**Website Documentation**

**PSS Paul Stokreef Secretmanager**

**Paul Stokreef**

**HBO-ICT CYBERSECURITY, 500964089**

**September 2024**

**Introduction**

An extensive documentation on the technical structure of my secret manager website, including full creation process analysis, used Flask modules analysis, HTML / CSS documentation, full PEN test, wireframes and wireflows, the relation between webpage and Flask module and JavaScript analysis.

This document has been created using LaTeX and Microsoft Word.

The Secret Manager project is a secure web application that allows users to store, retrieve, and manage their passwords and secrets. This documentation will provide detailed information on how the website functions, covering its technical architecture, design, and security measures. It is built using Flask as a backend framework, HTML, CSS for design, and various security practices such as encryption and Two-Factor Authentication (2FA).

This document will cover all HTML, CSS, and Flask modules, including their relationships, functionality, and implementation details. It also contains wireframes and wireflows, providing a visual representation of the system and web pages

Table of Contents

Each underlined word in this table is a hyperlink; ctrl + click to get redirected

[0 Introduction](#Page2)

[1 Project Overview](#Page31)

[1.1 GitLab Issues and main plan](#Page32)

[2 Web Pages Documentation (HTML / CSS)](#Page61)

[2.1 Home (index.html)](#Page62)

[2.2 Contact & About (contact.html & about.html)](#Page8)

[2.3 Register (register.html)](#Page81)

[2.4 Dashboard (dashboard.html)](#Page10)

[2.5 2FA (2fa.html)](#Page14)

[2.6 CSS (globals.css)](#Page15)

[3 Flask Modules Documentation](#Page16)

[3.1 Routes and Functions](#Page16)

3.2 Flask-Jinja Integration

3.3 User Authentication and Authorization

4 Wireframes, Wireflows and Navigation Structure

5 Flask and Webpage Relationship

6 PEN Test documentation

7 Conclusion and Sources

**1 Project Overview**

This project serves as a password manager with features like secure storage, user registration, login with 2FA, and encrypted passwords. It uses Flask for the backend, Jinja for templating, and HTML/CSS for frontend design. Some functions use JavaScript. Users can create accounts, store passwords and access them securely through the dashboard.

* 1. **GitLab Issues and main plan**

Before I began coding the Secret Manager project, I outlined a clear plan and structure for how I would approach creating the project. I focused on breaking down the project into smaller tasks, each reflecting an essential feature of the final web application. Very soon, I found out the given GitLab issues did exactly this for me. This strategy helped me systematically tackle each requirement and monitor my progress effectively. Some of the key issues that were assigned or created include main page creation, navbar creation, having a single CSS file, creating a registration page, creating a login page, creating a secrets page, 2FA integration and having a responsive design. By tracking these issues I could easily stay organized and manage feedback cycles. Especially thanks to GitLab’s labelling system, I could really organize each task. Each task was broken down and resolved before moving to the next, so I had a clear path and vision to complete the project. At the start of this project, I spent a significant amount of time planning the structure and functionality before diving into the code. This planning stage was crucial because it gave me a clear roadmap and prevented me from making hasty decisions that could lead to errors or inefficient designs. As with many development projects, the coding was done in English, a global standard among developers. Therefore, I decided to maintain all of my project documentation and comments in English as well, to align with best practices in software engineering / cybersecurity. I initially worked on an alpha version of this project on my **personal GitHub**, allowing me to experiment with key features. Afterward, I moved the project to the **HvA GitLab**, uploading the alpha components piece by piece. This transition ensured the project structure remained intact while adhering to the GitLab issue-tracking system. During the creation of the GitLab version of the project, I created a plan and put this plan in a separate file. As I progressed, I organized the GitLab issues in a way that made the most sense to me and the flow of the project. The plan was executed in a way that focused on completing core functionality first before moving to the more complex and fine-tuning tasks like 2FA, security testing and HTTPS integration.

As of 19th September 2024, the plan looked like this:

**# The plan**

- ✓ Main page (index.html)

- ✓ Make a wireframe, combine that in the wireflow and code the part

- ✓ Navigation bar

- ✓ 1 css file

- ✓ Registration page

- ✓ Registration instructions

- ✓ Login page

- ✓ Responsive design (so I don't have to rewrite anything / everything later)

- ✓ Registration to .CSV file

- ✓ Secrets page

- ✓ Recording secrets

- ✓ Showing secrets

- ✓ Unaccessable secrets page

- ✓ Logging out

- ✓ Changing secrets

- ✓ Deleting secrets

- ✓ Scrolling secrets

- ✓ Loginpage validation

- ✓ 2FA

- ✓ HTTPS

- PEN test

- Website documentation

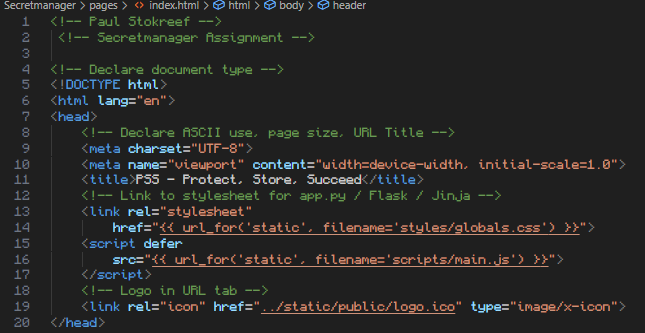
**2. Web Pages Documentation (HTML / CSS)**

I began by organizing a standard file structure based on the framework I knew we would use—Flask. Knowing Flask operates with templates and static files, I created a folder system that separated my HTML pages from static assets such as CSS, JavaScript, and images. Specifically, I created the **pages/** and **static/** folders. The pages folder contains all the HTML templates used in the project, which are rendered by Flask and use Jinja templating to dynamically load content. The static folder contains all static assets. I started by prototyping the website with wireframes and wireflows to visualize the user experience and interface. These helped in mapping out the navigation and interaction between different web pages. The final versions of the wireframe and wireflow will be shown at the end of the documentation, but they laid the groundwork for how the homepage was structured.

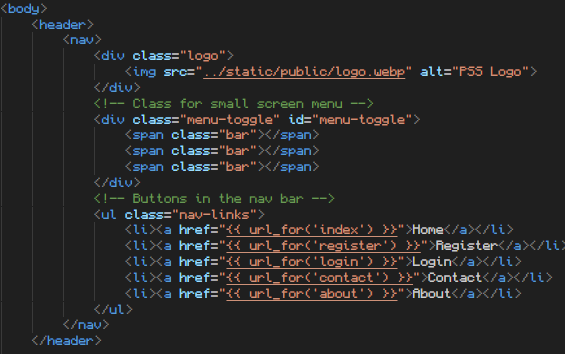
**Crucial Note: the visuals of the website are available in the wireframes / wireflows section**

**2.1 Home (index.html)**

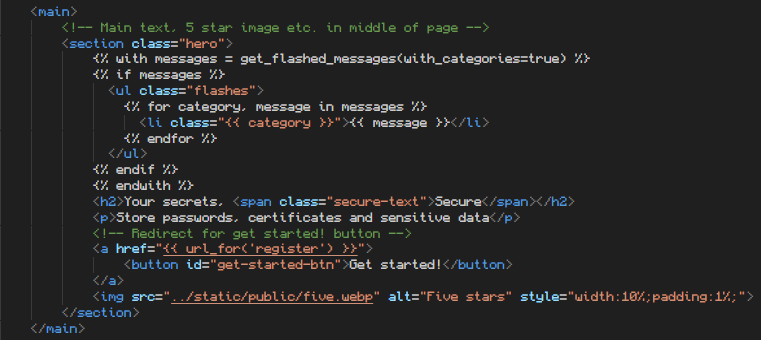
The homepage was the very first thing I started coding. It introduces my website, it highlights the purpose of the secret manager, it provides navigation and it encourages users to register and begin using the platform.



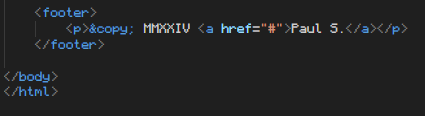
In the /head section I put the basic HTML prerequisites and standard rules (freecodecamp.org, 2022); the meta charset and viewport were properly defined to support HTML5 and mobile responsiveness, the globals.css stylesheet is being called through the Flask application (app.py), the JavaScript file is being called to handle the clickable hamburger menu on smaller screens (dev.to) and the favicon logo is included for brand identity.



The navbar provides seamless navigation across the website; the logo is displayed in the top left corner, the menu-toggle class, also known as the visuals for the toggle of the hamburger menu, was implemented for smaller screens, and the navbar links direct users to key pages: Home, Register, Login, Contact and About.



The hero section is the focal point of the page (w3schools, 1999); Flash messages are displayed at the top if there are any notifications, using Flask’s get\_flashed\_messages() (or simply: **flash()**) method (Flask, 2010). The main heading tells the core purpose of the application. A simple description tells the user what kind of data the platform can store, a “Get Started” button links to the registration page, encouraging new users to sign up and a five-star rating image adds visual credibility to the page. Besides this, A logo, an image and a header were some of the requirements for the main page.



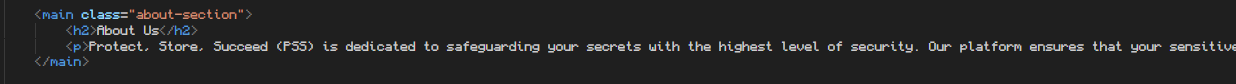
Lastly, the footer, with the current year in roman numerals, my name and the Unicode for the copyright symbol

**2.2 Contact & About (contact.html & about.html)**

The contact & about page are set up fairly simple. Up until the header, everything is the same as the homepage.



For the contact page, I’ve created a small non-functional contact-us form which doesn’t actually send information to anyone or anywhere.

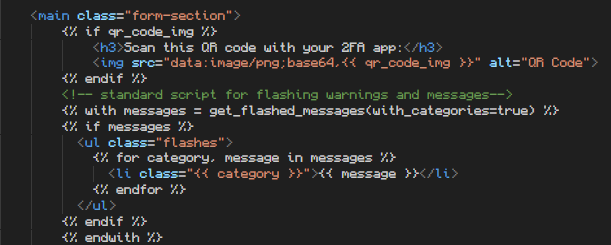


The main part of the about page simply shows text.

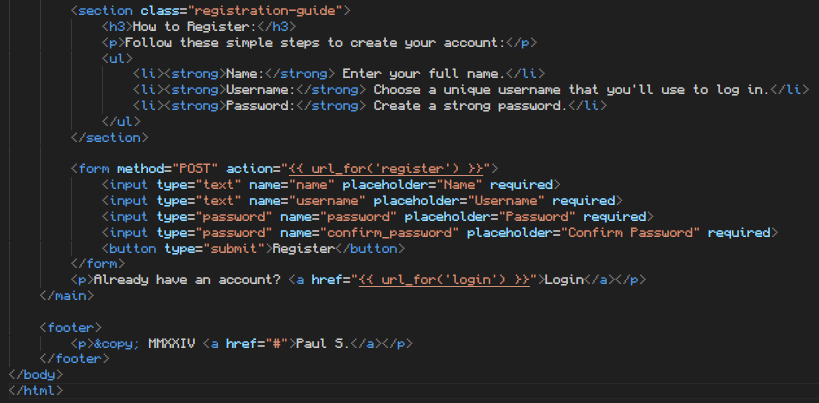
**2.3 Register (register.html)** **& Login (login.html)**

I knew that user registration was going to be a critical component. Before coding, I sketched a wireframe for the registration page, aligning it with the overall wireflow of the project. The registration page's purpose is to securely capture user credentials and register them into the system, including generating a Two-Factor Authentication (2FA) setup. The registration page also acts as an entry point to the system by enabling users to create accounts. Security was a priority from the outset, ensuring passwords are hashed and 2FA is integrated as an added layer of protection.

As with other pages, the head section establishes the basics for the HTML page and the navigation bar allows users to easily navigate between pages.



This part of the code was added to the file as one of the last things. When the page gets the notification that qr\_code\_img is being called, this part gets called. A QR code for an authenticator app will be generated from a generated base64 hex. Below the code is the standard template for flash messages.

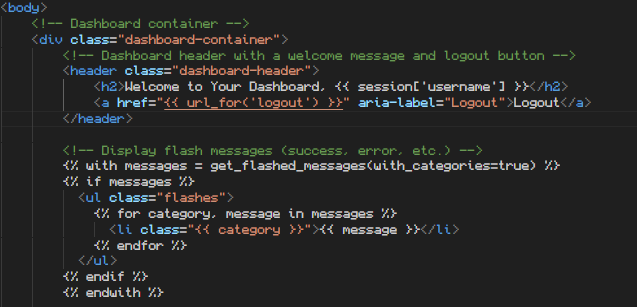


Before the QR Code, a guide is provided to walk users through the registration process. The form itself inclused the requirement for a name, a username, a password and confirming the password. There is also a link for users who already have an account to log in. the form uses POST method to send the data securely to the Flask backend for processing.

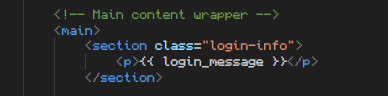
The footer remains consistent across the site. The login page follows a very similar structure to the register page, only having 2 input fields.

**2.4 Dashboard (dashboard.html)**

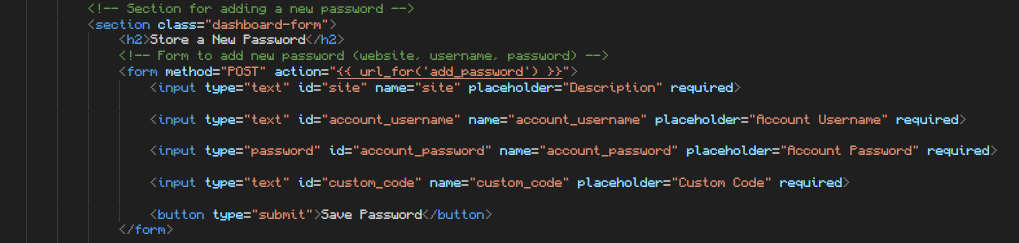
Code-wise, the dashboard is the most interesting HTML5 page. The dashboard page is the core of the secret manager project, where users interact with their stored secrets. Once a user successfully logs in and passes Two-Factor Authentication (2FA), they are directed to this dashboard. Here, users can securely store, manage, and search for their passwords or other sensitive data. Some key features include a welcome message with a “last login” message, a simple logout button, entire password management system, search functionality and even pagination.



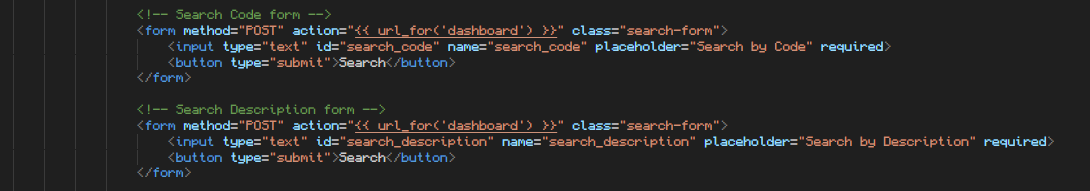
The head section remains the same. The header section displays a nice welcome message and the user’s username from the session, as well as a logout button that safely ends the session. Beneath these functions is the basic template for the flash messages.



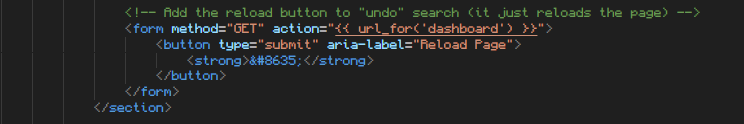
The above section shows the last login message, calculated and passed from the backend to provide the user with feedback on their last login time. Depending on their account state, the message could be a welcome for a new user or a time since the last login.



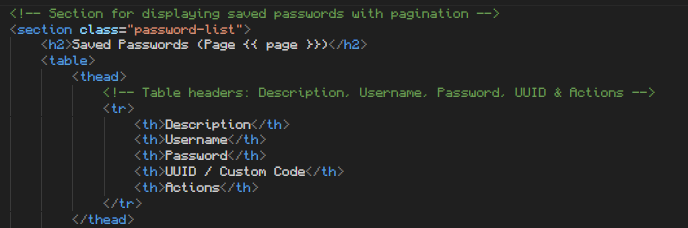
Here, users can store a new password using multiple fields that need to be filled out. On submission, the data is sent via POST to the Flask backend to be encrypted and stored.



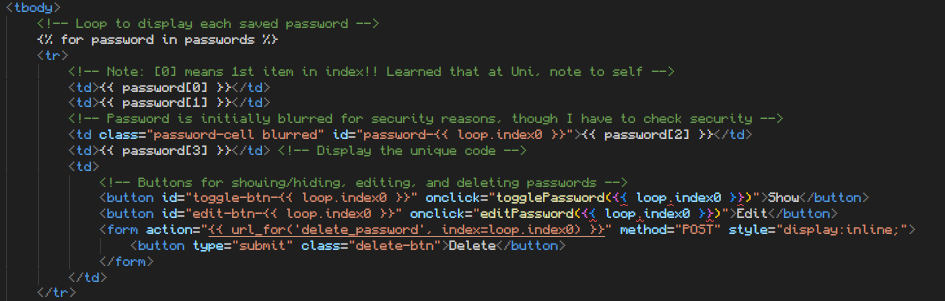
Users can also search for stored passwords. The first form allows users to search by custom code and the second form lets users search by description.



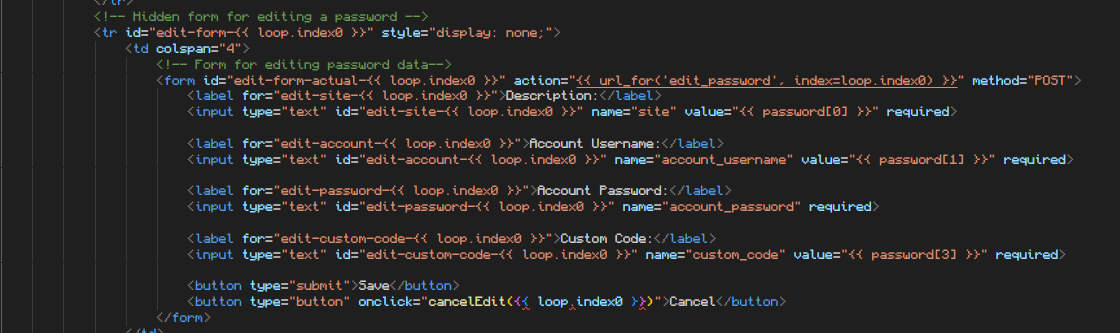
The reload button “clears” any active search and returns the user to the full list of passwords.



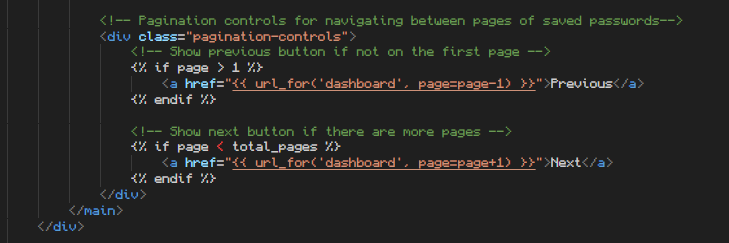
This section is the main part of the dashboard. With a table, we can list the 5 columns with the information we want the user to see. The page function calls the backend to check on which page the user is right now (page 1/.. of total pages of passwords / tables).



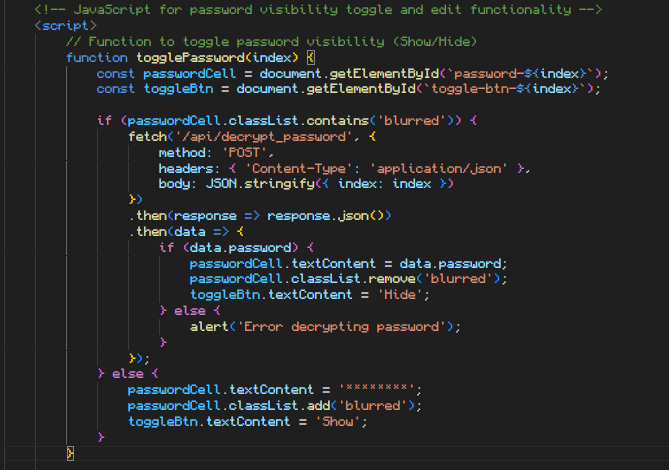
The code searches through the indexes of the passwords.csv file: Flask iterates over the stored passwords and displays them in the table. Because the 0th index is the description and the 1st index is the Username I could quickly conclude that the 2nd index should be blurred, since I structured the passwords.csv file as such that the password comes 3rd in the column. The edit button triggers a new form thanks to the JavaScript. The Flask application gathers the responses from the form, encrypts the new password and pushes this to the application. Thanks to AJAX requests, passwords are decrypted dynamically without refreshing the page or leading the user to a new URL (like ../dashboard/1, ../dashboard/2 etc).



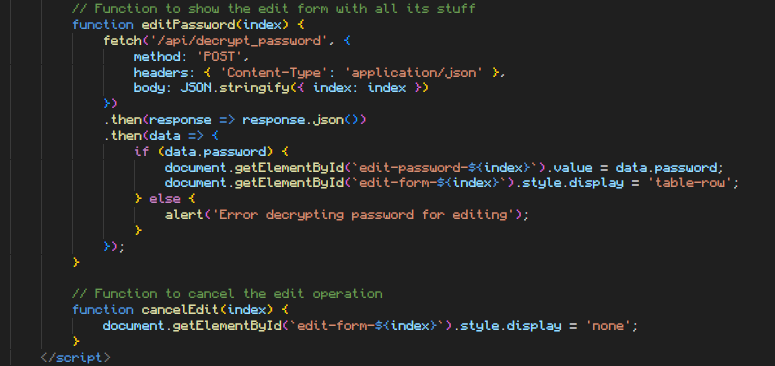
As such, the above form gets created when pressing the edit button. The description, account username and custom code all don’t have to be blurred or encrypted and they can easily be edited through directly editing their strings in the given index. The account password has multiple steps before being pushed to the passwords.csv file.



Pagination controls allows users to navigate between pages. If the user is on a page number bigger than 1, show the previous button. If the user is on a page number less than the total amount of pages (of which the total amount of pages must at least be 2), the next button appears.



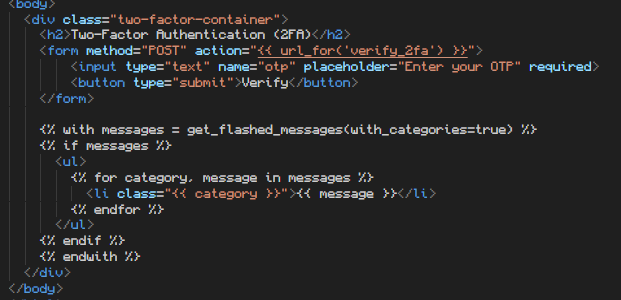
When the show button is pressed, the application calls the togglePassword function. The passwordCell variable gets the value of the encrypted password. If the passwordCell has the blurred attribute, the password itself gets decrypted by the backend, converted to a JSON string (as JSON is a common use of exchanging data to/ from a web server (w3schools, 1999)), then the decrypted password gets shown and the Show button transforms into a Hide button. When the process is done, the remaining password in its decrypted form gets blurred.



When editing a password, again the password gets decrypted first and shown in the newly made form by the edit form. Cancelling the edit operation simply “removes” all the styles and buttons associated with the edit functions.

**2.5 2FA (2fa.html)**

The 2FA HTML file is built fairly simple, as the backend handles all the processes and the HTML only shows the form for filling in the OTP code, a small button and a nice text signalling the user to open their 2FA app and finding their OTP code.



**2.6 CSS (globals.css)**

In this section, I will document the CSS file used for the Secret Manager project, which is responsible for styling all the web pages, including responsive design, form styling, navigation bars, and the dashboard. I structured the CSS file to ensure a consistent design across the website, using a root colour scheme and predefined variables for font and background. Additionally, media queries were implemented to ensure the website is responsive across all devices, including desktops, tablets, and smartphones. The file follows a logical structure: root variables for the primary, secondary and background colours along with fonts, which stay consistent across the entire site, global styling for basic body settings and text colours for all elements and component-specific styles, such as custom styling for headers, footers, the dashboard etc.

Screenshotting the code here would take up a lot of time and space, so I will globally talk about the structure. The :root section defines all the reusable CSS variables (such as colours and fonts) to maintain a consistent theme across the site. This makes it easy to change the colour scheme or font globally. The primary colour is a little purple, the secondary colour a bit darker purple, the background a dark colour and the text full white. The base styles for the body include default settings for margins, padding, background colour, and the font family inherited from the root variables. This ensures that the website uses a clean, minimal base styling. The header and navigation bar are styled to remain sticky at the top of the page, with navigation links spaced out and highlighted when hovered over. Flexbox is used to ensure proper alignment of elements and hover effects are included to give feedback to users as they interact with navigation links. Forms are used on the login, registration, and contact pages. These forms are centered on the page and styled with appropriate padding and borders for a modern look. Inputs and buttons are styled with rounded corners, solid borders, and smooth transitions. On hover and focus, elements change colours to improve user experience and accessibility. The dashboard features styles for listing passwords, forms, and pagination controls, ensuring that they are easy to interact with and visually appealing. The dashboard has a dark background, consistent with the overall theme, and a padded, centered layout for better readability. Forms in the dashboard are optimized for fast input, and buttons are colour-coordinated for visual hierarchy. The password list within the dashboard is styled for readability, with alternating table row colours, a smaller font size for passwords, and additional actions (like editing and deleting). Passwords are blurred initially for security reasons, and only revealed when the user interacts with the table. Media queries ensure that the design remains functional and visually appealing on smaller screens, including tablets and mobile phones. The navigation links collapse into a menu on smaller screens, with flex-direction changed for vertical stacking. This ensures the site works across devices without compromising on user experience. With reusable variables, clear component-based styling, and responsive design, this CSS file ensures a smooth, modern, and responsive experience across all pages.

CSS methods from W3schools (1999)

**3 Flask Modules Documentation**

The Secret Manager project is structured using the Flask framework, which is a lightweight and modular web application framework in Python. Flask is ideal for small-to-medium-sized applications and follows a micro-framework approach, meaning it provides essential functionality and allows flexibility in extending features through various libraries. In this project, Flask handles the routing, user authentication, form submissions, session management, and integration with Jinja2 templates for rendering HTML pages dynamically. The application structure includes the main Python script (app.py), HTML templates, static files (CSS, JavaScript, and images), and CSV files for storing user and password data. App.py contains the core logic of the Flask application, including routes, form handling, user authentication, and file operations. Users.csv and passwords.csv store user data and secrets, respectively, using simple CSV file manipulation for persistence. In Flask, each web page or operation is mapped to a route, where functions handle the logic of rendering templates and processing user input. This design separates the front-end (HTML/CSS) from the back-end logic (Python) while using the Jinja templating engine for dynamic content injection.

**3.1 Routes and Functions**