms of functionality, add more detailed comments, and clean up past documentation for v4. With that said, these changes are all aimed at making Zencrypt more modular and scalable, which will improve its quality as it expands into v5, which turns the CLI into a web-app with a clean UI/UX.

Reflection on Consistent Improvement:

None the less, I believe that sticking to practicing good habits can make all the difference for developers and the outcome of the work. For example, a healthy habit to keep consistent in the development process is sticking to effective coding standards. That’s why I always review my code before pushing any changes to the public. This step is crucial to maintain an optimal standard in quality control. This can also be easily detrimental because even a small mistake like accidentally including an API key in the code can have big consequences. Therefore, it’s imperative to actually put in some extra effort and time to review being pushed or merged and make sure that the code is secure, contains no leaks, and meets a high standard.

None the less, in conclusion, consistently reviewing the code, especially before merging into the main branch, helps to show just how important it is to maintain quality and keep things running smoothly.

Software Design and Engineering

Software Design and Engineering

Software Design and EngineeringZencrypt is a Python-based cryptographic application originally developed in August 2022 as a command line interface (CLI) tool. The most recent enhancement was just added on the 21st of January 2025 in order to convert the CLI into a modern web app, while still maintaining the original CLI functionality from Zencrypt v4. This enhancement helps to showcase software engineering principles through implementing the CLI functionality using the Flask web framework and keeping the core cryptographic functionality separate.

Software Development and Enhancement:

The original CLI version provided encryption, hashing, and key management through a text-based interface:

```
def encrypt_text():
    text_to_encrypt = input("\nEnter the text to encrypt: ")
    encrypted_text = cipher_suite.encrypt(text_to_encrypt.encode()).decod
    return encrypted_text
```

The enhanced version in v5 maintains this functionality while adding a web interface through Flask routes:

```
@app.route('/encrypt', methods=['GET', 'POST'])
def encrypt_page():
    if request.method == 'POST':
        text = request.form.get('text', '')
        if text:
            try:
                encrypted = cipher_suite.encrypt(text.encode())
                output = f"Encrypted Text:\n{encrypted.decode()}"
                return render_template_string(APP_TEMPLATE,
                    content=content, output=output)
```

Justification for Enhancement - The enhancement helps to showcase several key software engineering principles:

- **Separation of Concerns:**

Core cryptographic functions were moved to utils.py which allows both interfaces to share the same secure implementations of functions:from cryptography.fernet import Fernetdef initialize_key():if not os.path.exists(KEY_FILE):key = Fernet.generate_key()with open(KEY_FILE, "wb") as key_file:key_file.write(key)

- **Security Considerations:**

The web implementation maintains the same level of security as the CLI v4 while adding new considerations for web-based threats. For example, for secure session handling:app.secret_key = secrets.token_hex(32)

Learning Outcomes and Challenges:

The enhancement process provided helpful learning opportunities for several key areas:

- **Web Security:** Implementing secure web practices while maintaining cryptographic integrity
- **Interface Design:** Creating a solid web UI/UX for complex cryptographic operations
- **Code Organization:** Structuring the project in a scalable and modular format in order to properly maintain a clear separation between the CLI and webapps components.

The main challenge was adapting the already existing v4's CLI based operations to a stateless web-app environment, all done without compromising any of the user's anonymity and security. This development process required careful consideration of how to begin developing the server to later include security methods like temporary sessions and secure handling and maintenance of databases.

Future Improvements for the WebApp / Next enhancements:

- Adding Login and add logging and temporary sessions
- Implementing MongoDB or SQLite
- Utilizing .config, .env, or even a .docker file to be used for constants

References/ Links to my ePortfolio and Zencrypt:

ePortfolio - www.ryanshatch.com

Web Application - www.zencrypt.app

Whitepapers - https://zencrypt.gitbook.io/zencrypt

Enhancement on Algorithms and Data Structure

Artifact Description:

Zencrypt was originally a simple tool that was built for the command line interface, has been merged into a full-fledged web application that uses Flask. This upgrade helped to add a web-based UI/UX to the CLI program and finalize features like salted hashing, symmetric encryption, and securely processing files online. The Zencrypt web app is built with strong foundations in the development of algorithms and data structures in order to optimally process user inputs, files, and sessions.

I selected this artifact to showcase my progress in blending advanced cryptographic methods with sophisticated data management techniques:

- Hashing Algorithm: I upgraded the SHA256 hashing algorithm by adding an optional salting function to make password security more optimal and effectively is a good way to showcase my fluency in cryptographic principles.
- Modular Data Processing: I redesigned the way that the system processes file uploads and encryption operations to facilitate smooth and efficient data flow. This is a good technique that merges with MongoDB and the data that is being processed.

• Session Management: I integrated into the web application JWT-based authentication that uses MongoDB to give the program a more effective way to store and manage user sessions.

These enhancements have helped me be able to develop a formal UI/UX that is user friendly without compromising security. I have been able to successfully merge a CLI script into a web app that uses MongoDB for the back end to work with the ground zero of the of all data parsed through the front end/ client side inputs, uses JWT to handle secure sessions, all while having a modular functionality of a cipher that can handle different types of algorithms to encrypt/ decrypt text and files.

Course Outcomes:

This project gave me the chance to use advanced algorithms that align with the best practices used in cyber security and software development. It also helped me to use clean code practices in order to professionally design and deploy data structures that were optimal for authentication and session integrity.

Enhancement Process Reflection:

Working on Zencrypt v5-A is simply a way to use different secure algorithms in a singular web environment. I dealt with challenges like moving encryption operations from the command line to a web interface without compromising security, for example, managing user sessions and keys. Using MongoDB also helped me learn how to effectively build data structures that balance easy access to data with strong security. Implementing JWT authentication helped me learn more about token processes, like how they're made, validated, and managed. Every change that I made to the web app so far has developed with a balance of ease and security in mind. This project has been able to fit well as a real-world ability to showcase my progression in web development.

Enhancements on its Databases

Artifact Description:

Zencrypt started as a simple CLI cipher tool that I created back in August 2022. Since then, I have been able to transform it into something I'm truly proud of to be able to showcase my skills in computer science. I turned the one script CLI into a Flask web app with SQLite integration that runs on an AWS EC2. The current release, v6.2-alpha shows just how far I have taken the cipher and used different forms of cryptography in order to be able to create a tool that not just I would use, but I tailored it out for everyday use towards for other people with similar niche interests in encryption and anonymity.

Justification and Improvements:

I chose to focus on database improvements since my skills in data management are currently not as fluent as I would like them to be, especially in terms of having the code production ready for an online instance instead of a local instance. The merger from MongoDB to SQLite taught me the value of direct data control.

I moved the entire data layer to SQLite with Flask-SQLAlchemy. This was difficult because the Flask framework by nature had its own set of rules about how it approached secret variables. None the less, I set up a secure key storage system in an EC2 instance under the subdirectory “~/etc/secrets/.” It’s also important to note that Replit, Render, and a lot of other easy hosting platform services had yet their own formal set of rules and approaches to secrets and how they handled variables and specific subdirectories. In the end, for the ability to have the control that I want for zencrypt's webapp, I chose to deploy everything to AWS EC2. The rules are whatever I want to be for my environment variables along with where and how to store user data, without being forced to subscribe to get the functionality from, for example, Renders ability to run ssh/ shell or database files.

- • MIGRATED FROM MONGODB TO SQLITE FOR BETTER CONTROL OVER DATA STRUCTURE AND USER EXPERIENCE.
- IMPLEMENTATION OF SECURE KEY STORAGE IN /ETC/SECRETS/...
- INTEGRATED WITH AWS EC2 FOR MORE CONTROL AND FUNCTIONALITY WITHIN THE DEPLOYMENT AND POST-DEPLOYMENT.

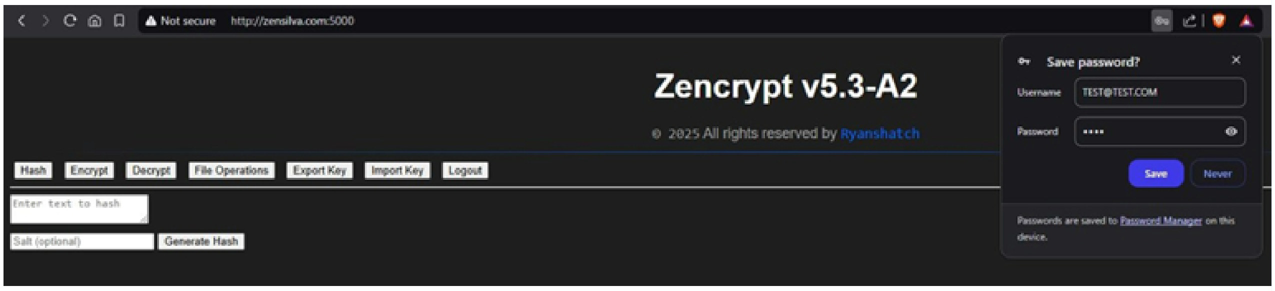
- OPTIMIZED THE DATABASE CONNECTIONS FOR THE PRODUCTION ENVIRONMENT.

Course Outcomes:

The database enhancement exceeded my initial goals. I learned to create solid user authentication, handle encrypted data storage, and set up a production-ready environment on AWS EC2.

Learning and Challenges:

Working through this merge and enhancement, none the less, was absolutely not very easy for me at all, nor was it anything I was fairly used to. Setting up /etc/secrets/ on AWS took some time to get right, and the MongoDB to SQLite migration had its moments. I noticed that managing environment variables in production needs a different approach and more attention to specific details. With that being said, moving to AWS EC2 with SQLite so far has been successful post deployment because the setup is much more effective than the last version on Render, thus giving me more control over the infrastructure while still maintaining security standards.


Not secure http://zensilva.com:5000

Zencrypt v5.3-A2

© 2025 All rights reserved by Ryanshatch

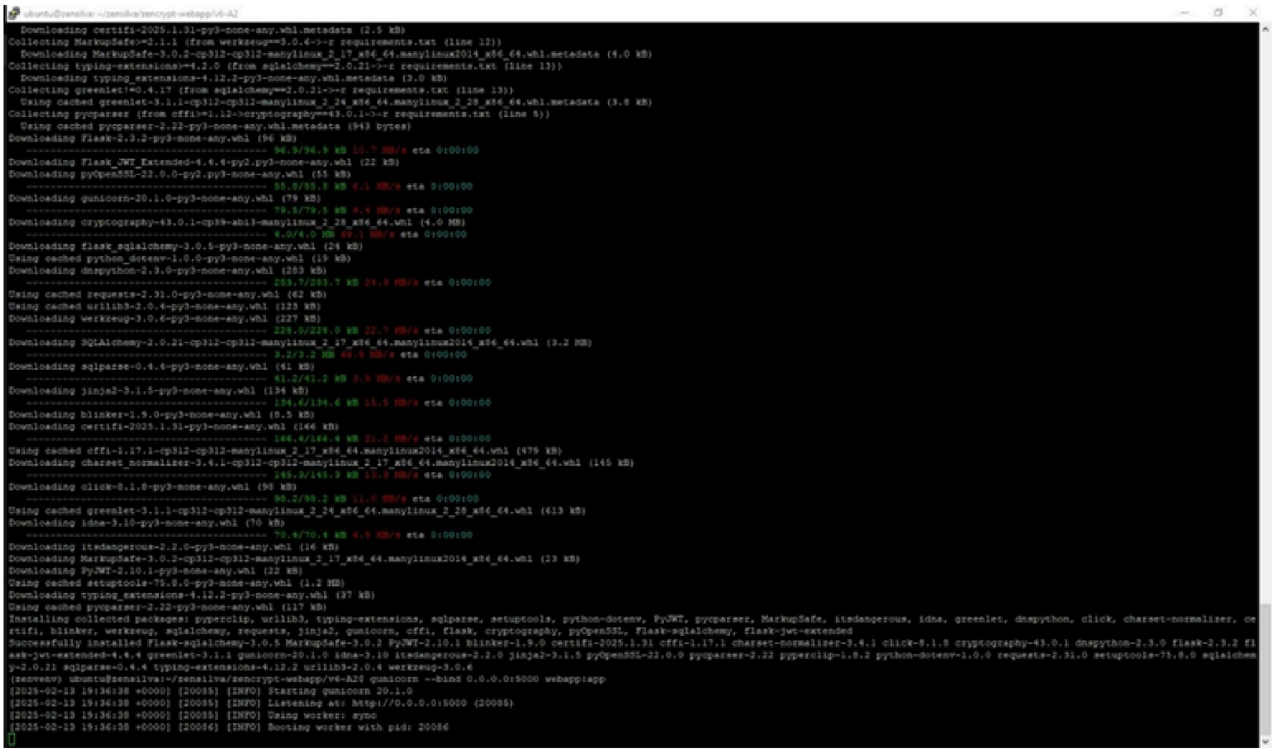
Save password?

Username: TEST@TEST.COM

Password: ••••

Passwords are saved to Password Manager on this device.

Webapp



```
ubuntu@zensilva: ~/zensilva/zencrypt-webapp/v6-A2
  Downloading certifi-2025.1.31-py3-none-any.whl.metadata (2.5 kB)
Collecting MarkupSafe>=2.1.1 (from werkzeug==3.0.6->-r requirements.txt (line 12))
  Downloading MarkupSafe-3.0.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.0 kB)
Collecting typing-extensions>=4.2.0 (from sqlalchemy==2.0.21->-r requirements.txt (line 13))
  Downloading typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 kB)
Collecting greenlet!=0.4.17 (from sqlalchemy==2.0.21->-r requirements.txt (line 13))
  Using cached greenlet-3.1.1-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl.metadata (3.8 kB)
Collecting pycparser (from cffi>=1.12->cryptography==43.0.1->-r requirements.txt (line 5))
  Using cached pycparser-2.22-py3-none-any.whl.metadata (943 bytes)
Downloading flask-2.3.2-py3-none-any.whl (96 kB)
   ---------------------------------------- 96.9/96.9 kB 10.7 MB/s eta 0:00:00
Downloading Flask_JWT_Extended-4.4.4-py2.py3-none-any.whl (22 kB)
Downloading pyOpenSSL-22.0.0-py2.py3-none-any.whl (55 kB)
   ---------------------------------------- 55.8/55.8 kB 4.1 MB/s eta 0:00:00
Downloading gunicorn-20.1.0-py3-none-any.whl (79 kB)
   ---------------------------------------- 79.5/79.5 kB 9.4 MB/s eta 0:00:00
Downloading cryptography-43.0.1-cp39-abi3-manylinux_2_28_x86_64.whl (4.0 MB)
   ---------------------------------------- 4.0/4.0 MB 49.1 MB/s eta 0:00:00
Downloading flask_sqlalchemy-3.0.5-py3-none-any.whl (24 kB)
Using cached python_dotenv-1.0.0-py3-none-any.whl (19 kB)
Downloading dnspython-2.3.0-py3-none-any.whl (283 kB)
   ---------------------------------------- 283.7/283.7 kB 24.3 MB/s eta 0:00:00
Using cached requests-2.31.0-py3-none-any.whl (62 kB)
Using cached urllib3-2.0.4-py3-none-any.whl (123 kB)
Downloading werkzeug-3.0.6-py3-none-any.whl (227 kB)
   ---------------------------------------- 228.0/228.0 kB 22.7 MB/s eta 0:00:00
Downloading SQLAlchemy-2.0.21-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.2 MB)
   ---------------------------------------- 3.2/3.2 MB 48.9 MB/s eta 0:00:00
Downloading sqlparse-0.4.4-py3-none-any.whl (41 kB)
   ---------------------------------------- 41.2/41.2 kB 3.9 MB/s eta 0:00:00
Downloading jinja2-3.1.5-py3-none-any.whl (134 kB)
   ---------------------------------------- 134.6/134.6 kB 15.5 MB/s eta 0:00:00
Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Downloading certifi-2025.1.31-py3-none-any.whl (166 kB)
   ---------------------------------------- 166.4/166.4 kB 21.2 MB/s eta 0:00:00
Using cached cffi-1.17.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (479 kB)
Downloading charset_normalizer-3.4.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (145 kB)
   ---------------------------------------- 145.3/145.3 kB 13.3 MB/s eta 0:00:00
Downloading click-8.1.8-py3-none-any.whl (98 kB)
   ---------------------------------------- 98.2/98.2 kB 11.0 MB/s eta 0:00:00
Using cached greenlet-3.1.1-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (613 kB)
Downloading idna-3.10-py3-none-any.whl (70 kB)
   ---------------------------------------- 70.4/70.4 kB 6.9 MB/s eta 0:00:00
Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Downloading MarkupSafe-3.0.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (23 kB)
Downloading PyJWT-2.10.1-py3-none-any.whl (22 kB)
Using cached setuptools-75.8.0-py3-none-any.whl (1.2 MB)
Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Using cached pycparser-2.22-py3-none-any.whl (117 kB)
Installing collected packages: pyperclip, urllib3, typing-extensions, sqlparse, setuptools, python-dotenv, PyJWT, pycparser, MarkupSafe, itsdangerous, idna, greenlet, dnspython, click, charset-normalizer, ce
rtifi, blinker, werkzeug, sqlalchemy, requests, jinja2, gunicorn, cffi, flask, cryptography, pyOpenSSL, Flask-sqlalchemy, flask-jwt-extended
Successfully installed Flask-sqlalchemy-3.0.5 MarkupSafe-3.0.2 PyJWT-2.10.1 blinker-1.9.0 certifi-2025.1.31 cffi-1.17.1 charset-normalizer-3.4.1 click-8.1.8 cryptography-43.0.1 dnspython-2.3.0 flask-2.3.2 fl
ask-jwt-extended-4.4.4 greenlet-3.1.1 gunicorn-20.1.0 idna-3.10 itsdangerous-2.2.0 jinja2-3.1.5 pyOpenSSL-22.0.0 pycparser-2.22 pyperclip-1.8.2 python-dotenv-1.0.0 requests-2.31.0 setuptools-75.8.0 sqlalchem
y-2.0.21 sqlparse-0.4.4 typing-extensions-4.12.2 urllib3-2.0.4 werkzeug-3.0.6
(zenvenv) ubuntu@zensilva:~/zensilva/zencrypt-webapp/v6-A2$ gunicorn --bind 0.0.0.0:5000 webapp:app
[2025-02-13 19:36:38 +0000] [20085] [INFO] Starting gunicorn 20.1.0
[2025-02-13 19:36:38 +0000] [20085] [INFO] Listening at: http://0.0.0.0:5000 (20085)
[2025-02-13 19:36:38 +0000] [20085] [INFO] Using worker: sync
[2025-02-13 19:36:38 +0000] [20086] [INFO] Booting worker with pid: 20086
```

EC2 Instance

Zencrypt Docs & Whitepapers

Zencrypt Documentation

Overview

Zencrypt is a cryptographic project designed to enhance data security through encryption and hashing. The application is evolving from a command-line interface (CLI) to a modular, scalable, and user-friendly application featuring GUI and web-service capabilities. This document provides an overview of the project, enhancement plans, and the future roadmap.

Enhancement Plan for Zencrypt CLI

Zencrypt aims to address modern cryptographic needs by incorporating a modular structure, advanced encryption techniques, and user-friendly interfaces. The enhancement plan focuses on the following key improvements:

Core Enhancements

- **UI/UX Integration:** Add a graphical user interface (GUI) using frameworks like Tkinter or PyQt.
- **Modular Structure:** Transition from a single-file CLI script to a modular architecture.
- **Web-Service Expansion:** Introduce a Flask/Django-based web-service to allow remote encryption/decryption.
- **Best Practices:** Adopt industry standards such as configuration files, logging, and environment variables for secure secret handling.

Algorithms and Data Structures

- **Advanced Cryptography:** Integrate elliptic-curve cryptography (ECC) and Argon2 hashing.

- **Concurrency:** Optimize large file encryption using multithreading or multiprocessing.
- **Data Handling:** Use sophisticated data structures like queues and Merkle trees to enhance performance.

Database Integration

- **Secure Storage:** Store keys, logs, and user information in a SQL or NoSQL database.
- **Authentication:** Implement user authentication and role-based access controls.
- **Key Management:** Add key rotation and expiry functionality for enhanced security.

Architecture and Workflow

High-Level Flowchart

1. **Startup:** Load configuration and environment variables.
2. **Logging & Initialization:** Set up modular architecture and log handlers.
3. **Interface Selection:** Choose between GUI, web-service, or CLI mode.
4. **Execution:**
    - **GUI Mode:** Launch a Tkinter/PyQt interface for user interaction.
    - **Web-Service Mode:** Run a Flask/Django server for remote operations.
    - **CLI Mode:** Present an updated menu with enhanced features.
5. **Database Operations:** Manage keys, logs, and encrypted data securely.
6. **Shutdown:** Close resources and save logs.

Modular Components

- `config.py`: Handles configuration and environment variables.

- `crypto_ops.py`: Contains encryption and hashing algorithms.
- `cli.py`: Provides command-line functionality.
- `ui.py`: Manages the GUI.
- `web.py`: Implements web-service functionality.

Database Flow

1. **Configuration Loading:** Retrieve database credentials from secure sources.
2. **Connection:** Establish a secure connection to MySQL, PostgreSQL, or MongoDB.
3. **Authentication:** Validate user credentials and roles.
4. **Operations:** Perform actions like storing keys, logging events, and managing key expiry.
5. **Response Handling:** Return results or errors to the main application.

Planned Features for Zencrypt v5

- **Scalability:** Built for long-term maintainability with a focus on modular design.
- **Performance:** Optimize large file handling and encryption tasks.
- **Security:** Incorporate advanced cryptographic algorithms and database-level security.
- **User Experience:** Simplify user interaction through a polished GUI and robust web interface.

Professional Assessment

Zencrypt reflects a commitment to secure software development, combining theoretical knowledge with practical implementation. By addressing real-world cybersecurity challenges, Zencrypt highlights skills in:

- **Full-Stack Development:** Expertise in building scalable applications.

- **Cybersecurity:** Proficiency in cryptography, secure data handling, and compliance.
- **Project Management:** Following Agile methodologies to meet client expectations.

Zencrypt’s documentation and enhancement roadmap demonstrate a dedication to quality, innovation, and practicality in cybersecurity solutions.

Personal Reflection Initial Enhancement Plans

A Shorter Description About My Enhancement Plan for ZENCRYPT CLI

My ePortfolio - Ryan Hatch Date: January 10, 2025 Zencrypt CLI – Source Code for Zencrypt v4

I've been in the CS program since the end of 2021, and I feel like I've learned more in these past few years than in my entire three decades of life. Gaining knowledge about data integrity, security, and applying programming skills feels almost like unlocking a cheat code at times. Beyond the technical skills, I've also learned how to work as part of a team, communicate effectively with stakeholders and clients, and understand the SDLC. For example, I now know that Agile Development can be the best approach to align a project with a client's needs and expectations.

As much as I wish I had taken more courses specifically focused on cybersecurity, the furthest I got were CYB 200 and CYB 210, which were still valuable in their own ways. None the less, the skills I took away from the CS program have turned out to be extremely relevant to the cybersecurity field. While I sometimes wished for the chance to take more advanced and hands-on cybersecurity classes, the CS coursework ended up being more beneficial for me overall. It gave me a broader understanding of the industry and a wider perspective on concepts that a cybersecurity specific degree might not have necessarily covered. For example, the CS program focused heavily on building strong foundational development skills, including UML diagrams, flowcharts, documentation, and pseudocode. These skills that I learned are incredibly relevant in cybersecurity and helped to provide a deeper understanding of the cyber security field as a whole.

This skillset helped to properly form and align specific priorities that are often overlooked. For example, even something as simple as knowing how to communicate with a client appropriately without overstepping any boundaries from a developer's perspective, is an important part in properly delivering a product that truly meets the client's needs and expectations.

Software Engineering and Design

Software Engineering and Design

Updating Zencrypt Software Engineering and Design:

- Add a solid UI/UX for a GUI (for example, in Python using Tkinter or PyQt)
- Migrate zencrypt CLI from a single-file CLI script into a modular project structure
- Implement industry best practices (For example using a config file approach, logging, environment variables for secrets)
- Expand to a web-service architecture (Flask/Django) so others can encrypt/decrypt remotely

This Flowchart helps to show the modular project structure, **GUI or web-service interface**, the config file usage, logging, environment-variable handling, etc along with all of the changes or additions that I am planning for a final release of Zencrypt.
With that in mind, I will mention that these adjustments wont change the core foundations of the Zencrypt v4 functions and their existing flowcharts for the logic used behind the encryption and decryption- More or less this will be used as a foundation to the final v5 of Zencrypt which will incorporate a new architectural approach that will be integrated around Zencrypt's core functionality and the bigger picture behind the cipher and its use case.

Flowchart:

```
              ┌────────────────────────────┐
              │      START (Zencrypt v5)     │
              └────────────────────────────┘
                            │
                            ▼
          ┌───────────────────────────────────┐
          │  1) Read Config & Env. Variables    │
          │     - Load config file (zencrypt.ini│
          │       or .env)                      │
          │     - Retrieve secrets / settings   │
          │       from environment variables    │
          └───────────────────────────────────┘
                            │
                            ▼
  ┌────────────────────────────────────────────┐
  │ 2) Initialize Logging & Modular Architecture  │
  │     - Configure log handlers (for example,    │
  │       file, console)                          │
  │     - Import modules (ui.py, cli.py, utils.py,│
  │       crypto_ops.py, config.py, etc.)         │
  │     - Validate any required environment vars  │
  └────────────────────────────────────────────┘
                            │
                            │
                            ▼
  ┌───────────────────────────────────────────────────────────┐
  │ 3) Check Interface Mode (GUI / Web / CLI)                      │
  │     - If "GUI" selected → proceed with Tkinter/PyQt            │
  │     - If "Web" selected → run Flask/Django server (REST / etc.)│
  │     - Else → continue in CLI mode (legacy Zencrypt approach)   │
  └───────────────────────────────────────────────────────────┘
                            │
      ┌────────────────────┴──────────────────┐
      │                                         │
      ▼                                         ▼
  ┌──────────────────────────────────────┐   ┌───────────────────────────
  │4A) LAUNCH GUI (Tkinter/PyQt)            │   │4B) LAUNCH WEB SERVICE (Flas
  │   - Build main window, forms, buttons   │   │   - Start server at configu
  │   - Connect event handlers →            │   │   - Expose endpoints for en
  │     (encrypt, decrypt, file ops, etc.)  │   │     / decryption / key mana
  │   - Integrate logging                   │   │   - Integrate logging
  └──────────────────────────────────────┘   └───────────────────────────
      │                                         │
      │                                         │
      │                                         │
      │                                         │
      │                                         │
      ▼                                         ▼
```

```
┌──────────────────────────────────┐    ┌─────────────────────────────
│ 5A) User Interacts With GUI          │    │ 5B) Users Interact With Wel
│     - Inputs text/files, chooses     │    │     - Submit encryption / (
│       encryption mode, etc.          │    │       requests
│     - Calls underlying Zencrypt      │    │     - API returns response:
│       modules (crypto_ops.py)        │    │     - Logging tracks usage
└──────────────────────────────────┘    └─────────────────────────────
    │                                         │
    └─────────────────────────────────────┤
                                              │
                                              ▼
                 ┌──────────────────────────────────┐
                 │     6) CLI Mode (If Selected)           │
                 │       - Present updated main menu       │
                 │         (hashing, encrypt text,         │
                 │         file ops, PGP, etc.)            │
                 │       - Integrate new logging           │
                 │         & config usage                  │
                 │       - Use same crypto_ops.py          │
                 │         functions as GUI/Web modes      │
                 └──────────────────────────────────┘
                                   │
                                   ▼
     ┌──────────────────────────────────────────────┐
     │ 7) Finalize Operations & Exit/Shutdown (Any Mode)   │
     │   - Close opened files, sockets, windows            │
     │   - Clean up environment variables in memory        │
     │   - Save logs if necessary                          │
     └──────────────────────────────────────────────┘
                                   │
                                   ▼
                 ┌────────────────────────┐
                 │           END            │
                 └────────────────────────┘
```

Explanation of Key Flowchart

Software Engineering and Design - Explanation of Key Flowchart:

- Read Config & Env. Variables
  - At startup, the application loads any config file and retrieves environment variables that might store secrets, database credentials, or user settings.

- Initialize Logging & Modular Architecture
  - The new version splits Zencrypt into multiple modules or files (for example, ui.py, cli.py, crypto_ops.py) for better maintainability.
  - Logging is centrally configured to capture events from all modules.
- Check Interface Mode
  - Users (or a config setting) decide whether to run Zencrypt as a GUI application, as a web service with Flask/Django, or remain in CLI mode.
- Launch GUI or Web Service
  - GUI: Creates main window with Python's Tkinter or PyQt. Buttons and menus call the same underlying crypto modules.
  - Web: Spawns a Flask or Django server, exposing REST endpoints for encryption, key management, etc.
- User Interactions
  - GUI mode: Buttons open dialogs for file encryption, text hashing, etc.
  - Web mode: Clients send requests to endpoints; server returns JSON or file responses.
- CLI Mode
  - The user is presented with your traditional command-line menu (just updated for the new modular design, logging, config usage, etc.).
- Exit/Shutdown
  - All modes converge into a final teardown sequence—closing files, saving logs, clearing secrets from memory, and gracefully exiting.

Updating Zencrypt Algorithms and Data Structures:

Updating Zencrypt Algorithms and Data Structures:

- Incorporate more advanced or efficient data structures for handling large files
- Optimize or parallelize encryption tasks using concurrency (for example, multithreading or multiprocessing using Python)
- Add elliptic-curve cryptography (ECC) or Argon2 for hashing as an alternative to SHA-256
- Evaluate computational complexity and compare different modes of encryption (CBC, GCM, etc.)

This modular approach showcases I will be enhancing Zencrypt to handle large files more efficiently, optionally leverage advanced cryptographic algorithms, and even use concurrency. The flow also leaves open the possibility of using more sophisticated data structures (for example, using queues, thread pools, or even Merkle trees for batch hashing) to optimize the app even further.

```
                   ┌──────────────────────────┐
                   │          START (v5)          │
                   └──────────────────────────┘
                                │
                                ▼
     ┌────────────────────────────────────────┐
     │1. USER CHOOSES ENCRYPTION/HASHING METHOD    │
     │   - "Encrypt Large File," "ECC Mode," │
     │     "Argon2 Hash," etc.                    │
     └────────────────────────────────────────┘
                                │
                    ┌──────────┴─────────────────────────────┐
                    │2A. USE ECC / ARGON2? (ADV. ALGORITHMS)          │
                    │   (If user selected ECC, Argon2, or other adv.)    │
                    └────────────────────────────────────────┘
                                │
        ┌───────────────┴─────────────────┐
        │ Yes (ECC / Argon2)                    │ No (Fallback: AES/SHA)
        ▼                                       ▼
  ┌──────────────────────────────┐  ┌─────────────────────────────┐
  │Initialize ECC or Argon2 logic   │    │Initialize AES, SHA-256, etc.
  │  - ECC Keygen or Argon2 hashing │    │  (Existing cipher/hash logic)
  │  - Prepare any required params  │    │
  └──────────────────────────────┘  └─────────────────────────────┘
                 │                                 │
                 └───────── Both paths eventually converge ───────┘
                                │
                                │
                                ▼
     ┌──────────────────────────────────────────────────────┐
     │3. CHECK IF CONCURRENCY IS ENABLED FOR LARGE FILE ENCRYPTION    │
     │   (Multithreading or Multiprocessing)                           │
     └──────────────────────────────────────────────────────┘
                                │
              ┌─────────┴───────────────┐
              │Yes (Use concurrency / chunking)   │
              │  (Optimized path)                  │
              └─────────────────────────┘
                                │
                                ▼
        ┌──────────────────────────────────────────────┐
        │4A. SPLIT FILE INTO CHUNKS                                  │
        │   - Read file in fixed-size blocks (for example, using  │
  │ 2MB or 4MB for each file) │
        │   - Store them in a work queue or list                     │
        └──────────────────────────────────────────────┘
                                │
                                ▼
```

```
┌──────────────────────────────────────────────────────────┐
│5A. LAUNCH THREAD POOL / MULTIPROCESS WORKERS             │
│   For each chunk in queue:                               │
│     - Encrypt/Hash chunk with chosen algorithm (ECC, AES, │
│       Argon2, etc.)                                      │
│     - Store partial results (ciphertext, checksums)      │
└──────────────────────────────────────────────────────────┘
                       │
                       ▼
┌───────────────────────────────────────────────────────┐
│6A. REASSEMBLE CHUNKS                                  │
│   - Combine encrypted chunks or hashed results        │
│   - If streaming approach, write partial chunks to output │
│     file as they finish                               │
└───────────────────────────────────────────────────────┘
                       │
                       ▼
         ┌────────────────────────────────────────┐
         │No (Single-threaded or small file)      │
         │  (Straight-line path)                  │
         └────────────────────────────────────────┘
                       │
                       │
                       │
                       ▼
┌─────────────────────────────────────────────────────┐
│4B/5B/6B. SINGLE-PASS ENCRYPT/HASH                   │
│   - If concurrency is off or file is small, process in │
│     one pass with standard logic (AES/ECC, etc.)    │
└─────────────────────────────────────────────────────┘
                       │
                       ▼
   ┌──────────────────────────────────────────────────┐
   │7. RETURN/STORE FINAL OUTPUT                      │
   │   - Write final ciphertext or hash to file/db, etc. │
   │   - Provide success message or handle errors     │
   └──────────────────────────────────────────────────┘
                       │
                       ▼
             ┌────────────────────────┐
             │         END            │
             └────────────────────────┘
```

Flowchart Explanation

Algorithms and Data Structures Flowchart Explanation:

1. **User Chooses Method**
   The user decides whether they want to encrypt a large file, use elliptic-curve cryptography, or generate Argon2 hashes (instead of SHA-256).
2. **Check for Advanced Algorithms**
   - If advanced algorithms (ECC, Argon2) are selected, Zencrypt initializes those cryptographic methods and any parameters, for example ECC curves or Argon2 memory cost.
   - Otherwise, it defaults to existing methods like AES or SHA-256.
3. **Check Concurrency Option**
   - If enabled (for example, for large files), Zencrypt uses a multithreading or multiprocessing approach to handle chunk-based encryption or hashing in parallel.
4. **Chunk File (If Using Concurrency)**
   - For large file encryption, read the file in small chunks, place them in a queue or list, and then distribute them to worker threads/processes.
   - If concurrency is **off**, a simpler single-pass encryption or hashing routine is used.
5. **Parallel Processing**
   - Each worker encrypts or hashes its chunk with the chosen algorithm.
   - This step significantly speeds up the process on multi-core systems.
6. **Reassemble Results**
   - Combine or stream the partially encrypted chunks into the final file or combine hashed outputs.
   - In a hashing scenario, you might incorporate a final combine step (for example, using a Merkle tree approach).
     1. **Output**
        - Write the final encrypted file or final hash to its destination.
        - Provide a success message or handle errors as needed.

Updating Zencrypt Databases

Updating Zencrypt Databases:

- Store keys and user information in a SQL or NoSQL database (MySQL, SQLite, MongoDB, PostgreSQL, etc.)
- Integrate key expiry and key rotation logs in a secure database table
- Implement user authentication and roles for key usage (which ties directly into data defense)
- Possibly demonstrate how to store encrypted data objects within the database and decrypt on retrieval

With this final release of Zencrypt v5 I am planning on integrating database functionality for storing keys, user credentials, logs, analytics, and even encrypted data. This diagram focuses specifically on the database enhancement portion of Zencrypt and shows the major steps for connecting, authenticating, and performing database operations securely.

```
                 ┌───────────────────────────────┐
                 │          START - DB FLOW          │
                 └───────────────────────────────┘
                               │
                               ▼
  ┌─────────────────────────────────────────────────────┐
  │ 1) Load DB Config & Env. Vars                            │
  │    - Retrieve database credentials, connection strings   │
  │      from config file or environment variables           │
  └─────────────────────────────────────────────────────┘
                               │
                               ▼
  ┌────────────────────────────────────────────────────┐
  │ 2) Initialize Database Connection                      │
  │    - Connect to SQL/NoSQL DB (MySQL, PostgreSQL,       │
  │      MongoDB, etc.)                                    │
  │    - Handle exceptions if DB is unreachable            │
  └────────────────────────────────────────────────────┘
                               │
                               ▼
  ┌─────────────────────────────────────────────────────┐
  │ 3) Check/Perform User Authentication                     │
  │    - If user not logged in: prompt login/signup          │
  │      (salt + hash password, verify in DB)                │
  │    - Generate or validate JWT on successful login        │
  │    - Retrieve any user roles/permissions                 │
  └─────────────────────────────────────────────────────┘
                               │
                               │
                      ┌────────┴───────────────┐
                      │ User Authenticated?        │
                      │ (JWT Valid?)               │
                      └────────────────────────┘
                               │
                             │
              ┌────────────┴───────────┐
              │          NO                 │
              │  (Auth fails or token       │
              │   invalid)                  │
              ▼                             │
      ┌────────────────────────┐          │
      │  Return Error /            │          │
      │  Prompt Re-Login           │          │
      └────────────────────────┘          │
                                          │
                                            │ YES
                                            ▼
                  ┌─────────────────────────────────────┐
                  │ 4) Present DB-Related Actions            │
```

```
                │     A) Store/Retrieve Keys in DB          │
                │     B) Store/Retrieve Encrypted Data      │
                │     C) Log Usage (encryption events)      │
                │     D) Insert Access / Audit Records      │
                │     E) Key Expiry & Rotation Checks       │
                └──────────────────────────────────────┘
                                    │
                                    ▼
         ┌─────────────────────────────────────────────┐
         │ 5) Execute Chosen DB Operation                   │
         │    - "Insert new key," "Get user’s key,"  │
         │      "Write encryption log entry," etc.          │
         │    - Enforce user permissions/roles              │
         │    - Handle success/errors gracefully            │
         └─────────────────────────────────────────────┘
                                    │
                                    ▼
  ┌──────────────────────────────────────────────────────┐
  │ 6) Return Response / Results to Zencrypt Main App    │
  │    - Encryption key retrieved? Next step is encrypt. │
  │    - Log entry stored? Display success message.      │
  │    - Access denied? Inform user with error/log details. │
  └──────────────────────────────────────────────────────┘
                                    │
                       │
                                    ▼
      ┌──────────────────────────────────────────────────┐
      │ 7) Continue Zencrypt Workflow or Exit             │
      │    - If more actions needed, loop back (Step 4)   │
      │    - Otherwise finalize/close DB connection if done │
      └──────────────────────────────────────────────────┘
                                    │
                       │
                                    ▼
                  ┌─────────────────────────────────┐
                  │          END - DB FLOW              │
                  └─────────────────────────────────┘
```

Flowchart Explanation

Database Flowchart Explanation:

- **Load DB Config & Env. Vars**

- Zencrypt reads database credentials (keys, username, password, host, port, etc.) from environment variables or a secure config file (for example, env, zencrypt.ini).

- **Initialize Database Connection**
  - The application attempts to connect to the chosen database system whether it is MySQL, MongoDB, PostgreSQL, etc.
  - Handles exceptions if the DB is unreachable or credentials are invalid.
- **Check/Perform User Authentication**
  - If the user is not already authenticated, Zencrypt prompts for login or signup.
  - Passwords are salted and hashed using PBKDF2 or Argon2 before checking against the stored hash in the DB.
  - On success, it either issues or validates a JWT/ JSON Web Token.
  - Retrieves user roles/ permissions from the Database (for example, "admin," "basic_user," etc.).
- **Present DB-Related Actions**
  - Once authenticated, the user can select different database-related functions, such as:
    - **Store or retrieve** encryption keys from a secure table.
    - **Store or retrieve** encrypted data objects.
    - **Log usage**: Insert a record of encryption or decryption events like timestamps, user IDs, and file info.
    - **Insert access or audit records** for compliance.
    - **Check and handle key expiry** or rotation. For example, if a key is expired, deny usage or auto rotate.
- **Execute Chosen DB Operation**
  - The appropriate Zencrypt function runs. For example, store_key(), retrieve_key(), log_event(), etc.
  - Zencrypt checks that the user has the correct **role/permission** for the action.
  - Success or error is returned.
- **Return Response / Results to Main App**

  - If a key was retrieved, Zencrypt can proceed to encrypt or decrypt data using that key.
  - If logs were stored, it confirms success.
  - If access is denied or an error occurs, Zencrypt handles it gracefully and logs it.
- **Continue Zencrypt Workflow or Exit**
  - The user can continue performing more database actions or return to other parts of Zencrypt (for example, navigating around the encryption manager, PGP, and all GUI/CLI menus).
  - Once finished, Zencrypt closes the database connection gracefully as part of its teardown function.

Skills and Illustrated Outcomes

Skills and Illustrated Outcomes:

- **Code Review:**
    - **Commitment to Quality Assurance:** Zencrypt will help to show off a strong dedication to high-quality software development through adhering to industry best practices, including modular design, clean coding standards, along with having comprehensive testing methods for QA.
    - **Project Development and Evolution:** My ePortfolio will be able to highlight my ability to take projects from a simple foundational concept to a professional and full-fledged product, incorporating advanced features such as GUI integration, web-based interfaces, and secure database connection.
    - **Cybersecurity Tools and Data Handling:** My ePortfolio will help to show that I have a strong proficiency and understanding in implementing encryption and cryptographic algorithms into my work which helps to show that I have a strong passion for cyber security and understanding about the way that data is properly managed and secured.
- **Narratives:**
    - **Detailed Thought Process:** I will document the thought process and reasoning behind each improvement, like comparing cryptographic methods (for example, ECC vs. SHA-256), improving speed with multitasking, all while following security best practices by adding features like rotating keys and using safe encryption methods.
    - **Scalability and Maintainability:** Showcasing all of my design choices, tailored mostly towards long-term scalability and ease of maintenance. For this example, I will be turning a single file CLI script into a modular web-based application.
    - **Real-World Problem Solving:** I will explain in the documentation for Zencrypt how these specific improvements help to solve real world cyber security challenges, with a focus on databases, secure data handling, encryption, and authentication.
- **Professional Self-Assessment:**
    - **Skillset Take-aways:** Throughout my venture as a cybersecurity specialist and a computer science student, I have developed a strong skill set to help

me move forward in the fields that I want to pursue. Some skills I have taken away from college are very powerful and solid foundationally.

For example, understanding the fundamentals of databases, along with how to develop them or using Sequel or MongoDB to incorporate the ability to further integrate a reliable and secure connection to a database. Another example of a skillset I can take away from this program and add to my ePortfolio will be the understanding of cryptography and optimizing algorithms.

With that said, these skills will be openly demonstrated through the Zencrypt roadmap and documentation, where I explain the logic applied behind the industry standard techniques that I will use for encryption and to develop a user-friendly and reliable UI/UX solution for managing sensitive data. The success of this project will help to properly project my ability to balance innovation with security and efficiency. My goal on this continent is to help in solving real-world problems in a constantly changing digital environment.

- **Full-Stack Development and Web-Service Architecture:** My skills in full-stack development and web-service architecture will be showcased with the release of Zencrypt v5, building on the foundation and expansion introduced in Zencrypt v4. The updated version will be built for scalability and feature a user-friendly UI/UX that will be powered by frameworks like Flask.

  This project will help to show my dedication to meeting goals and requirements while following industry standards, with a focus on consistently enhancing Zencrypt's security and performance. My goal is to polish Zencrypt CLI into a final cyber security program that provides reliability, security and efficiency with advanced features yet keeps it simple and practical for all users.

- **Proven Expertise:** By combining all of my technical skills with an understanding of industry needs, my ePortfolio will help to prove that I can not only understand but also help to solve problems in cyber security. My projects help to properly mirror my ability to deliver secure, scalable, and practical solutions that can go beyond any set expectations.

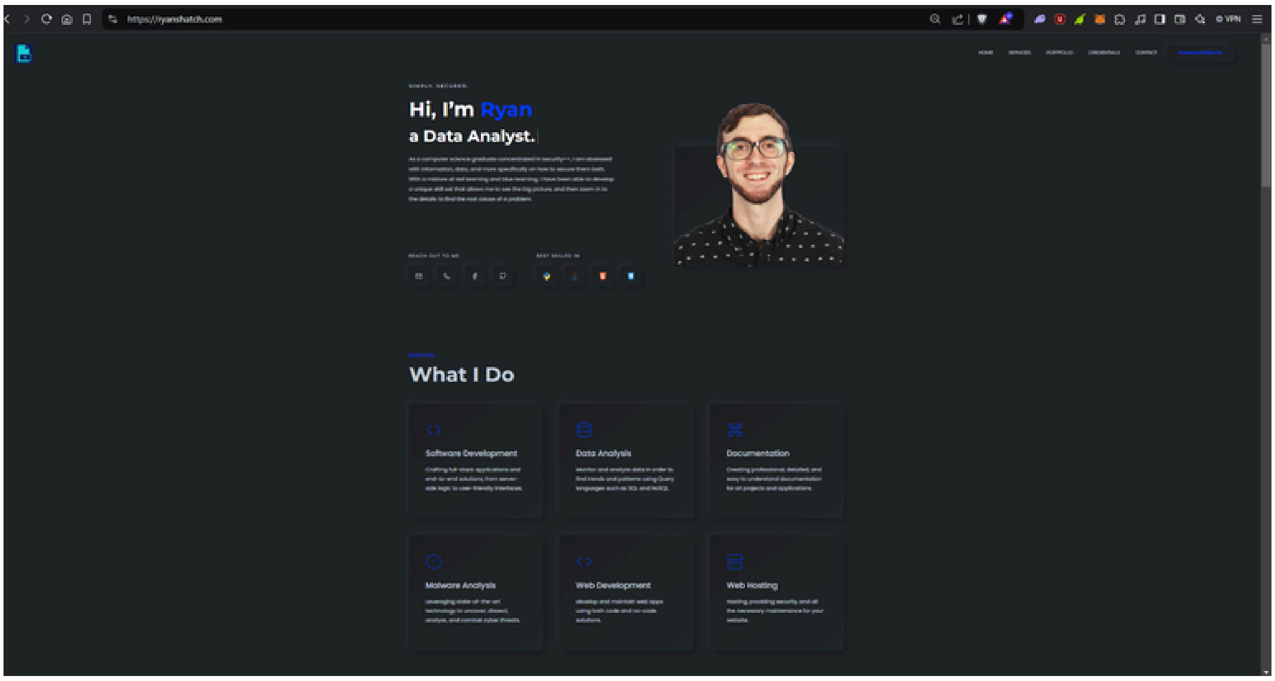
ePortfolio in current state

https://www.zencrypt.app
www.zencrypt.app

Web Application

ePortfolio and GitHub Screen Shots:

ePortfolio i: https://www.ryanshatch.com



Hi, I'm Ryan
a Data Analyst.

As a computer science graduate concentrated in security++, I am obsessed with information, data, and more specifically on how to secure them both. With a mixture of red teaming and blue teaming, I have been able to develop a unique skill set that allows me to see the big picture, and then zoom in to the details to find the root cause of a problem.

What I Do

Software Development
Crafting full-stack applications and end-to-end solutions, from server-side logic to user-friendly interfaces.

Data Analysis
Monitor and analyze data in order to find trends and patterns using Query languages such as SQL and NoSQL.

Documentation
Creating professional, detailed, and easy to understand documentation for all projects and applications.

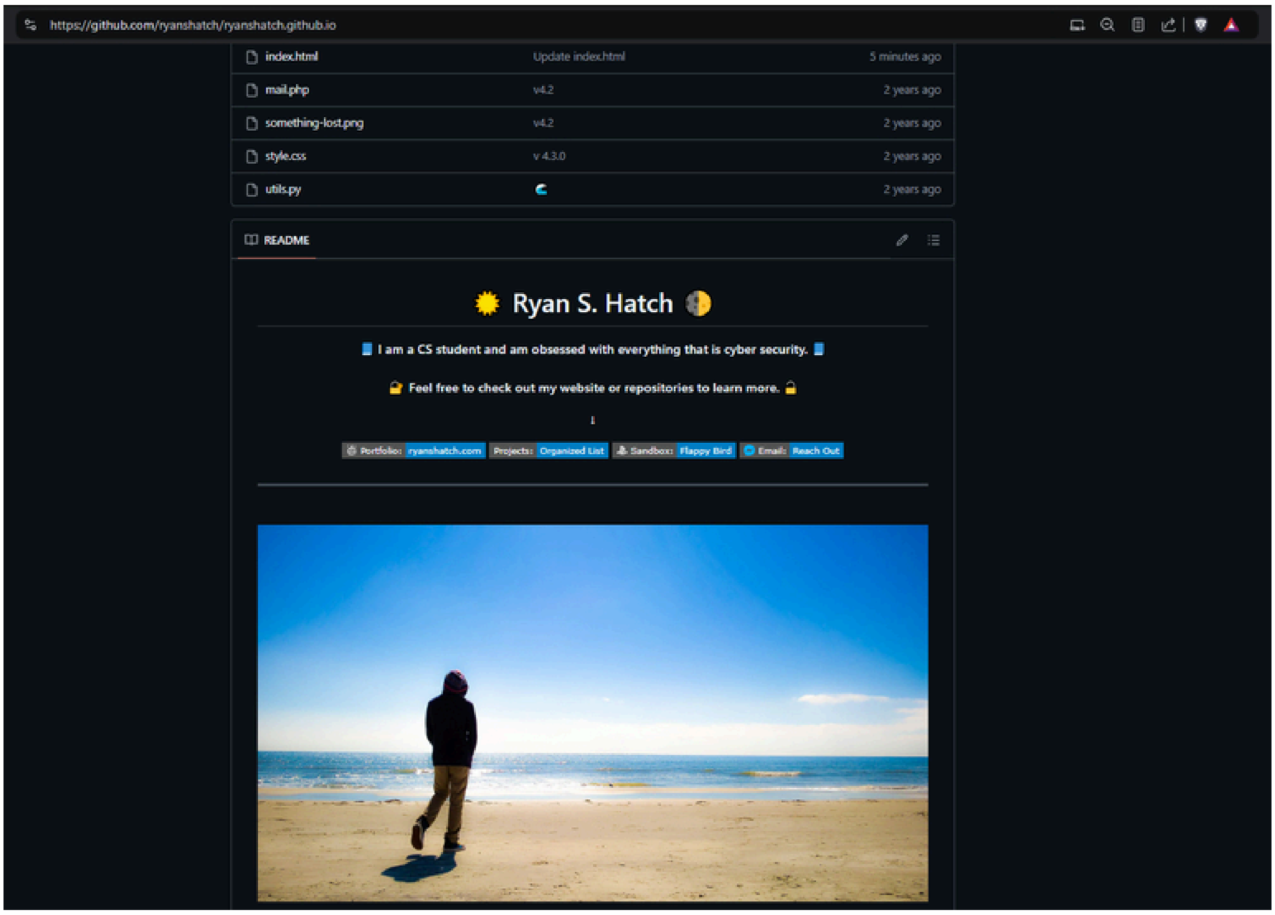
Malware Analysis
Leveraging state-of-the-art technology to uncover, dissect, analyze, and combat cyber threats.

Web Development
develop and maintain web apps using both code and no-code solutions.

Web Hosting
Hosting, providing security, and all the necessary maintenance for your website.

A screenshot of my website.

ePortfolio ii: https://www.github.com/ryanshatch/readme



https://github.com/ryanshatch/ryanshatch.github.io

index.html	Update index.html	5 minutes ago
mail.php	v4.2	2 years ago
something-lost.png	v4.2	2 years ago
style.css	v 4.3.0	2 years ago
utils.py		2 years ago

README

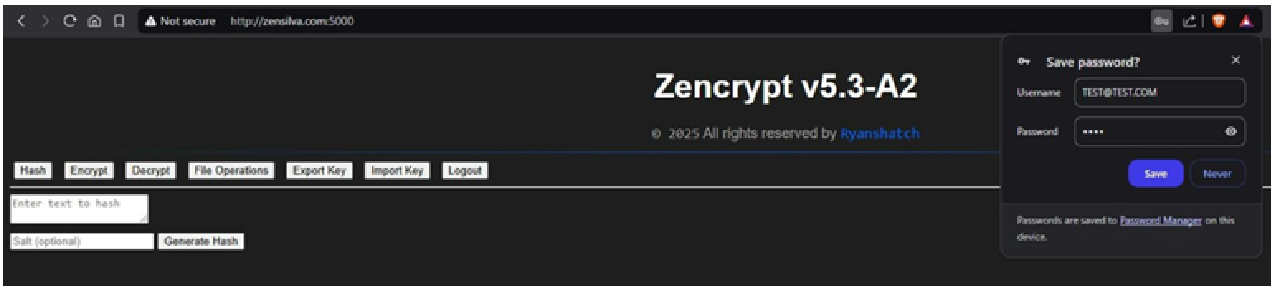
Ryan S. Hatch

I am a CS student and am obsessed with everything that is cyber security.

Feel free to check out my website or repositories to learn more.

A screenshot of my Github and my ePortfolio reflecting updates.

AWS EC2 Instance: zencrypt.app



Not secure http://zensilva.com:5000

Zencrypt v5.3-A2

© 2025 All rights reserved by Ryanshatch

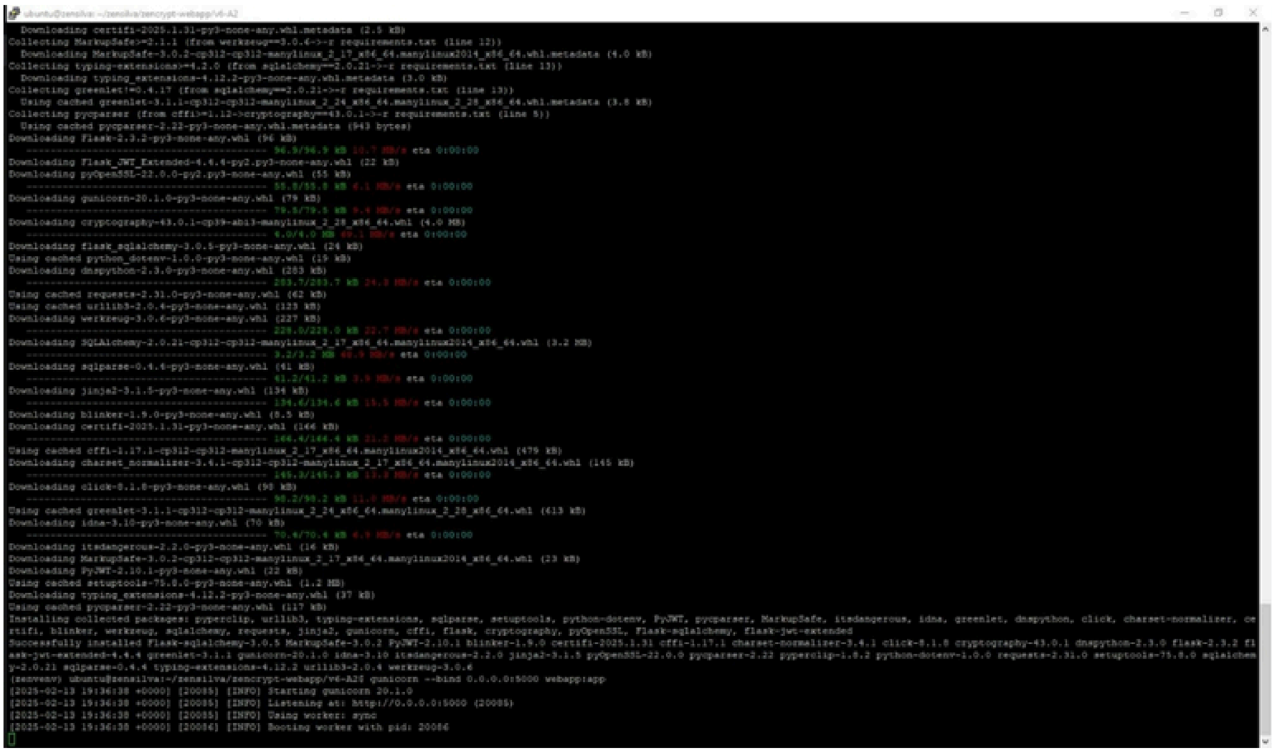
Save password?

Username: TEST@TEST.COM

Password: ••••

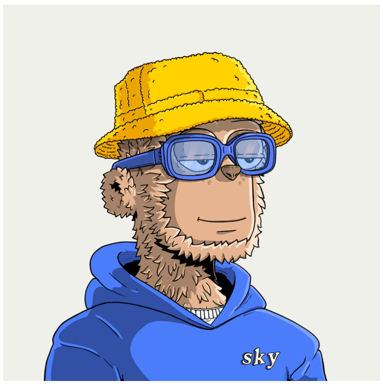
Passwords are saved to Password Manager on this device.

website testing



```
  Downloading certifi-2025.1.31-py3-none-any.whl.metadata (2.5 kB)
Collecting MarkupSafe>=2.1.1 (from werkzeug==3.0.6->-r requirements.txt (line 12))
  Downloading MarkupSafe-3.0.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.0 kB)
Collecting typing-extensions>=4.2.0 (from sqlalchemy==2.0.21->-r requirements.txt (line 13))
  Downloading typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 kB)
Collecting greenlet!=0.4.17 (from sqlalchemy==2.0.21->-r requirements.txt (line 13))
  Using cached greenlet-3.1.1-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl.metadata (3.8 kB)
Collecting pycparser (from cffi>=1.12->cryptography==43.0.1->-r requirements.txt (line 5))
  Using cached pycparser-2.22-py3-none-any.whl.metadata (943 bytes)
Downloading flask-2.3.2-py3-none-any.whl (96 kB)
   ---------------------------------------- 96.9/96.9 kB 10.7 MB/s eta 0:00:00
Downloading Flask_JWT_Extended-4.4.4-py2.py3-none-any.whl (22 kB)
Downloading pyOpenSSL-22.0.0-py2.py3-none-any.whl (55 kB)
   ---------------------------------------- 55.8/55.8 kB 6.1 MB/s eta 0:00:00
Downloading gunicorn-20.1.0-py3-none-any.whl (79 kB)
   ---------------------------------------- 79.5/79.5 kB 9.4 MB/s eta 0:00:00
Downloading cryptography-43.0.1-cp39-abi3-manylinux_2_28_x86_64.whl (4.0 MB)
   ---------------------------------------- 4.0/4.0 MB 69.1 MB/s eta 0:00:00
Downloading flask_sqlalchemy-3.0.5-py3-none-any.whl (24 kB)
Using cached python_dotenv-1.0.0-py3-none-any.whl (19 kB)
Downloading dnspython-2.3.0-py3-none-any.whl (283 kB)
   ---------------------------------------- 283.7/283.7 kB 24.3 MB/s eta 0:00:00
Using cached requests-2.31.0-py3-none-any.whl (62 kB)
Using cached urllib3-2.0.4-py3-none-any.whl (123 kB)
Downloading werkzeug-3.0.6-py3-none-any.whl (227 kB)
   ---------------------------------------- 228.0/228.0 kB 22.7 MB/s eta 0:00:00
Downloading SQLAlchemy-2.0.21-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.2 MB)
   ---------------------------------------- 3.2/3.2 MB 48.9 MB/s eta 0:00:00
Downloading sqlparse-0.4.4-py3-none-any.whl (41 kB)
   ---------------------------------------- 41.2/41.2 kB 3.9 MB/s eta 0:00:00
Downloading jinja2-3.1.5-py3-none-any.whl (134 kB)
   ---------------------------------------- 134.6/134.6 kB 15.5 MB/s eta 0:00:00
Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Downloading certifi-2025.1.31-py3-none-any.whl (166 kB)
   ---------------------------------------- 166.4/166.4 kB 21.2 MB/s eta 0:00:00
Using cached cffi-1.17.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (479 kB)
Downloading charset_normalizer-3.4.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (145 kB)
   ---------------------------------------- 145.3/145.3 kB 13.3 MB/s eta 0:00:00
Downloading click-8.1.8-py3-none-any.whl (98 kB)
   ---------------------------------------- 98.2/98.2 kB 11.0 MB/s eta 0:00:00
Using cached greenlet-3.1.1-cp312-cp312-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (613 kB)
Downloading idna-3.10-py3-none-any.whl (70 kB)
   ---------------------------------------- 70.4/70.4 kB 6.9 MB/s eta 0:00:00
Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Downloading MarkupSafe-3.0.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (23 kB)
Downloading PyJWT-2.10.1-py3-none-any.whl (22 kB)
Using cached setuptools-75.8.0-py3-none-any.whl (1.2 MB)
Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Using cached pycparser-2.22-py3-none-any.whl (117 kB)
Installing collected packages: pyperclip, urllib3, typing-extensions, sqlparse, setuptools, python-dotenv, PyJWT, pycparser, MarkupSafe, itsdangerous, idna, greenlet, dnspython, click, charset-normalizer, ce
rtifi, blinker, werkzeug, sqlalchemy, requests, jinja2, gunicorn, cffi, flask, cryptography, pyOpenSSL, flask-sqlalchemy, flask-jwt-extended
Successfully installed Flask-sqlalchemy-3.0.5 MarkupSafe-3.0.2 PyJWT-2.10.1 blinker-1.9.0 certifi-2025.1.31 cffi-1.17.1 charset-normalizer-3.4.1 click-8.1.8 cryptography-43.0.1 dnspython-2.3.0 flask-2.3.2 fl
ask-jwt-extended-4.4.4 greenlet-3.1.1 gunicorn-20.1.0 idna-3.10 itsdangerous-2.2.0 jinja2-3.1.5 pyOpenSSL-22.0.0 pycparser-2.22 pyperclip-1.8.2 python-dotenv-1.0.0 requests-2.31.0 setuptools-75.8.0 sqlalchem
y-2.0.21 sqlparse-0.4.4 typing-extensions-4.12.2 urllib3-2.0.4 werkzeug-3.0.6
(zenvenv) ubuntu@zensilva:~/zensilva/zencrypt-webapp/v6-A2$ gunicorn --bind 0.0.0.0:5000 webapp:app
[2025-02-13 19:36:38 +0000] [20085] [INFO] Starting gunicorn 20.1.0
[2025-02-13 19:36:38 +0000] [20085] [INFO] Listening at: http://0.0.0.0:5000 (20085)
[2025-02-13 19:36:38 +0000] [20085] [INFO] Using worker: sync
[2025-02-13 19:36:38 +0000] [20086] [INFO] Booting worker with pid: 20086
```

EC2 instance



Ryanshatch