Ryan Hatch  
Date: January 10, 2025  
[CS 499 - Computer Science Capstone 2025](https://learn.snhu.edu/d2l/home/1831836)  
[My ePortfolio](http://www.ryanshatch.com/) - Module One  
[Zencrypt CLI](https://github.com/ryanshatch/zencrypt/tree/v5) – All documentation on any updates and changes made to the final stable release.  
GitHub Repository: [https://github.com/ryanshatch/zencrypt](https://github.com/ryanshatch/zencrypt/tree/v5)

# ***Enhancement Plan for ZENCRYPT CLI:***

I have been in the CS Program since roughly EOY in 2021 and I feel like I have learned more in the last four years than I have learned in my entire three decades of human living. Data Integrity and security, along with the knowledge and capability to apply programming skills into the mix can almost feel like a cheat code sometimes. Even in terms of understanding how to work as a team and how to communicate as a team player to stake holders and clients, how exactly do SDLC work, in what scenario can, for example, Agile Development be applied in order to properly align with a client and their expectations?

As much as I wish I spent more time learning subjects that were tailored more towards cyber security itself, the farthest I went was with CYB 200 and CYB 210 which were beyond rewarding in their own ways. The skills that I learned from the CS program can easily be applied towards the cyber security industry, no matter how badly I wish I was able to do the “cool looking classes” instead of the CS courseload, which in the end was more beneficial for me because I am able to understand things in a much more broad lens than if I were to have gotten my 4 year degree with the cyber security program. For example, they absolutely tailored a fair chunk of the CS program towards development competency in general, for example, UML diagrams, flow charts, documentation, pseudo code, etc.

All of these are priorities that are skimmed over a lot of times even though, for example, simply knowing how to talk to the client appropriately without crossing boundaries from a developer’s lens/ position, is a very crucial step towards being able to properly deliver a product the client actually wanted to begin with.

## **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**.**

            ┌──────────────────────────────┐

            │      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 (e.g., 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 (Flask/Django)    │

│   - Build main window, forms, buttons   │  │   - Start server at configured port     │

│   - Connect event handlers →            │  │   - Expose endpoints for encryption     │

│     (encrypt, decrypt, file ops, etc.)  │  │     / decryption / key management       │

│   - Integrate logging                   │  │   - Integrate logging                   │

└─────────────────────────────────────────┘  └─────────────────────────────────────────┘

    │                                         │

    │                                         │

  │                                         │  
   │                                         │

  │                                         │  
 ▼                                         ▼

┌───────────────────────────────────────┐    ┌───────────────────────────────────────────┐

│ 5A) User Interacts With GUI           │    │ 5B) Users Interact With Web Endpoints     │

│     - Inputs text/files, chooses      │    │     - Submit encryption / decryption      │

│       encryption mode, etc.           │    │       requests                            │

│     - Calls underlying Zencrypt       │    │     - API returns responses or files      │

│       modules (crypto\_ops.py)         │    │     - Logging tracks usage / errors       │

└───────────────────────────────────────┘    └───────────────────────────────────────────┘

    │                                          │

    └──────────────────────────────────────────┤

                                               │

                                               ▼

                        ┌──────────────────────────────────────┐

                        │     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            │

                        └──────────────────────────┘

## *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/files (e.g., 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:**

* Incorporate more advanced or efficient data structures for handling large files
* Optimize or parallelize encryption tasks using concurrency (e.g., multithreading or multiprocessing in 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 how you are 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 further optimize the application.

Flowchart for Algorithms and Data Structures:

                 ┌─────────────────────────────┐

                 │         START (v5)          │

                 └─────────────────────────────┘

                          │

                          ▼

   ┌─────────────────────────────────────────────┐

   │1. USER CHOOSES ENCRYPTION/HASHING METHOD    │

   │   - e.g., "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 (e.g., 4MB each)     │

      │   - 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            │

                └─────────────────────────┘

## 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 (e.g., ECC curves, Argon2 memory cost).
   * Otherwise, it defaults to existing methods like AES or SHA-256.
3. **Check Concurrency Option**
   * If enabled (e.g., 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 (e.g., a Merkle tree approach, if desired).
7. **Output**
   * Write the final encrypted file or final hash to its destination.
   * Provide a success message or handle errors as needed.

## **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                 │

                │    - e.g. "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            │

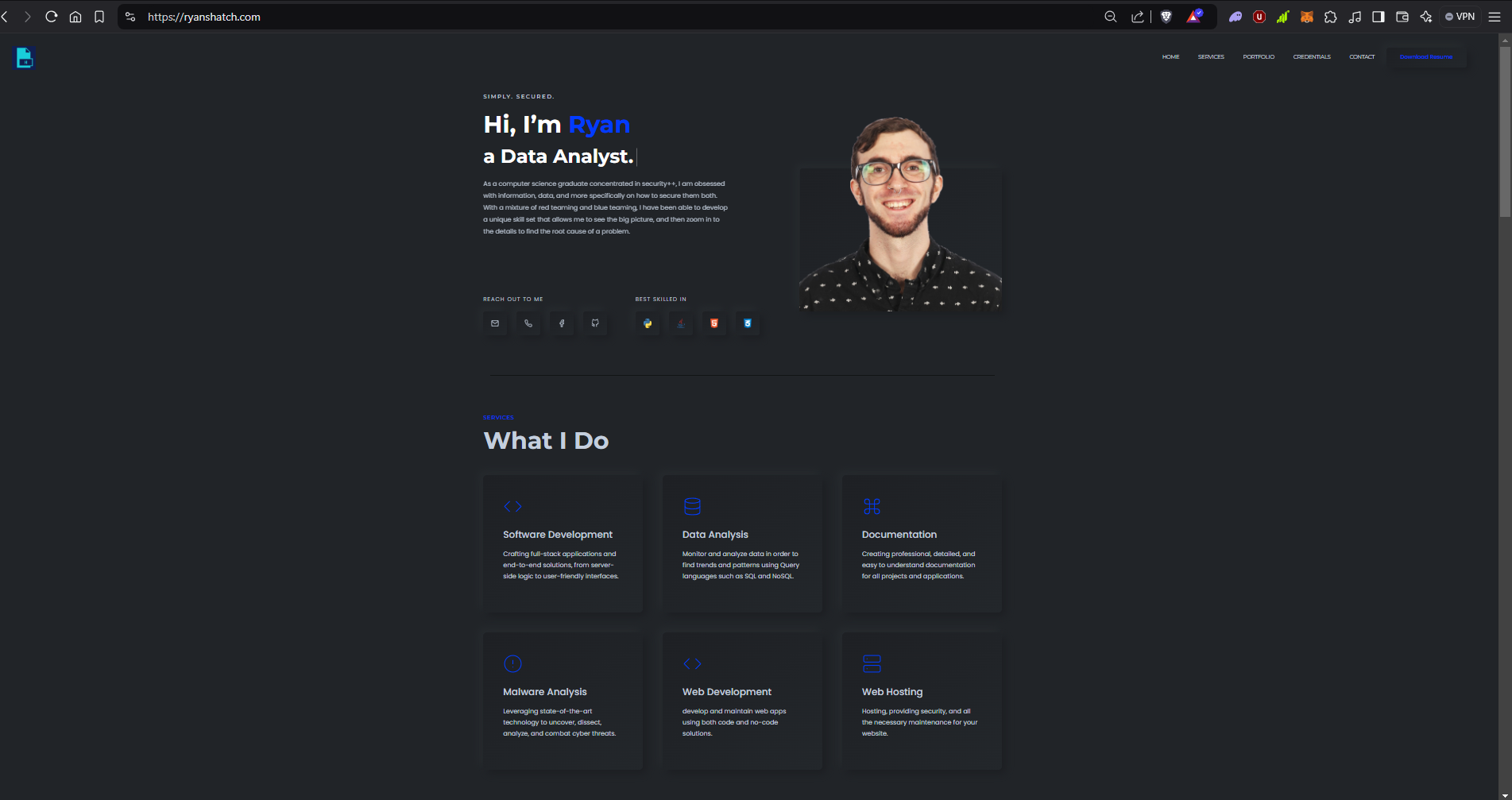
                          └───────────────────────────────────┘

## Database Flowchart Explanation:

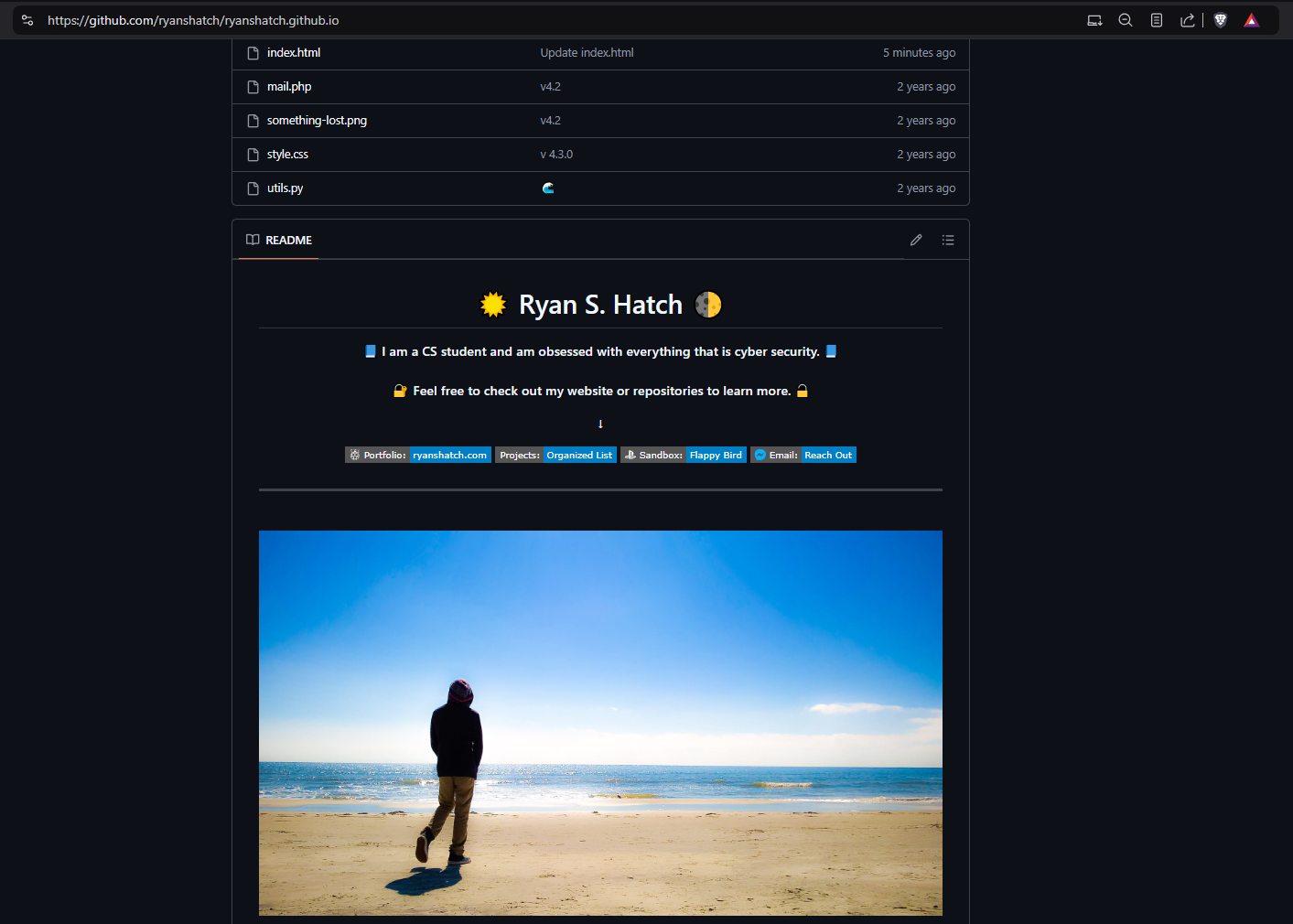
* **Load DB Config & Env. Vars**
  + Zencrypt reads database credentials (e.g., username, password, host, port) from environment variables or a secure config file (e.g., .env, zencrypt.ini).
* **Initialize Database Connection**
  + The application attempts to connect to the chosen database system (e.g., MySQL, MongoDB, PostgreSQL).
  + 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 (e.g., PBKDF2, Argon2) before checking against the stored hash in the DB.
  + On success, it either issues or validates a JSON Web Token (JWT).
  + Retrieves user roles/permissions from DB (e.g., “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 (timestamp, user ID, file info).
    - **Insert access or audit records** for compliance.
    - **Check and handle key expiry** or rotation (if a key is expired, deny usage or auto-rotate).
* **Execute Chosen DB Operation**
  + The appropriate function runs (e.g., 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 (e.g., encryption manager, PGP, GUI/CLI menus).
  + Once finished, Zencrypt closes the database connection gracefully as part of its teardown.

## ***ePortfolio*** *and* ***GitHub*** *Screen Shots:*

**ePortfolio i**



**ePortfolio ii**



## **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 absolutely robust by nature.   
      
    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.