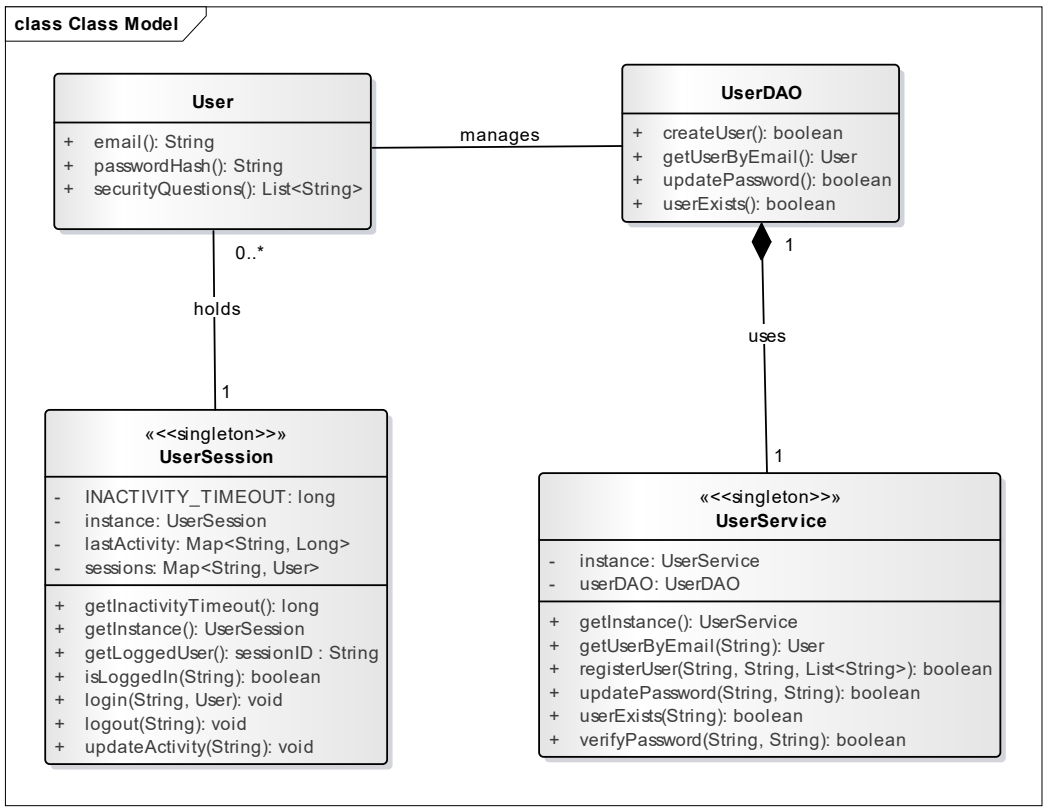
MyPass Report

Implementation

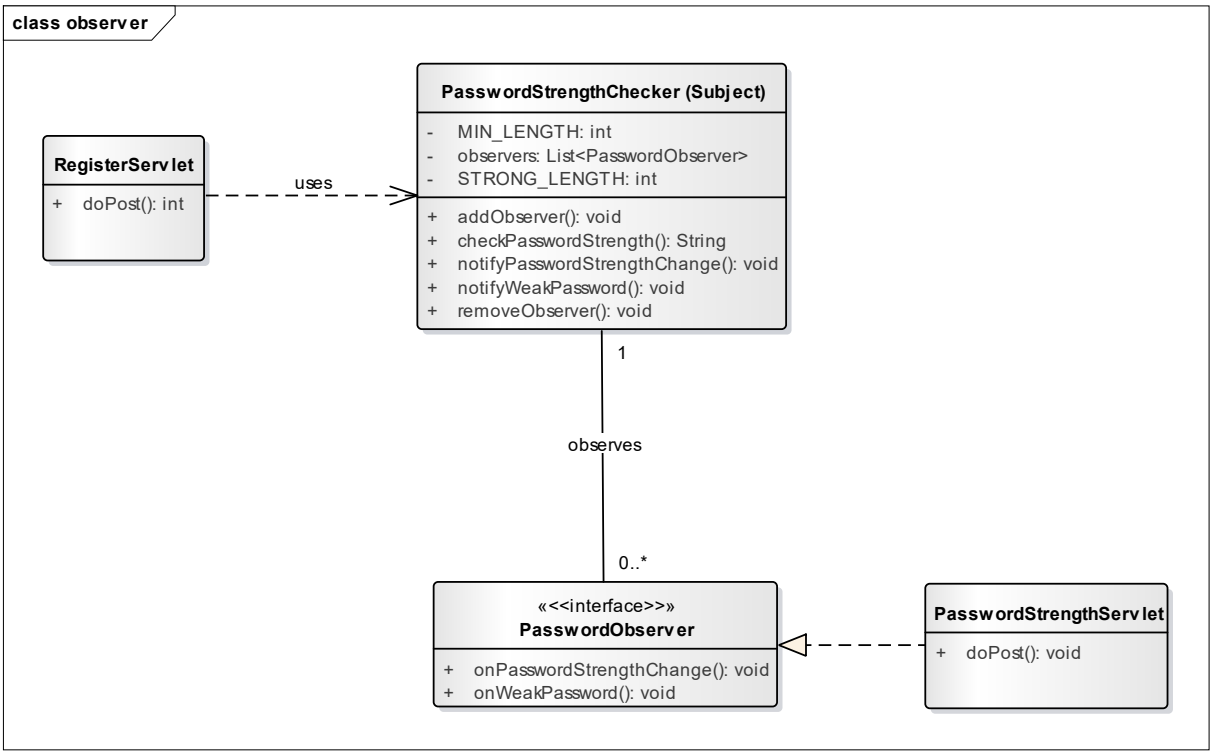
User Authentication and Encryption:
Implement the Singleton pattern to manage the user's session securely.



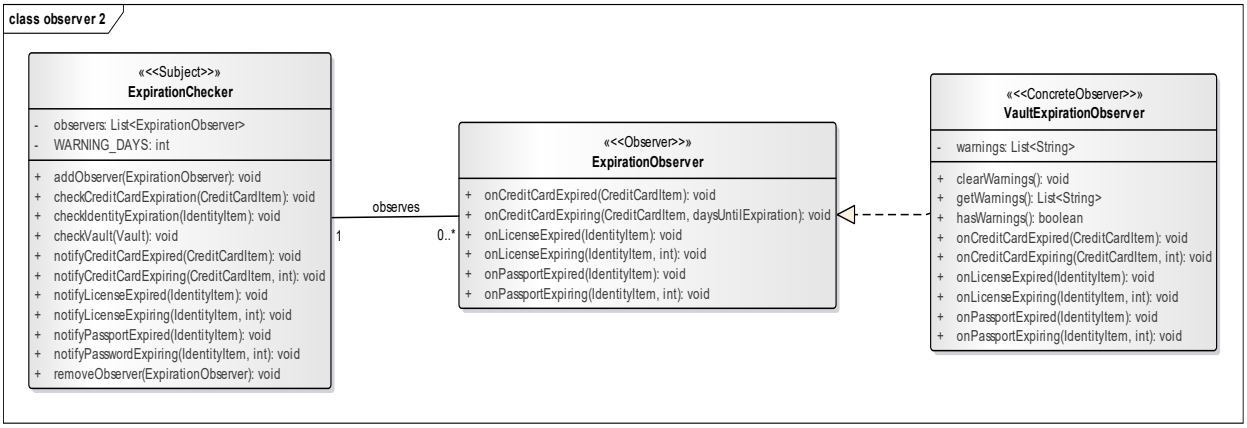
The implementation of the Singleton pattern is found in **UserSession** and **UserService** classes. **UserSession** class acts as a centralized, manager that ensures only one instance controls all active user sessions, mapping each HTTP session IDs to a user while tracking last activity timestamps to enforce automatic session locking after periods of inactivity, preventing unauthorized access. **UserService** is also implemented as a Singleton, and it provides a unified, secure interface for user-related operations such as password hashing, verification, and account creation. It uses BCrypt to encrypt passwords before storage, ensuring that no plaintext credentials are ever saved or exposed. Together, these two Singleton classes provide consistent, secure, and unified control over session management, authentication, and password encryption across the entire application.

Password Storage and Management:

Apply the Observer pattern to notify users in the events of weak password, credit card expiration, passport expiration, license expiration, etc



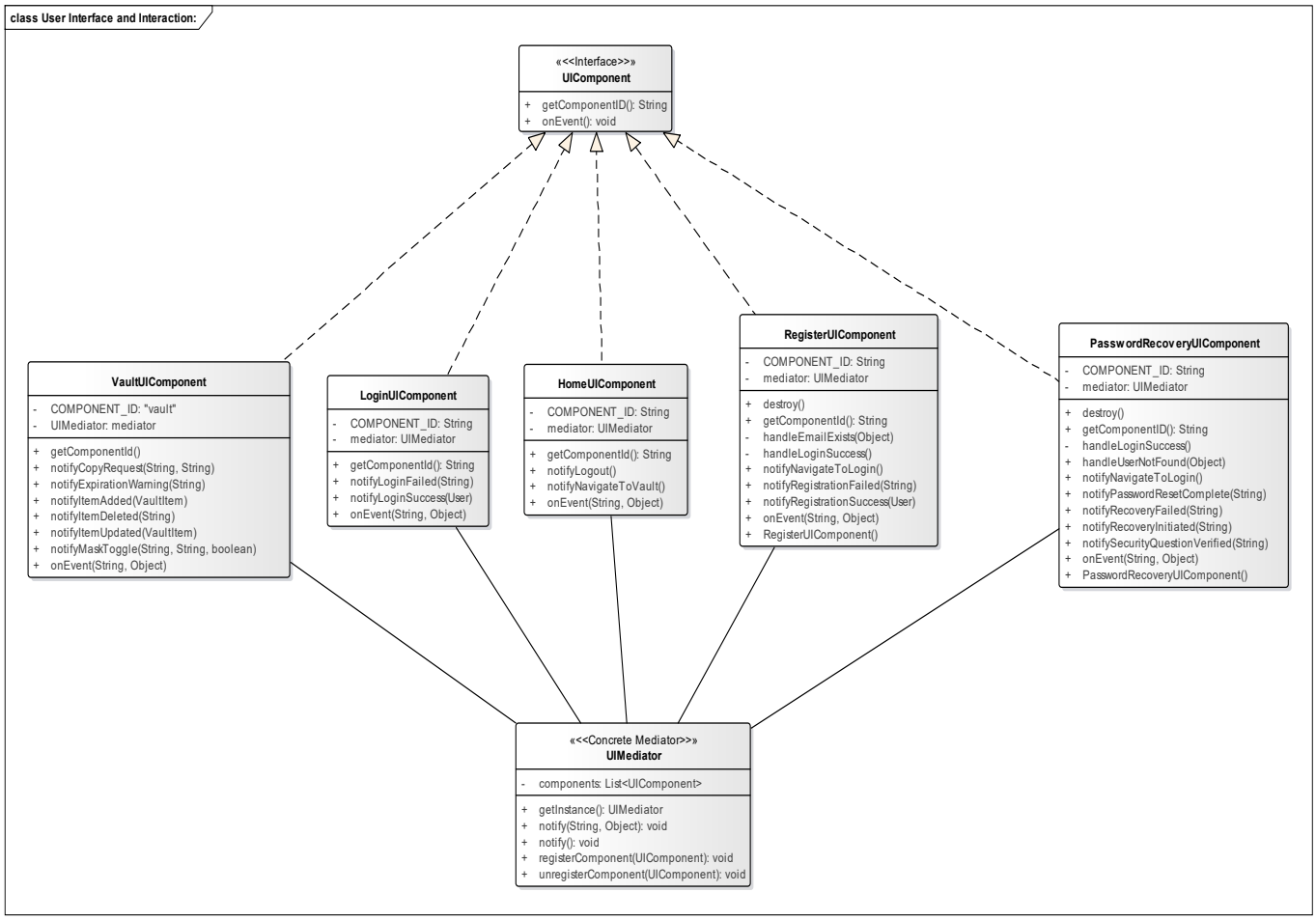
The implementation of password storage and management is distributed across several classes. For password validation, the Observer pattern is applied via **PasswordStrengthChecker** (the Subject) and **PasswordObserver** (Observer interface). **PasswordStrengthChecker** evaluates password strength based on length, character variety, and complexity, maintains a list of registered observers, and notifies them whenever a password is weak or when its strength changes. **PasswordObserver** interface defines the methods that observers must implement, decoupling password evaluation from any specific UI or notification mechanism. A **ConcreteObserver**, implemented as a class inside **PasswordStrengthServlet**, captures password strength events. **RegisterServlet** uses **PasswordStrengthChecker** during the registration process to check passwords and provide warnings to users.



For expiration **ExpirationChecker**, **ExpirationObserver**, and **VaultExperiationObserver** extend the observer pattern to track time-sensitive credentials such as credit cards, passports, and driver licenses. **ExpirationChecker** is the subject, and it periodically scans the vault for items approaching or past their expiration dates and notifies all registered observers of any relevant events. **VaultExpirationObserver** acts as a concrete observer, collecting these warnings for display in the user interface or further processing. By decoupling the expiration detection logic from the UI, the system ensures users are consistently notified about expiring items without tightly coupling business logic to presentation. Combined with the password observer subsystem, these classes collectively fulfill the requirement for robust password storage and management.

User Interface and Interaction:

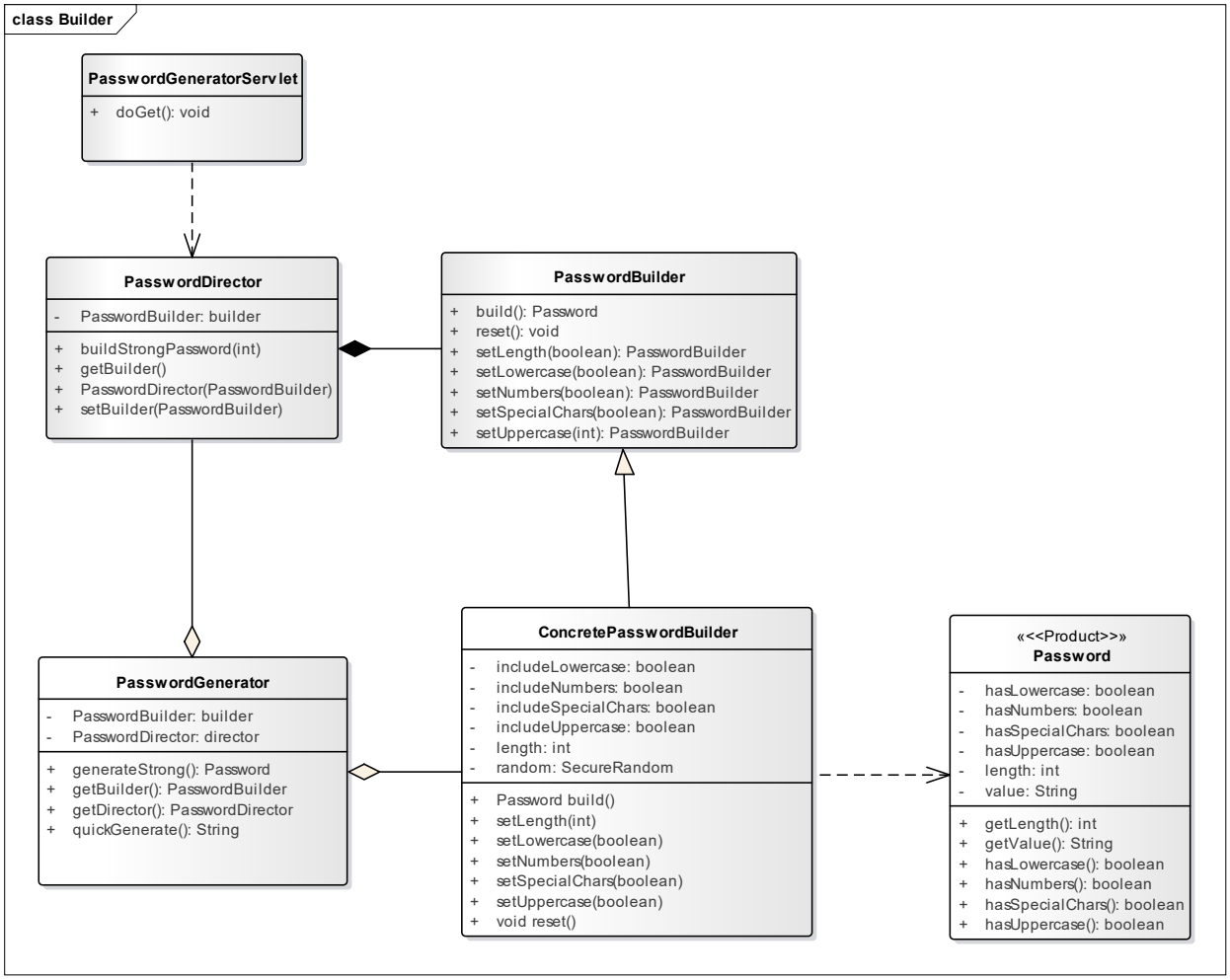
o Implement the Mediator pattern to manage communication between various UI components.



The Mediator pattern in this web application is implemented using the **UIMediator** (Concrete Mediator), the **UIComponent** interface (Mediator-facing Colleague interface), and the individual UI screens such as **VaultUIComponent**, **HomeUIComponent**, **LoginUIComponent**, **RegisterUIComponent**, and **PasswordRecoveryUIComponent** which are the colleague classes. According to the Mediator pattern, the mediator defines how colleague objects communicate, and the concrete mediator coordinates their interactions. In this system, each UI component implements the **UIComponent** interface and registers itself with **UIMediator**, meaning the mediator knows and maintains all its colleagues. This structure fully decouples UI components, prevents cross-dependencies, centralizes UI communication logic, and cleanly follows the Mediator pattern: colleagues know the mediator, but never know or talk to each other.

Password Generation:

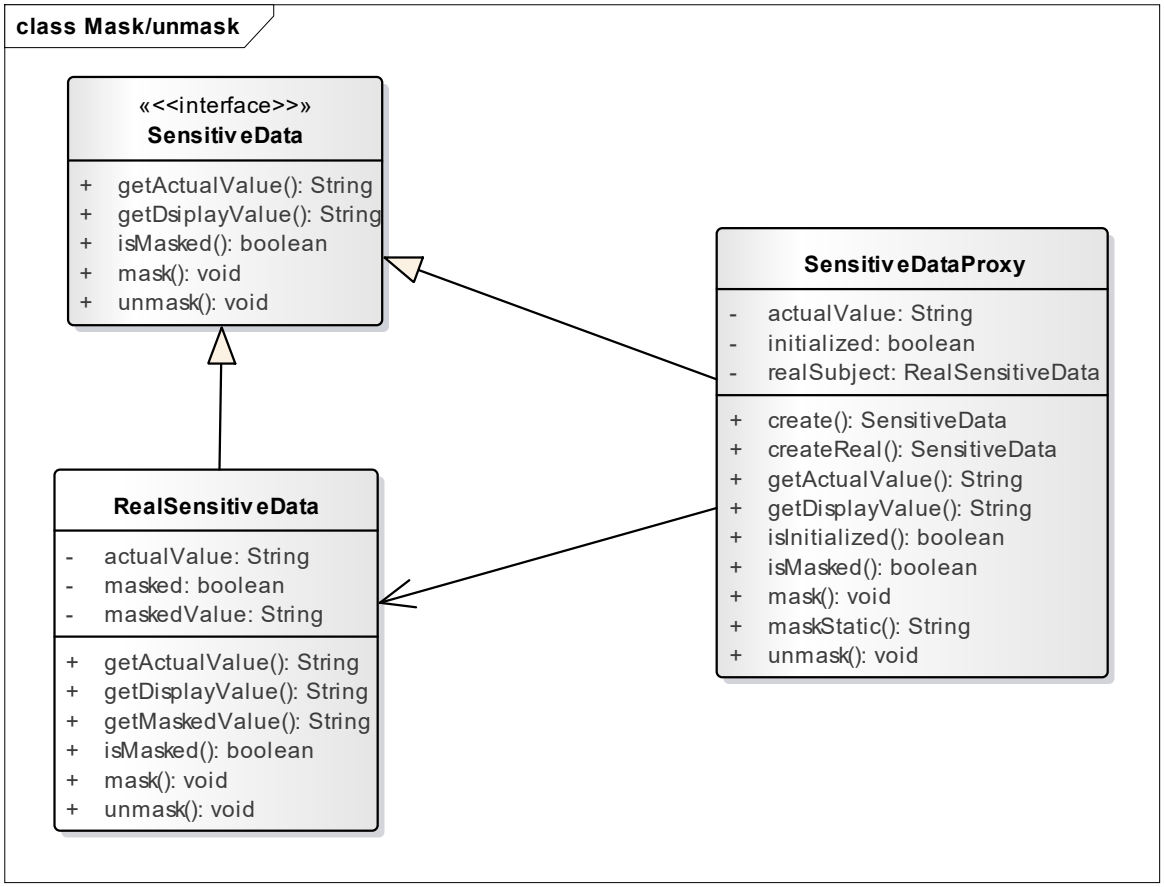
o Apply the Builder pattern to create complex passwords with specific requirements (length, complexity).



The Builder pattern in the application is implemented through the **PasswordBuilder** interface, **ConcreatePasswordBuilder**, **PasswordDirector**, and the Password class to create complex passwords with specific requirements. The Password class serves as the product, representing the final password and its configuration. The **PasswordBuilder** interface defines abstract steps for constructing a password, while **ConcretePasswordBuilder** implements these steps, maintains the internal state of the password parts, and assembles the final Password object. The **PasswordDirector** orchestrates the construction process by instructing the builder which parts to include for different password types, such as simple or strong passwords. The client, for example **PasswordGeneratorServlet**, creates the director and builder, requests a specific password configuration, and retrieves the final Password product.

Data Mask and Unmask:

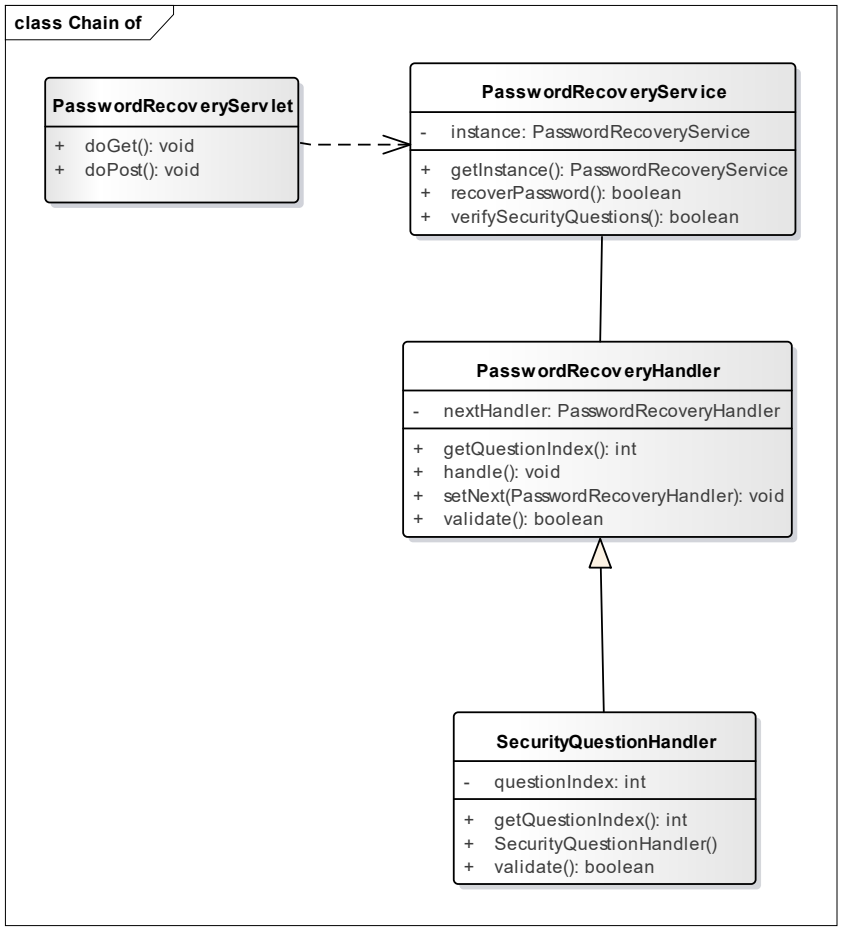
o Implement the Proxy pattern to mask and unmask sensitive data



The Proxy pattern in this web application is implemented through the **SensitiveDataProxy, SensitiveData**, and **RealSensitiveData** classes to manage access to sensitive information such as passwords or card numbers. The subject is defined by the **SensitiveData** interface, which specifies a common set of operations that can be used by any client. The real subject encapsulates the actual sensitive data and manages its internal state, including both masked and unmasked representations. The proxy, **SensitiveDataProxy**, maintains a reference to the **RealSensitiveData** and provides the same interface as the Subject, allowing it to be substituted for the real object anywhere the UI or other components expect a **SensitiveData** instance. The proxy controls access to the **RealSubject**, supports lazy initialization, and adds optional features such as logging or access checks, while the UI components rely on the Proxy to handle masking and unmasking without managing sensitive data logic themselves.

Master Password Recovery:

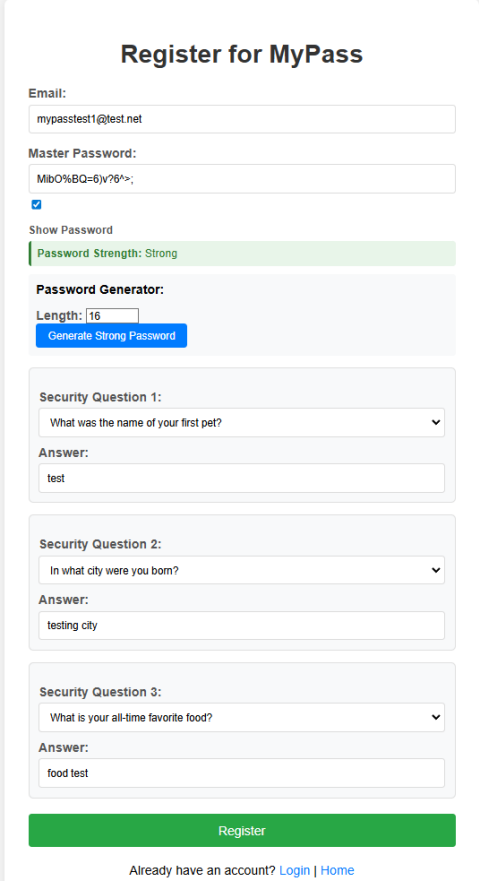
o Apply the Chain of Responsibility pattern to create a secure process (using three security questions to build the chain) for recovering a forgotten master password



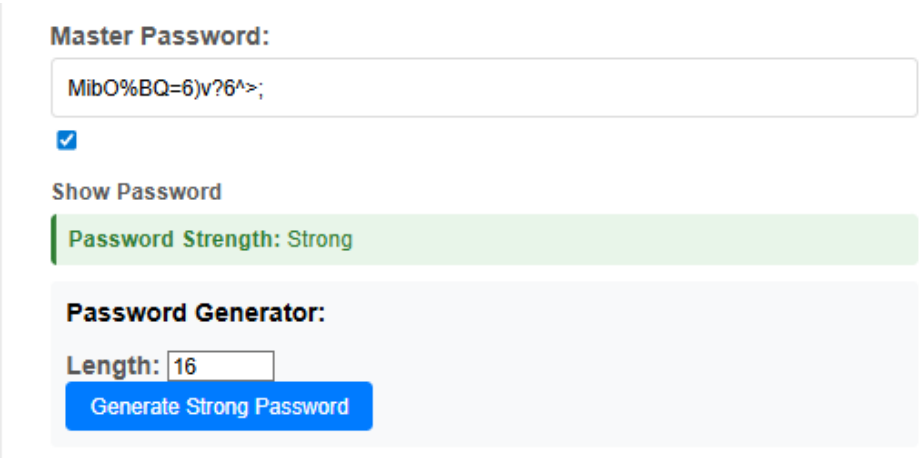
The master password recovery functionality implements the chain of responsibility pattern to securely validate a user before allowing a password reset. Each security question is handled by a separate ConcreteHandler (**SecurityQuestionHandler**) responsible for verifying one specific answer. These handlers are linked together in a chain, with each handler forwarding the request to its successor only if its own validation succeeds. The abstract Handler **PasswordRecoveryHandler** defines the interface for handling requests and maintaining the successor link, while the Client (**PasswordRecoveryService** and **PasswordRecoveryServlet**) constructs the chain and initiates the recovery process. By using the chain, the system ensures that all three security questions must be answered correctly to reset the master password, encapsulating the multi-step verification logic and providing a secure, extensible recovery workflow.

User-Interface Screen Shots

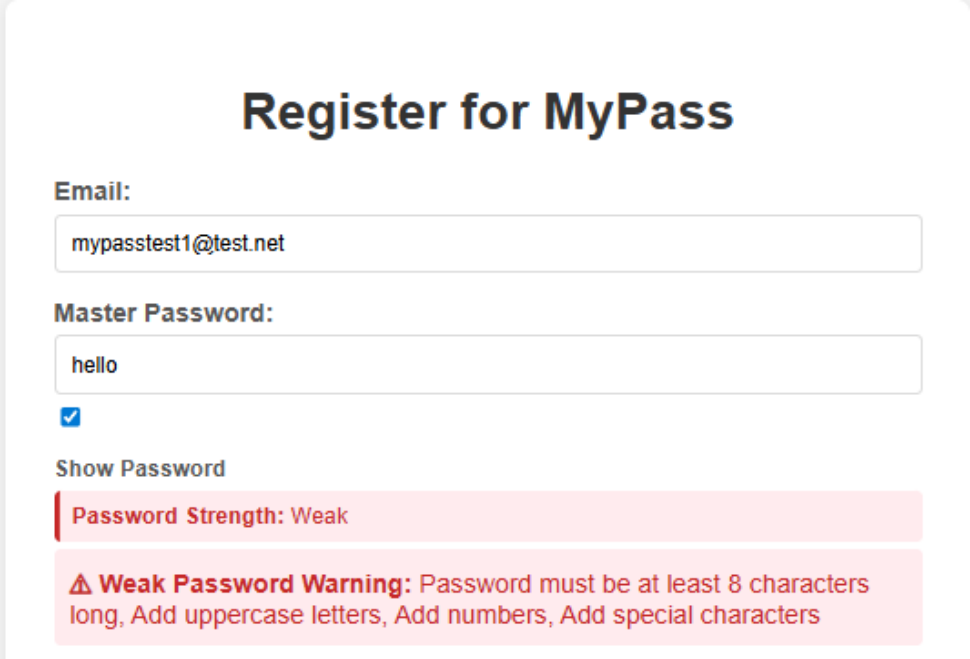
Allow user to register an account (email address as username and a master password and three security questions)



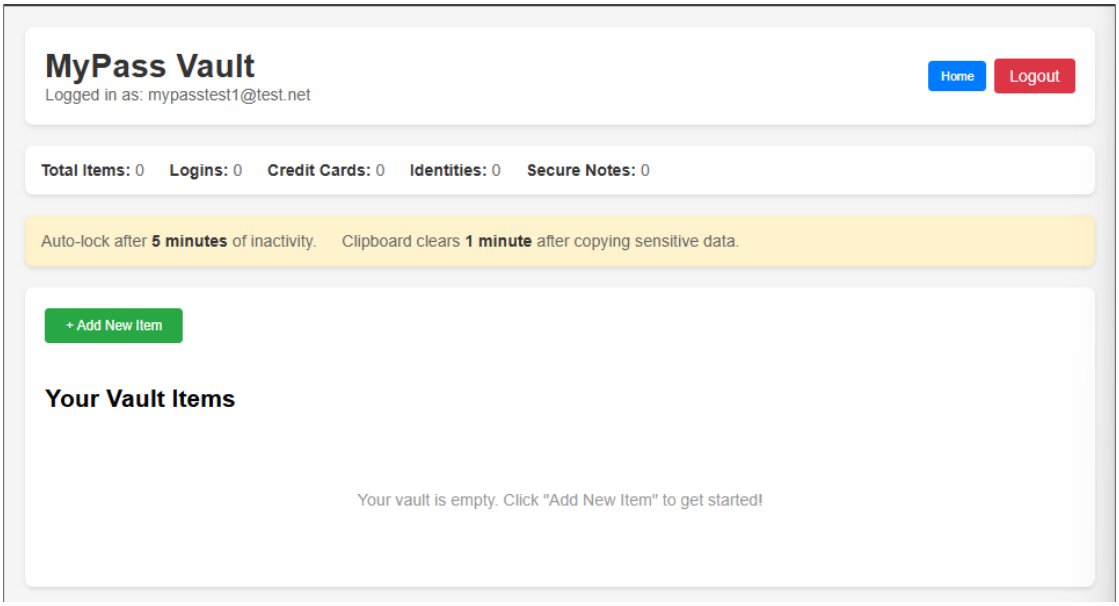
• Suggest strong password (password generator)



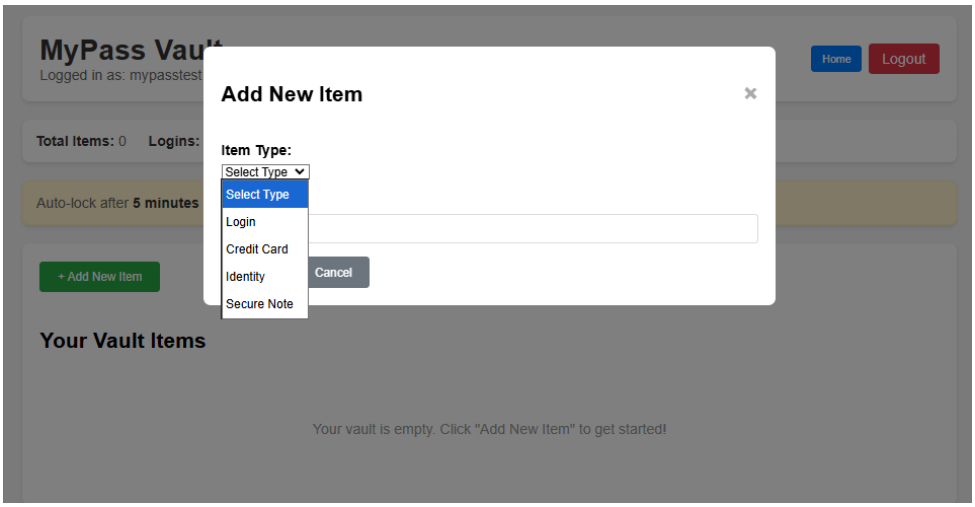
• Weak master password warning



• Once logs in, the user will have access to all sensitive data saved in the vault.

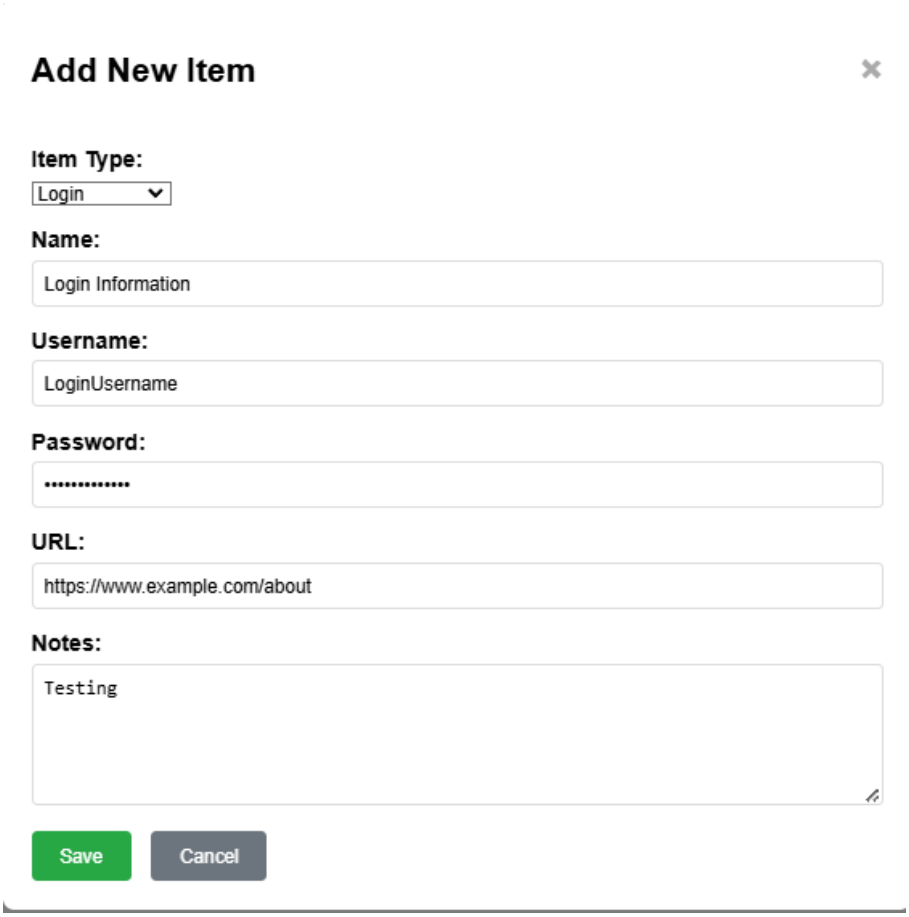


• MyPass has built-in data type in the vault, including Login, Credit Card, Identity, Secure Notes.

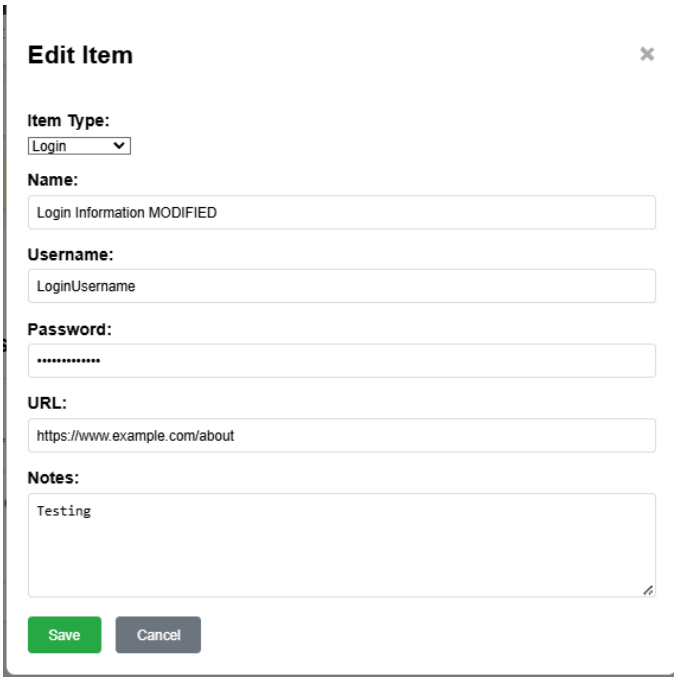


• MyPass allows user to create/modify/delete items in the vault.

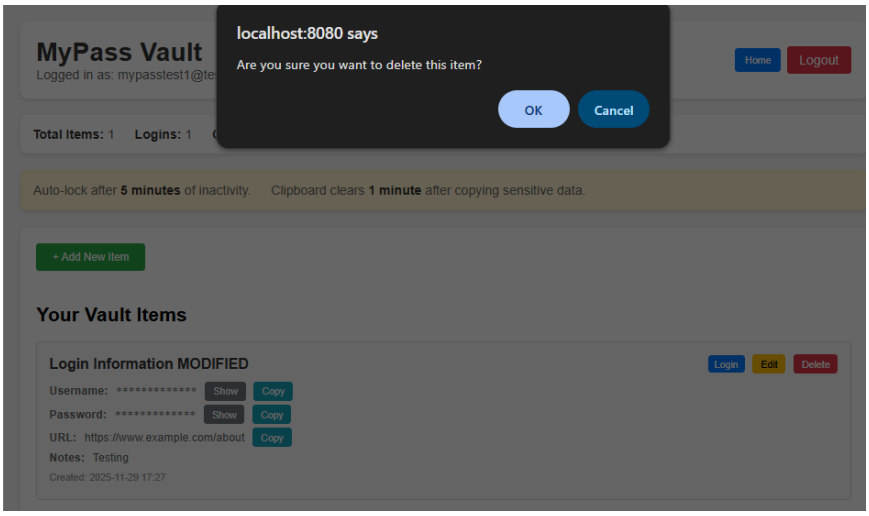
Creating Item



Modifying Item

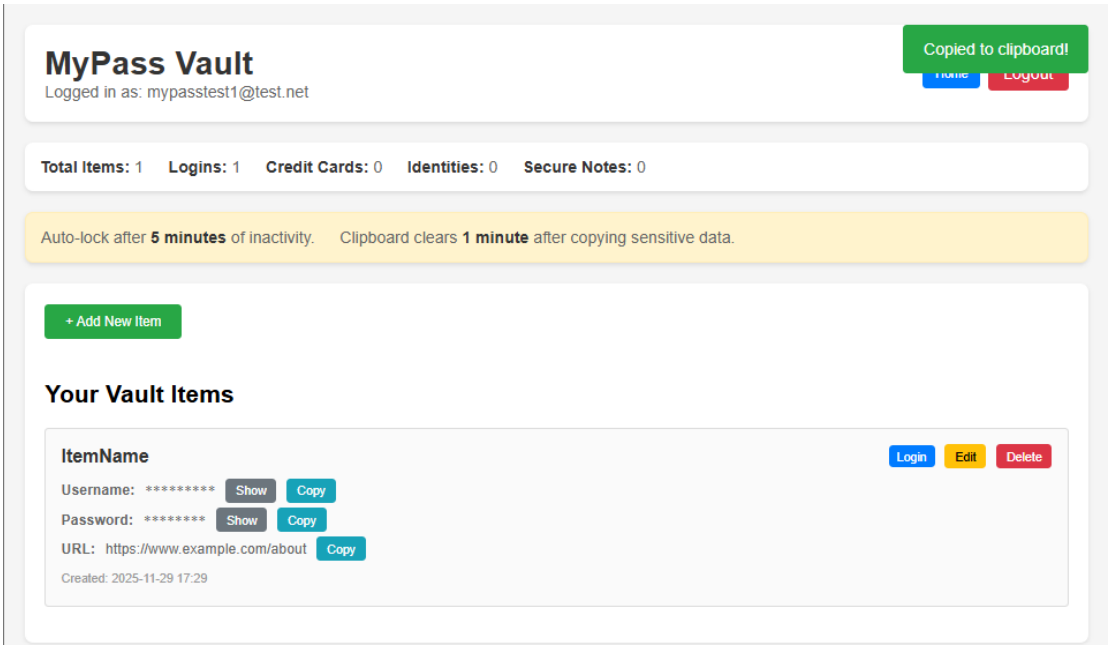


Deleting Item



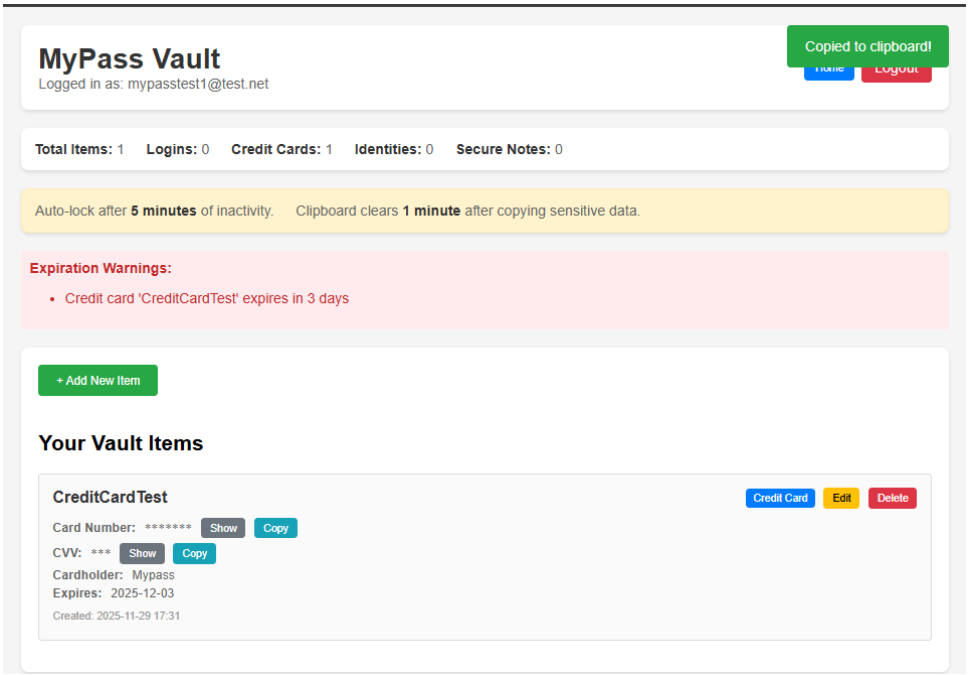
• MyPass allows user to easily copy username/password and URL in Login data type

Whenever user presses copy on username/password and URL in Login Data type copied to clipboard message is shown:



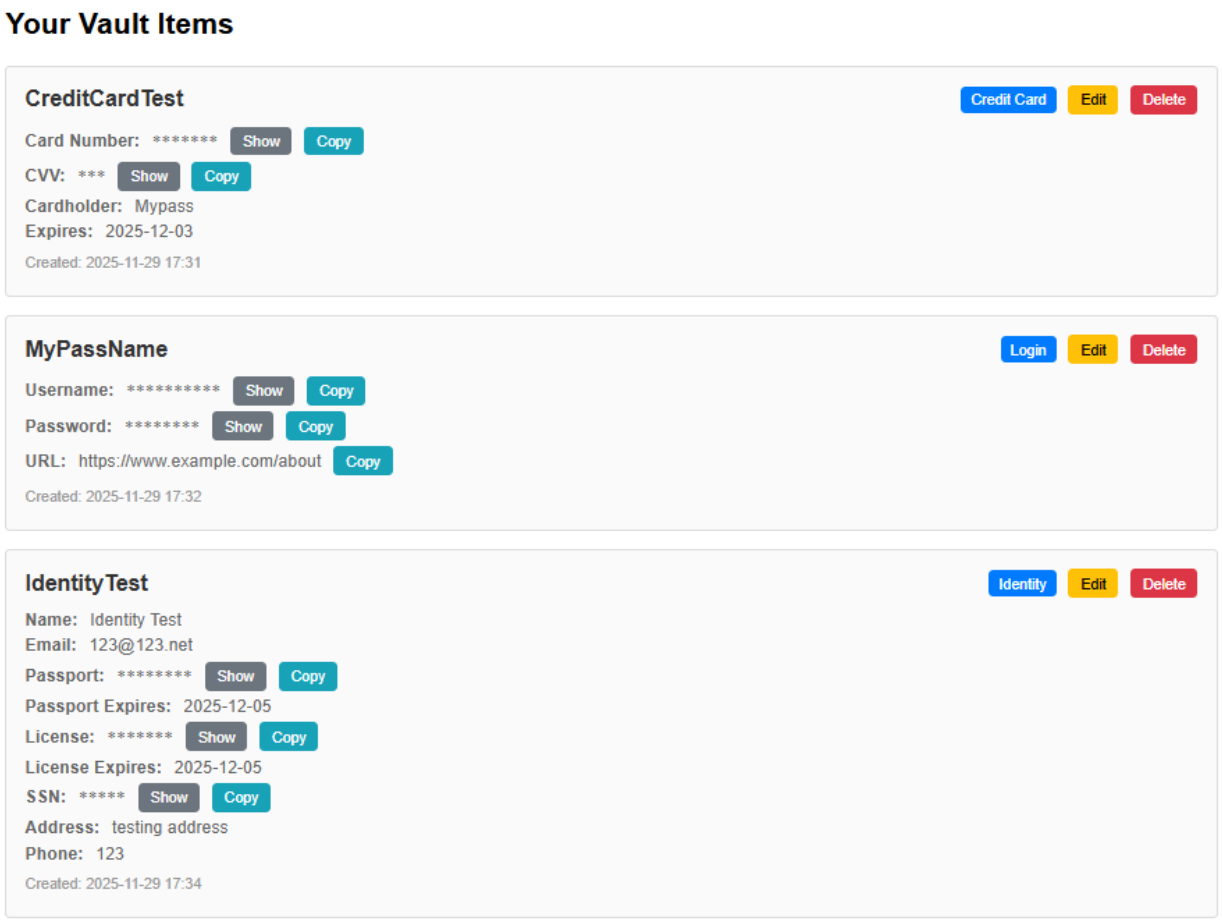
• MyPass allows user to easily copy credit card number and CVV in credit card data type

When a user presses copy on card number and CVV in credit card data type, it is shown as copied to clipboard on top right corner.

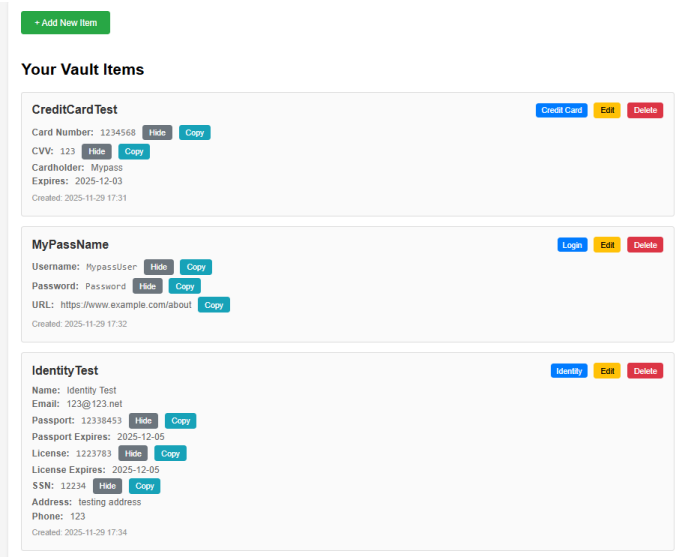


• Sensitive data such as username, password, credit card number, CVV, passport number, license number, social security number must be masked

Data masked

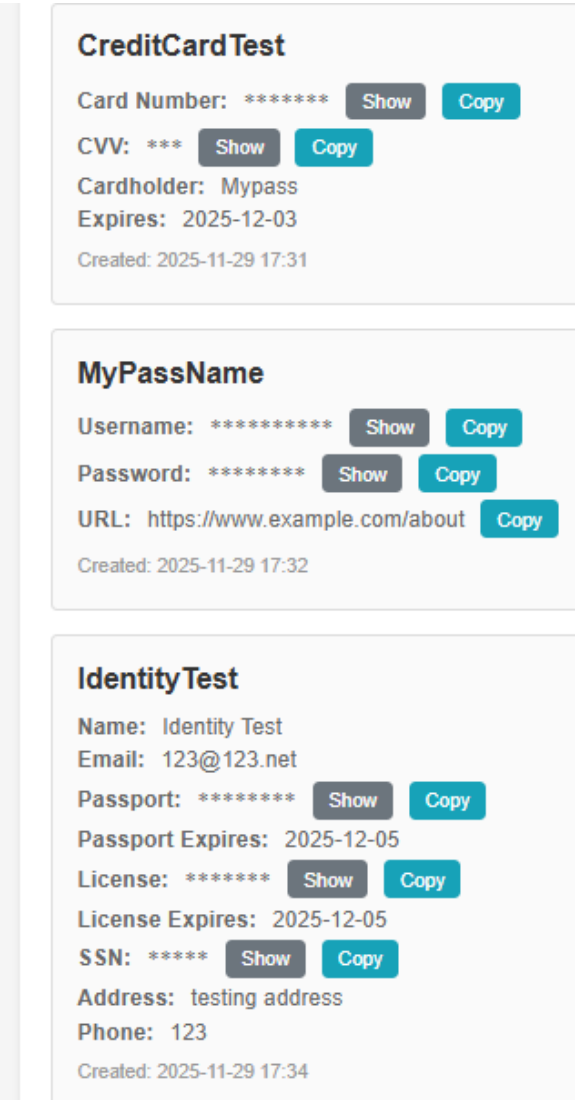


Data unmasked:

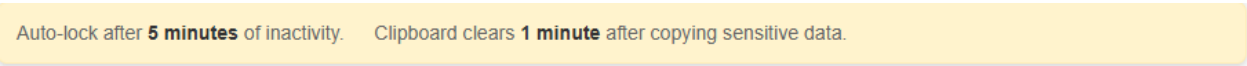


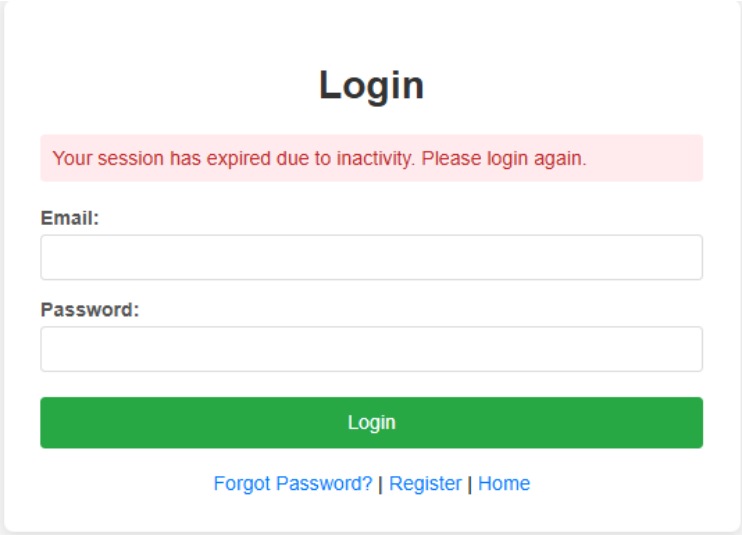
• All masked data must be given an option to unmask.

“Show” button unmasks data:

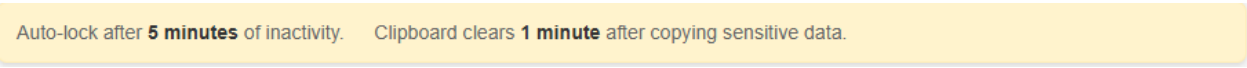


• Auto lock after x mins of inactivity

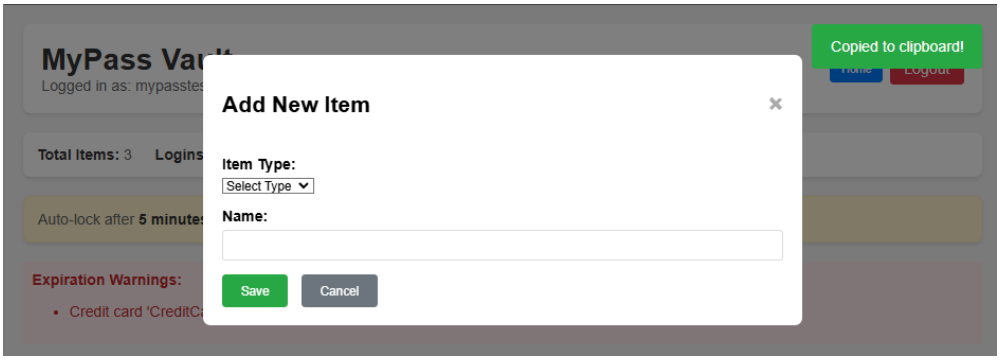


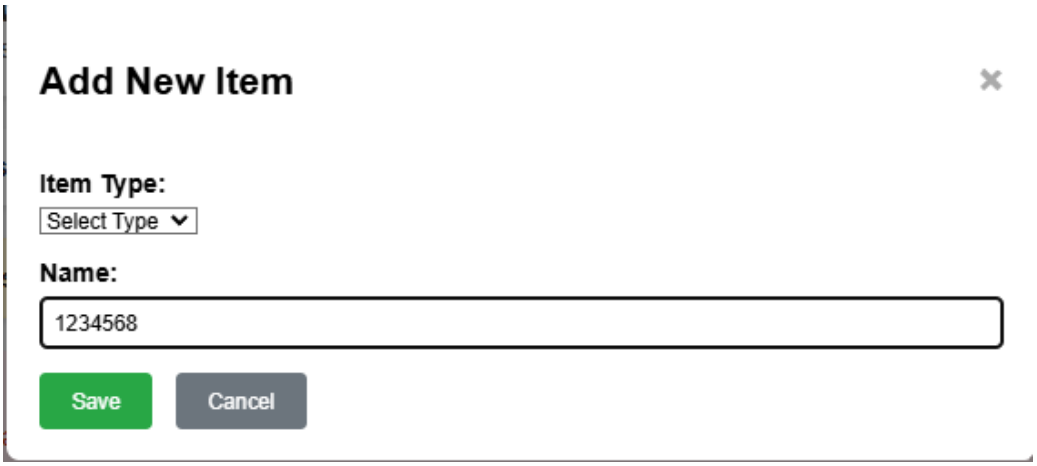


• Auto delete copied sensitive data in clipboard after x mins

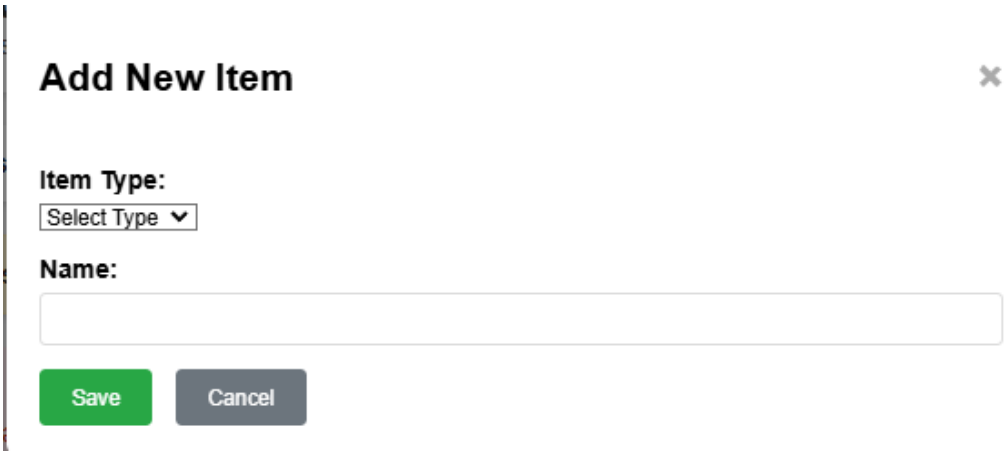


Copying sensitive data works when initially doing it:





After 1 minute it doesn’t allow to paste:



Database Schema

Users Table

Column Name	Type	Constraints	Description
Email	VARCHAR(255)	Primary Key	Unique user email
Password_hash	VARCHAR(255)	NOT NULL	Hashes master password
Security_question_1	VARCHAR(500)	NULL	Answer to security question #1
Security_question_2	VARCHAR(500)	NULL	Answer to security question #2
Security_question_3	VARCHAR(500)	NULL	Answer to security question #3
Created_date	TIMESTAMP	Default current_timestamp	When account was created
Last_modified	TIMESTAMP	Default current_timestamp	Last account update

Vault_Items Table

Column Name	Type	Constraints	Description
Id	VARCHAR(36)	Primary key	UUID for each vault item
User_email	VARCHAR(255)	Not Null, FK users(email_, on delete cascade	Owner of the vault item
Item_type	VARCHAR(50)	NOT NULL	Type of item (Login, card, identity, note)
Name	VARCHAR(255)	NOT NULL	User-friendly label
Created_date	TIMESTAMP	Default Current_timestamp	When the item was created
Modified_date	TIMESTAMP	Default Current_timestamp	When item was last updated

Login_Items Table

Column Name	Type	Constraints	Description
Item_id	VARCHAR(36)	Primary Key, FK is vault_items(id), On delete cascade	Linked vault items ID
Username	VARCHAR(500)	NULL	Login username
Password	VARCHAR(500)	NULL	Login Password
url	VARCHAR(1000)	NULL	Website URL
Notes	TEXT	NULL	Additional Notes

Credit_Card_Items Table

Column Name	Type	Constraints	Description
Item_id	VARCHAR(36)	Primary key, FK vault_items(id), on delete cascase	Linked vault item
Card_number	VARCHAR(50)	NULL	Card Number
Cardholder_name	VARCHAR(255)	NULL	Cardholder name
Cvv	VARCHAR(10)	NULL	CVV codde
Expiration_date	DATE	NULL	Expiration date
Notes	TEXT	NULL	Notes

Identity_Items Table

Column Name	Type	Constraints	Description
Item_id	VARCHAR(36)	Primary Key, FK vault_items(id), on delete cascase	Linked vault item
First_name	VARCHAR(255)	NULL	First Name
Last_name	VARCHAR(255)	NULL	Las name
Passport_number	VARCHAR(100)	NULL	Passport Number
License_number	VARCHAR(100)	NULL	Driver’s license
Social_security_number	VARCHAR(50)	NULL	SSN
Address	VARCHAR(500)	NULL	Physical address
Phone	VARCHAR(50)	NULL	Phone number
Email	VARCHAR(255)	NULL	Identity email
Notes	TEXT	NULL	Notes

Secure_Note_Items Table

Column Name	Type	Constraints	Description
Item_id	VARCHAR(36)	Primary Key, FK vault_items(id), on delete cascade	Linked vault item
Content	TEXT	NULL	Note content