Term Project:

My Pass Project

Team members:
Layal Saad
Moria Almadrahi
Sara Kassem

Demo Video

Design Patterns Overview:

Our design uses six patterns:

- Singleton
- Builder
- Observer
- Proxy
- Mediator
- Chain of Responsibility

Singleton Pattern

- We implemented the Singleton pattern through our SessionManager class. Why Singleton?
- ● Only one session controller should exist
- ● Controls login/logout
- ● Prevents accidental double session handling This ensures stable session behavior across the entire project.

Builder Pattern

- Our PasswordBuilder and PasswordDirector handle generating secure passwords. Builder lets us:
- ● Choose password length
- ● Add/remove uppercase
- ● Add/remove symbols
- ● Plug in different builders later The frontend slider + checkboxes communicate with a PHP backend script that uses the Builder pattern.

Observer Pattern

- We used two observers:
- ● WeakPasswordObserver
- ● ExpirationObserver When an item is saved: $subject->notify(['type' => 'password', 'value' => $password]); These observers instantly print warnings. This keeps the dashboard logic clean because observers self-manage their behavior.”

CRUD + UI

- We implemented the core application features: CRUD (Create, Read, Update, Delete)
- ● Adding vault items
- ● Updating items
- ● Removing items UI & Forms
- ● Clean layout with forms and tables
- ● Password toggle, notes field
- ● Integration of all validation + pattern messages Security
- ● Hashed user passwords
- ● Input sanitization This forms the main functionality used by the user.

Proxy Pattern

- We implemented the Proxy Pattern to simulate restricted access. Why Proxy?
- It lets us control permissions before letting a user access sensitive vault features. Example: $proxy = new VaultAccessProxy("admin"); echo $proxy->getItem(5);
- If you change the role to “user”, the proxy denies access.
- This pattern allows controlled access without modifying the real vault logic.

Mediator Pattern

- The Mediator Pattern coordinates validation logic between TitleComponent and PasswordComponent. What it does:
- ● Centralizes form validation
- ● Components don’t talk to each other directly
- ● The mediator orchestrates everything Example: $mediatorErrors = $mediator->validate(["title" => $title, "password" => $password]); This improves maintainability by keeping the dashboard simple and keeping validation modular.”

Chain of Responsibility

- For advanced password validation, I implemented a Chain of Responsibility validator: Handlers in order:
- 1. LengthHandler
- 2. NumberHandler
- 3. UppercaseHandler
- 4. SymbolHandler Each handler checks ONE rule and passes the request onward: $length->setNext($number)->setNext($upper)->setNext($symbol); $message = $length->handle($password); If a rule fails, the chain stops. This mimics industry-standard password strength systems.

Group Contributions:

- Layal:
- ● Registration + login
- ● SessionManager (Singleton)
- ● Password Builder pattern
- ● Observer pattern
- ● Program demonstration

Group Contributions:

- Moria:
- ● Dashboard UI
- ● Vault CRUD functionality
- ● Security hashing &sanitization
- ● Frontend integration

Group Contributions:

- Sara:
- ● Proxy Pattern
- ● MediatorPattern
- ● Chain of Responsibility

GitHub URL:

- https://github.com/malmadra/MyPassProject.git