CIS 476 Final Project Presentation

By Vlad Nitu

Introduction

- MyPass is a system that allows users to securely store sensitive information
- This includes credit cards, identities, site logins, and secure notes
- It is a web-based application
- Sensitive information is masked and encrypted
- Users can also use the password generator tool to create strong passwords based on their criteria
- The program also warns users of security risks such as weak passwords and expired items

Main Features

- Registration and login with recovery questions
- Auto lock after inactivity
- Ability to create, edit, and delete vault items
- Ability to mask and unmask sensitive fields
- Customizable password generate
- Tech Stack
  - Python backend
  - Flask server
  - SQLlite + SQLAlchemy
  - Flask-Bcrypt for password hashing

Singleton Pattern

- Implemented in SessionManager class
- Role
  - Ensures single object handles authentication
  - Enforces auto-lock
- Centralizes session logic
- Enhanced Security

Builder Pattern

- Implemented in PasswordBuilder and PasswordDirector
- Stores the user configurations for the password
- Director configures builder and calls build()
- Supports complex password options in a clean way
- Keeps logic flexible

Proxy Pattern

- Implemented in SecretFieldProxy
- Used by:
    - LoginItem: password
    - CreditCardItem: card number, CVV
    - IdentityItem: passport, license, SSN
- Methods:
    - Masked(): returns masked string
    - Unmask() returns original string value
- Encapsulated masking logic
- Prevents direct exposure of sensitive values
- Supports safe behavior of show, hide, and copy

Observer

- Implemented in:
  - Observer → abstract class
  - InAppNotificationObserver → concrete class
  - NotificationSubject → defines subjects for observers
- Checks for weak master password, weak login passwords, expiration logic
- Uses notify() method to send messages to observers
- Messages shown on dashboard

Mediator

- Implemented in DashboardMediator class
- Creates NotificationSubject and InAppNotificationObserver
- Runs evaluate_user() and returns a dict of notifications
- Keeps dashboard route simple
- Decouples the UI view from notification details

Chain of Responsibility

- Implemented in RecoverHandler → abstract class
  - Holds reference to next handler
- Concrete handlers:
  - Question1Handler
  - Question2Handler
  - Question3Handler
- Linked by Question1Handler → Question2Handler → Question3Handler
- Each handler checks a security question
- If failure, the recovery request does not go through
- Question must be correctly answered to pass to next handler
- Easy to add/remove questions with minimal code changes

References

- https://www.sqlalchemy.org/
- https://sqlite.org/
- https://flask.palletsprojects.com/en/stable/
- https://www.python.org/
- https://bitwarden.com/
- https://www.w3schools.com/html/
-