MyPass Term Project

CIS 476

Daniel Johnson

Developed In Godot 4.5

1. Introduction

MyPass is a secure, lightweight password/vault manager built in **Godot 4.5** using **multiple software design patterns**, including:

- **Singleton**
- **Observer**
- **Mediator**
- **Builder**
- **Proxy**
- **Chain of Responsibility**
- **Simple Factory**

The application supports secure storage of logins, credit cards, identity documents, and secure notes, encrypted on disk using a custom XOR scheme combined with salted SHA-256 hashing.

All UML images used here are included separately as well for viewing as full PNG.

2. System Architecture Overview

MyPass is structured around **autoload singletons**, a UI navigation mediator, a vault persistence layer, and a set of strongly typed vault item models.

The core implementation in MyPass includes:

- Core Services (Autoloads)
  - Session_Manager.gd — login, timeout, key derivation, CoR builder
  - Vault_Store.gd — in-memory vault, item CRUD, persistence
  - UI_Hub.gd — central mediator for UI navigation + toasts
  - Security_Watcher.gd — weak-password & expiration observer
  - Clipboard_Guard.gd — secure clipboard with auto-clear
- Models / Entities
  - VaultItem.gd — base class for all item types
  - LoginItem.gd
  - CreditCardItem.gd
  - IdentityItem.gd
  - SecureNoteItem.gd
  - Storage Files (Generated)
    - account.json — user credentials + recovery info + salt
    - vault.enc — encrypted vault database
- Cryptography & Utilities
  - CryptoUtil.gd — salt, hashing, XOR encryption, vault I/O
  - JsonUtil.gd — JSON encode/decode wrapper
- UI Screens / Scenes
  - Auth.gd — login screen
  - Register.gd — account creation + recovery setup
  - Recovery.gd — Chain of Responsibility question flow
  - VaultRoot.gd — vault item list + filters
  - VaultItemEditor.gd — create/edit items
  - SelectItemType.gd — choose item type to create
  - ToastLayer.gd — toast UI overlay
  - ItemListPanel.gd — renders vault items as buttons
  - Main Application
  - Main.gd — root scene controller, connects UI_Hub + services
- Patterns / Reusable Components
  - PasswordBuilder.gd — Builder pattern (password generation)
  - MaskedFieldProxy.gd — Proxy for masked input fields
  - RecoveryStep.gd — CoR step base class
  - QuestionStep.gd — CoR concrete handler

2.1 High-Level Architecture Diagram

*(See **Diagram 2.1** in the Diagrams folder. Included but too large for this document)*

Key Components

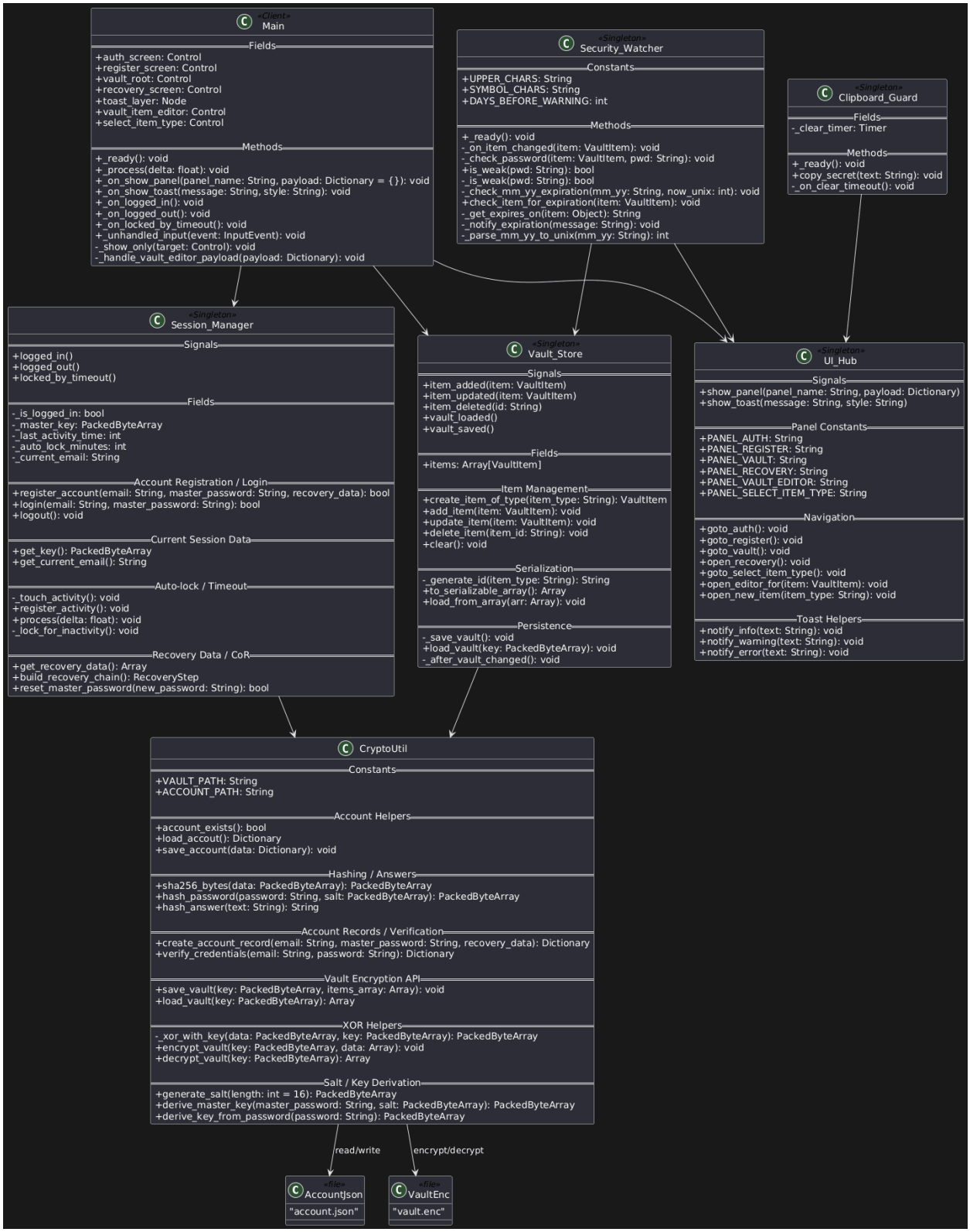
Component	Description
Session_Manager	Handles login/logout, session key derivation, and auto-lock timeout logic.
Vault_Store	Manages in-memory vault items and persistence to disk.
UI_Hub	Central mediator for screen-to-screen navigation and toast notifications.
Security_Watcher	Observes vault items for weak passwords or upcoming expirations.
Clipboard_Guard	Manages secure copy-to-clipboard with automatic clearing.
VaultItem + subclasses	Models representing vault data types (Login, CreditCard, Identity, SecureNote).
CryptoUtil	Hashing, encryption, and JSON I/O.
UI Scenes	Auth, Register, Recovery, VaultRoot, SelectItemType, VaultItemEditor, ToastLayer.

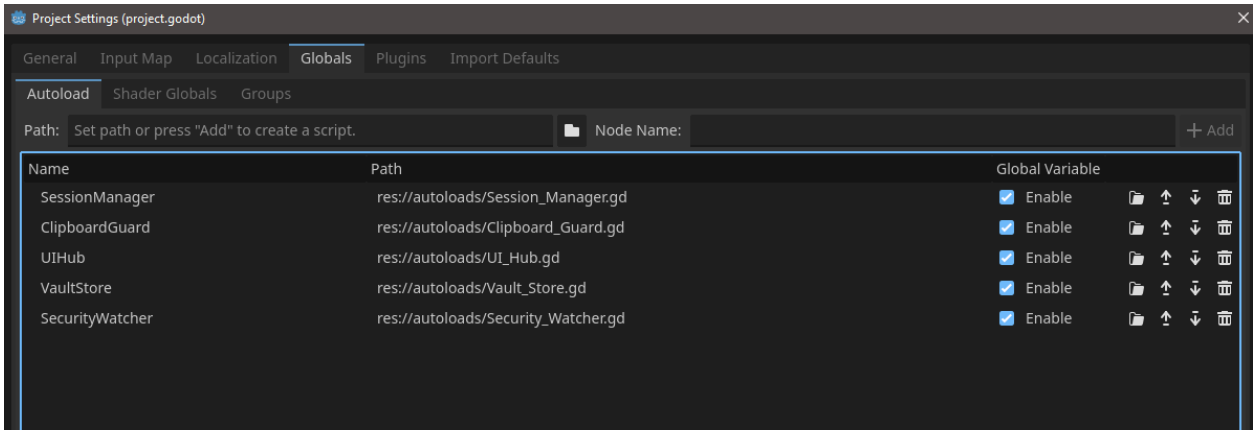
3. Design Patterns Implemented

In this section we will go over each of the required design patterns in the Term Project requirements. The implementation will be discussed as well as its adherence to the design pattern requirements.

3.1 Singleton Pattern (Global Autoloads)

Singleton implementation is handled in-engine in Godot 4.5 via the Autoloads functionality. Autoloading a script for a scene will default it to singleton. Singleton classes in MyPass are for Global use and operation.

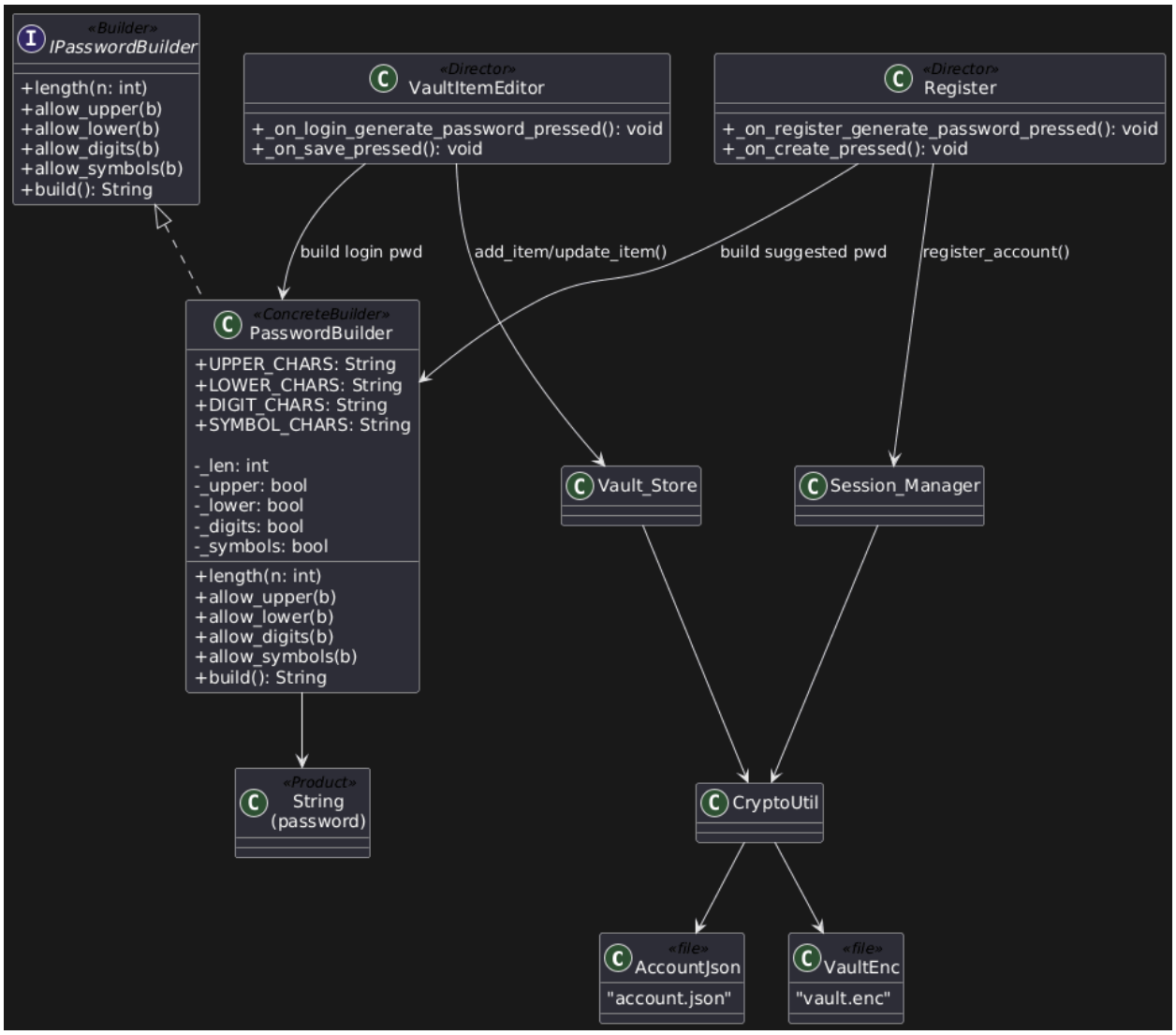

«Client»
Main
Fields
+auth_screen: Control
+register_screen: Control
+vault_root: Control
+recovery_screen: Control
+toast_layer: Node
+vault_item_editor: Control
+select_item_type: Control
Methods
+_ready(): void
+_process(delta: float): void
+_on_show_panel(panel_name: String, payload: Dictionary = {}): void
+_on_show_toast(message: String, style: String): void
+_on_logged_in(): void
+_on_logged_out(): void
+_on_locked_by_timeout(): void
+_unhandled_input(event: InputEvent): void
-_show_only(target: Control): void
-_handle_vault_editor_payload(payload: Dictionary): void
«Singleton»
Security_Watcher
Constants
+UPPER_CHARS: String
+SYMBOL_CHARS: String
+DAYS_BEFORE_WARNING: int
Methods
+_ready(): void
-_on_item_changed(item: VaultItem): void
-_check_password(item: VaultItem, pwd: String): void
+is_weak(pwd: String): bool
-_is_weak(pwd: String): bool
-_check_mm_yy_expiration(mm_yy: String, now_unix: int): void
+check_item_for_expiration(item: VaultItem): void
-_get_expires_on(item: Object): String
-_notify_expiration(message: String): void
-_parse_mm_yy_to_unix(mm_yy: String): int
«Singleton»
Clipboard_Guard
Fields
-_clear_timer: Timer
Methods
+_ready(): void
+copy_secret(text: String): void
-_on_clear_timeout(): void
«Singleton»
Session_Manager
Signals
+logged_in()
+logged_out()
+locked_by_timeout()
Fields
-_is_logged_in: bool
-_master_key: PackedByteArray
-_last_activity_time: int
-_auto_lock_minutes: int
-_current_email: String
Account Registration / Login
+register_account(email: String, master_password: String, recovery_data): bool
+login(email: String, master_password: String): bool
+logout(): void
Current Session Data
+get_key(): PackedByteArray
+get_current_email(): String
Auto-lock / Timeout
-_touch_activity(): void
+register_activity(): void
+process(delta: float): void
-_lock_for_inactivity(): void
Recovery Data / CoR
+get_recovery_data(): Array
+build_recovery_chain(): RecoveryStep
+reset_master_password(new_password: String): bool
«Singleton»
Vault_Store
Signals
+item_added(item: VaultItem)
+item_updated(item: VaultItem)
+item_deleted(id: String)
+vault_loaded()
+vault_saved()
Fields
+items: Array[VaultItem]
Item Management
+create_item_of_type(item_type: String): VaultItem
+add_item(item: VaultItem): void
+update_item(item: VaultItem): void
+delete_item(item_id: String): void
+clear(): void
Serialization
-_generate_id(item_type: String): String
+to_serializable_array(): Array
+load_from_array(arr: Array): void
Persistence
-_save_vault(): void
+load_vault(key: PackedByteArray): void
-_after_vault_changed(): void
«Singleton»
UI_Hub
Signals
+show_panel(panel_name: String, payload: Dictionary)
+show_toast(message: String, style: String)
Panel Constants
+PANEL_AUTH: String
+PANEL_REGISTER: String
+PANEL_VAULT: String
+PANEL_RECOVERY: String
+PANEL_VAULT_EDITOR: String
+PANEL_SELECT_ITEM_TYPE: String
Navigation
+goto_auth(): void
+goto_register(): void
+goto_vault(): void
+open_recovery(): void
+goto_select_item_type(): void
+open_editor_for(item: VaultItem): void
+open_new_item(item_type: String): void
Toast Helpers
+notify_info(text: String): void
+notify_warning(text: String): void
+notify_error(text: String): void
CryptoUtil
Constants
+VAULT_PATH: String
+ACCOUNT_PATH: String
Account Helpers
+account_exists(): bool
+load_accout(): Dictionary
+save_account(data: Dictionary): void
Hashing / Answers
+sha256_bytes(data: PackedByteArray): PackedByteArray
+hash_password(password: String, salt: PackedByteArray): PackedByteArray
+hash_answer(text: String): String
Account Records / Verification
+create_account_record(email: String, master_password: String, recovery_data): Dictionary
+verify_credentials(email: String, password: String): Dictionary
Vault Encryption API
+save_vault(key: PackedByteArray, items_array: Array): void
+load_vault(key: PackedByteArray): Array
XOR Helpers
-_xor_with_key(data: PackedByteArray, key: PackedByteArray): PackedByteArray
+encrypt_vault(key: PackedByteArray, data: Array): void
+decrypt_vault(key: PackedByteArray): Array
Salt / Key Derivation
+generate_salt(length: int = 16): PackedByteArray
+derive_master_key(master_password: String, salt: PackedByteArray): PackedByteArray
+derive_key_from_password(password: String): PackedByteArray
read/write
encrypt/decrypt
«file»
AccountJson
"account.json"
«file»
VaultEnc
"vault.enc"



Descriptions

- **Session_Manager**
  Stores login state, runs auto-lock timer, exposes recovery data.
- **Vault_Store**
  Stores all loaded vault items in memory, emits signals on update, saves to disk.
- **Security_Watcher**
  Subscribes to vault changes and emits warnings (weak passwords, expired documents).
- **UI_Hub**
  A UI mediator that centralizes screen transitions and notifications.
- **Clipboard_Guard**
  Ensures sensitive clipboard contents auto-clear after 60 seconds.

3.2 Builder Pattern (Password Builder)



Description

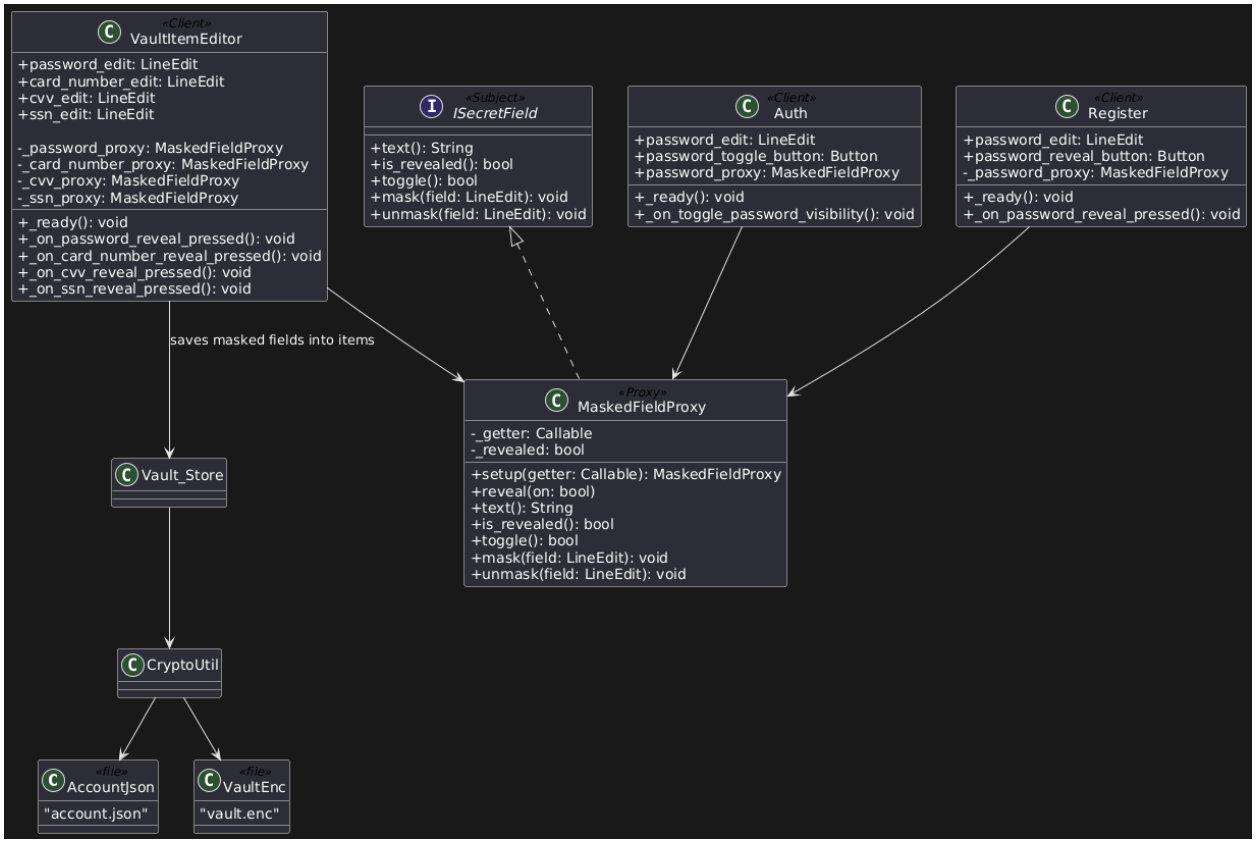
PasswordBuilder constructs strong, randomized passwords based on configurable rules, used by:

- Registration scene
- VaultItemEditor → Login items

Builder pattern characteristics:

Role	Class
Builder	PasswordBuilder
Product	Strong password string
Director	Register, VaultItemEditor

3.3 Proxy Pattern (Masked Fields)



Description

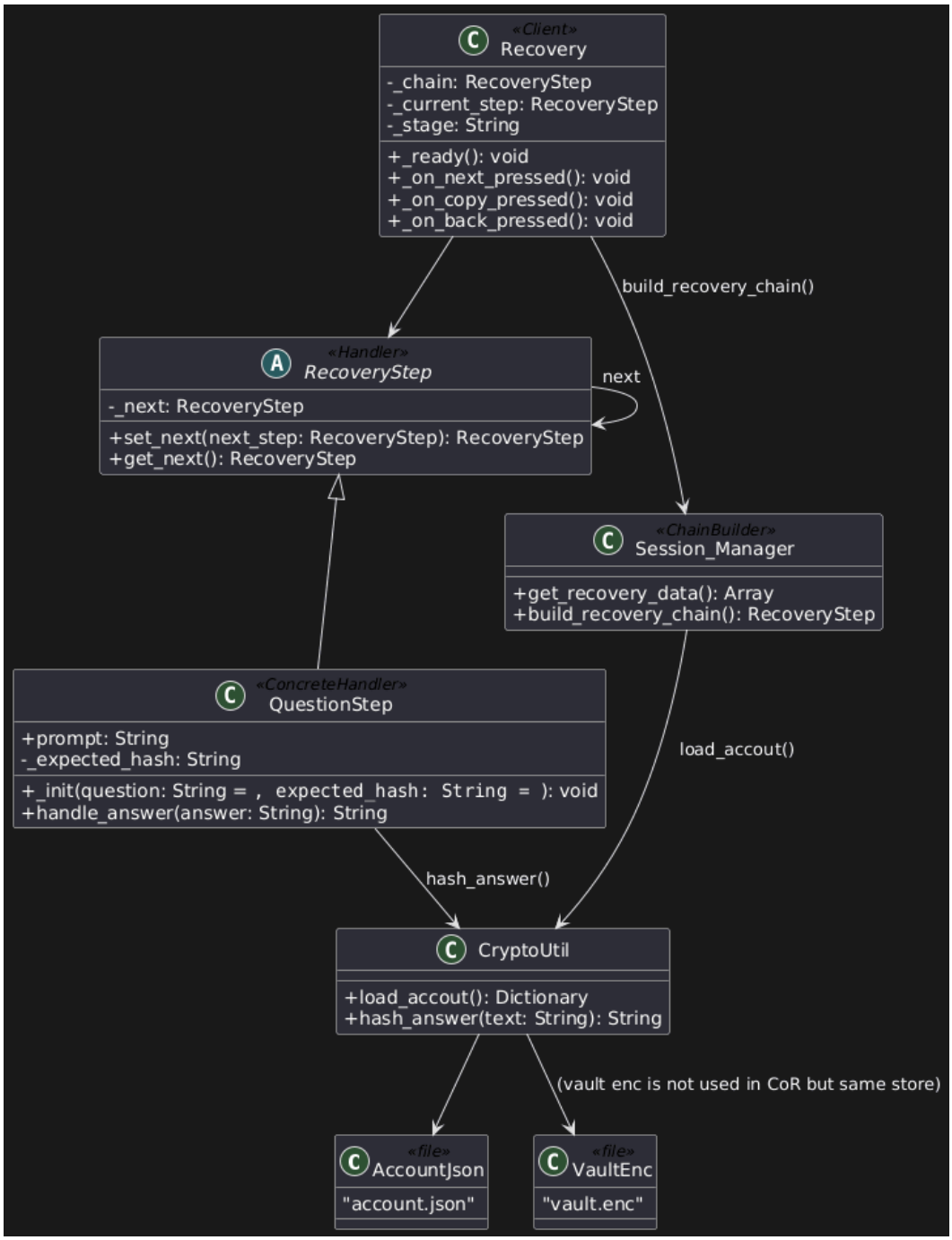
MaskedFieldProxy adds an access-control layer on top of Godot LineEdit fields, returning either:

- The raw string (unmasked)
- A placeholder mask string (e.g., •••••••)

This safely reuses UI code for many fields:

- Login password
- Card number
- CVV
- SSN
- Identity number

3.4 Chain of Responsibility Pattern (Password Recovery)



Description

Password recovery is explicitly required in the initial requirements. I would recommend moving toward a “password reset” workflow, as “password recovery” is almost never used in enterprise applications. To recover a password it must be stored insecurely (raw) in the database. Here we have implemented a raw recovery workflow with recommendations to move to a reset workflow in future updates/design.

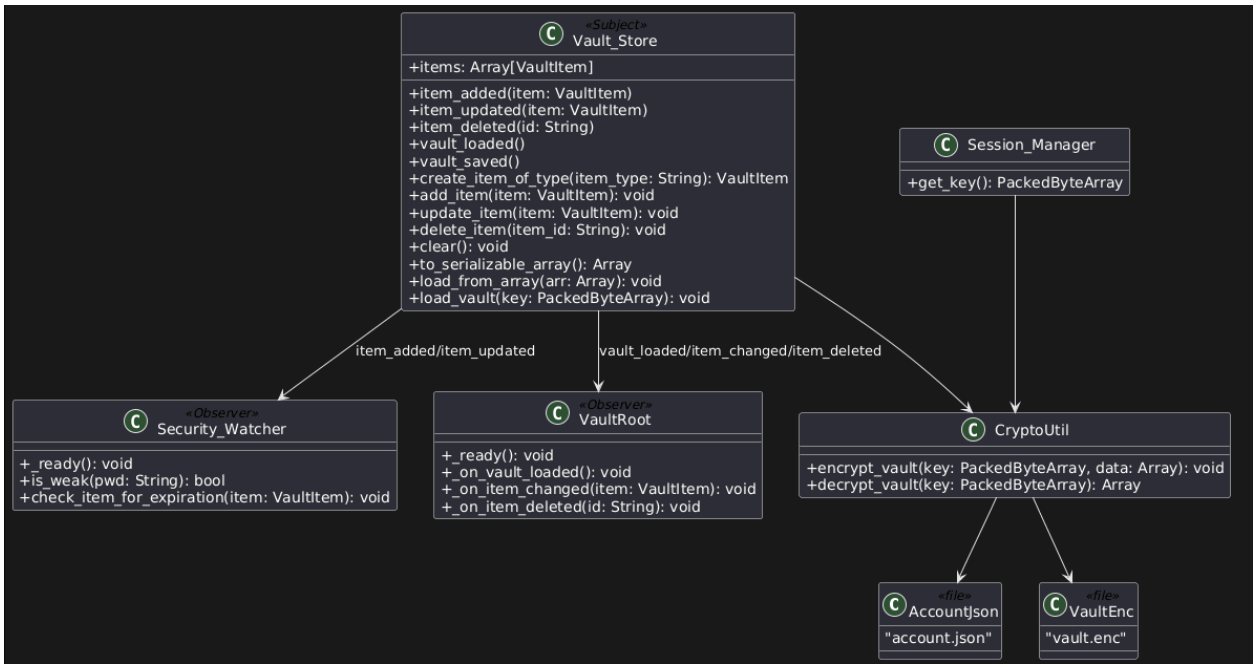
The recovery system is implemented using a Chain of Responsibility:

Component	**Role**
RecoveryStep	Abstract handler
QuestionStep	Concrete handler (validates one question)
Session_Manager.build_recovery_chain ()	Builds the CoR dynamically
Recovery scene	The client

Each step validates one answer and either:

- Stops the chain (fail)
- Proceeds (ok_next)
- Ends chain successfully (ok_done)

3.5 Observer Pattern (Item Updates/ Security)



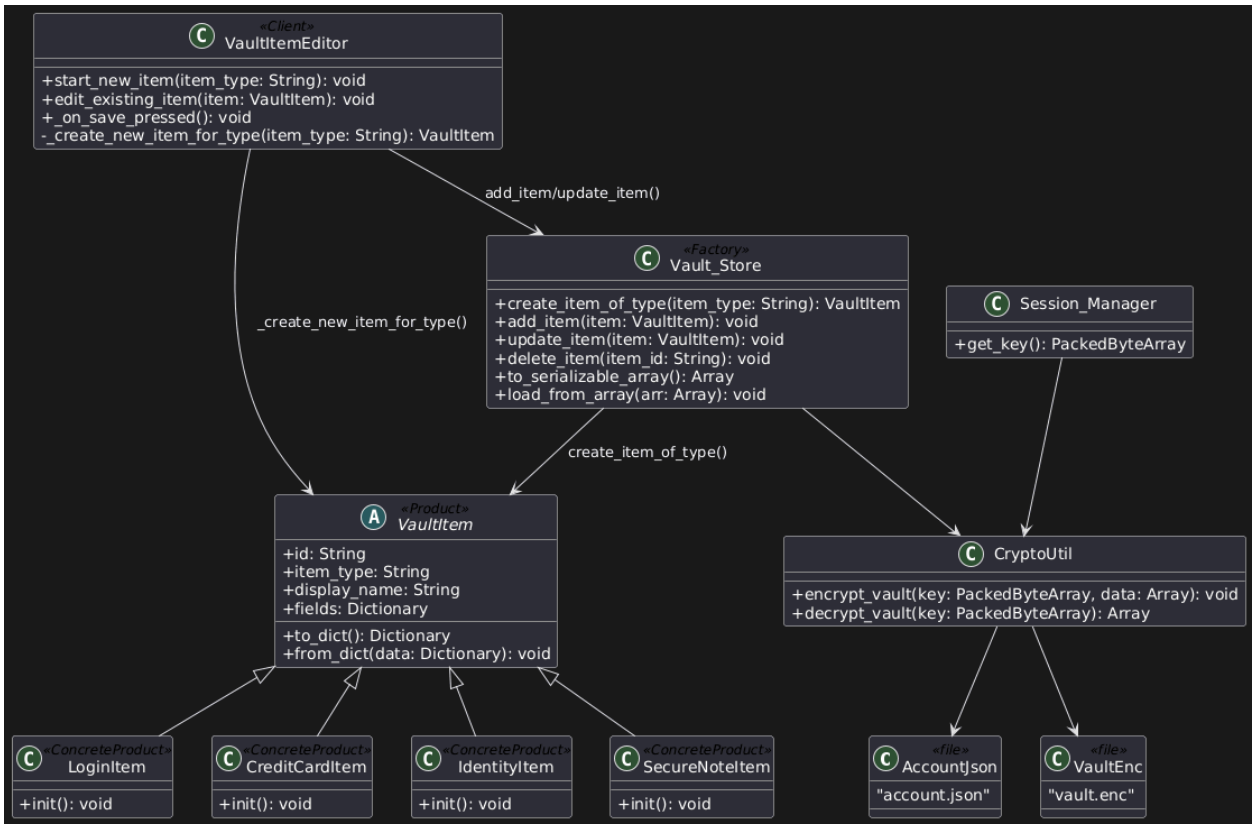
Description

Observer pattern is used to watch for Vault Items updated and connect signals to the UI and ToastLayer systems. The app uses the Godot signals to implement Observer:

Subject	**Observers**
Vault_Store	VaultRoot, Security_Watcher
Security_Watcher	UI_Hub (to show warnings)

VaultRoot updates UI lists; Security_Watcher checks password strength & expirations.

3.6 Simple Factory Pattern (Vault Items)



Description

Factory pattern is used for creating vault items:

Role	**Class**
Factory	Vault_Store.create_item_of_type()
Products	LoginItem, CreditCardItem, IdentityItem, SecureNoteItem

3.7 Mediator (UIHub)

Description

UI_Hub acts as a central communication hub between all UI screens, allowing them to request navigation or toasts without directly referencing each other.**Main** listens to UI_Hub signals and determines which screen is visible.Together they implement the Mediator pattern to keep UI components decoupled and coordinated.

Used by:

- Auth
- Register
- VaultRoot
- VaultItemEditor
- Recovery
- SelectItemType
- ToastLayer

Role	Class
Mediator	UI_Hub
Concrete Mediator / Controller	Main
Colleague Screens	Auth, Register, VaultRoot, VaultItemEditor, Recovery, SelectItemType
Colleague Overlay	ToastLayer
Supporting Services Used by Colleagues	Session_Manager, Vault_Store, Security_Watcher, Clipboard_Guard

4. Data Storage & Database Schema

MyPass uses two “database files”:

4.1 account.json schema

```
{
  "email": "user@example.com",
  "salt": "hexstring",
  "password_hash": "hexstring",
  "recovery": [
        {
        "question": "What is…?",
        "answer_hash": "hexstring"
        }
  ],
  "master_password_plain": "plaintextPassword"
}
```

Field Descriptions

Field	**Description**
email	Primary key for account (only one account supported).

salt	Random 16-byte salt encoded as hex.
password_hash	SHA-256(salt + UTF8(password)).
recovery	Array of recovery questions and hashed answers.
master_password_plain	Required for project spec (not secure IRL).

4.2 vault.enc schema

Encrypted using XOR(key, data).

When decrypted, contents is:


```
[
  {
        "id": "login_1733020000",
        "item_type": "login",
        "display_name": "My Steam",
```

```
        "fields": {
        "username": "myname",
        "password": "hunter2",
        "url": "https://store.steampowered.com"
        }
  }
]
```


Each entry corresponds to a VaultItem subclass.

Field Description

Field	Description
id	Unique ID (type_timestamp).
item_type	Determines concrete class on load.
display_name	Human-readable name.
fields	Key/value dictionary; structure depends on item type.

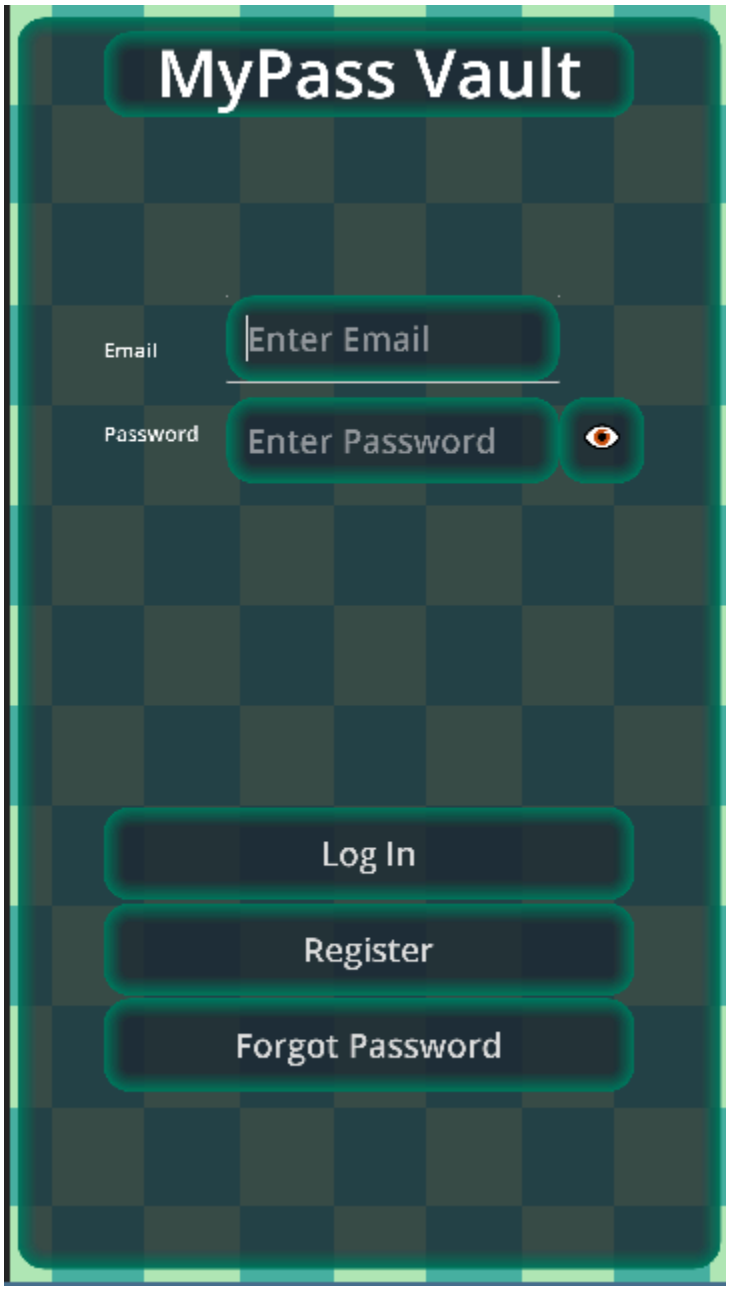
5. Application Flow

Below are descriptions; you can insert screenshots in the placeholders.

5.1 Login Screen

Features

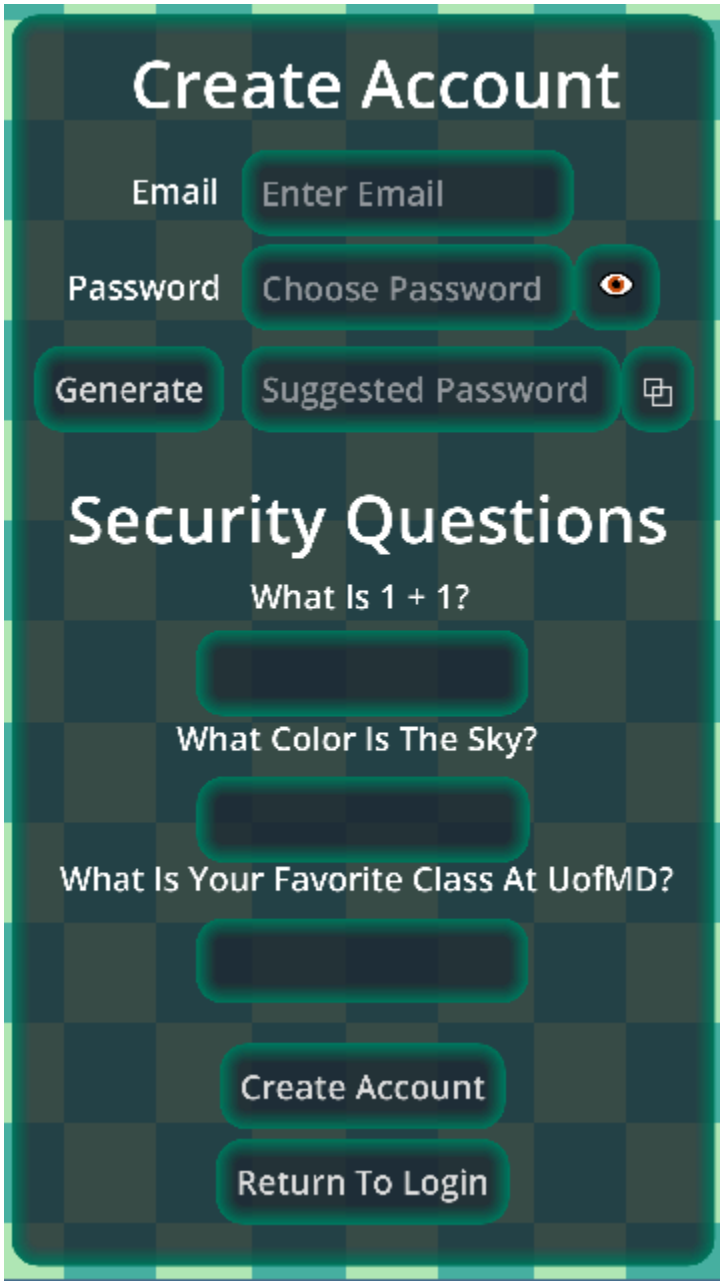
- Email + password entry
- Mask toggle
- Links to: Register / Recovery
- Clipboard guard on copy



5.2 Register Screen

Features

- Email + password
- 3 fixed recovery questions
- Password generator (Builder pattern)
- Mask toggle and copy



5.3 Vault Root Screen

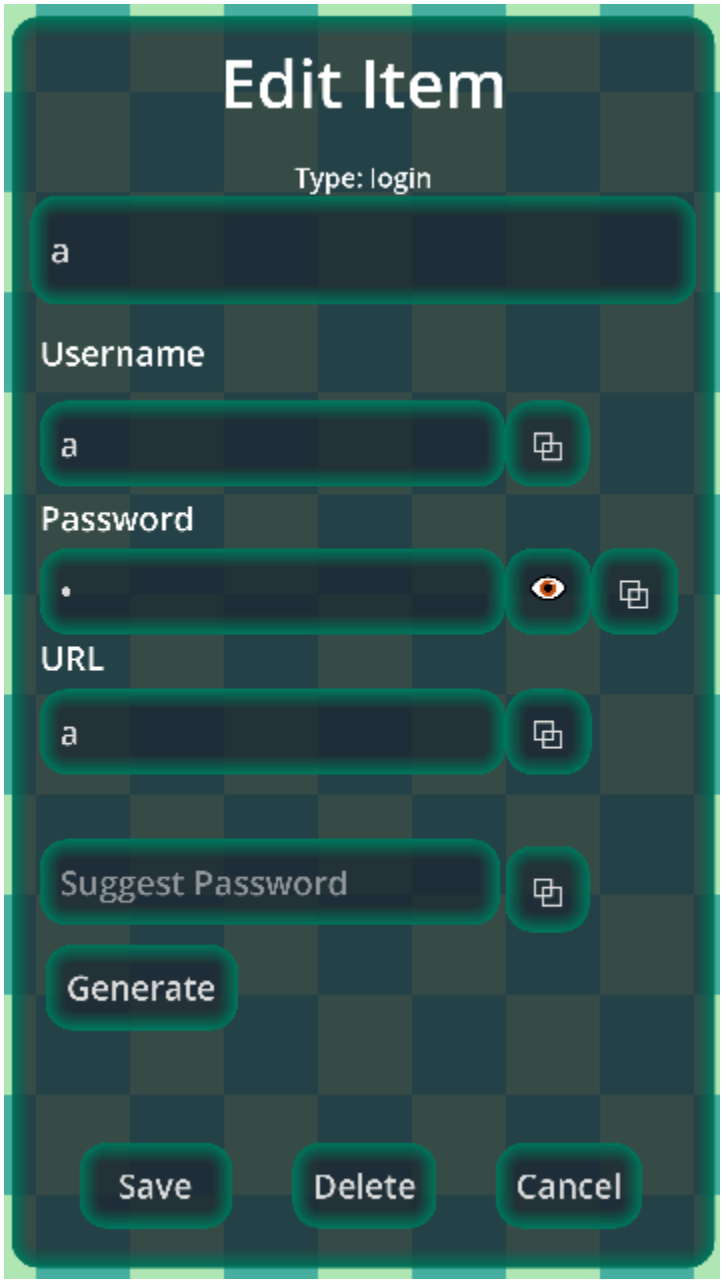
Features

- Type filter (All / Logins / Cards / IDs / Notes)
- List of items via ItemListPanel
- Add Item button → SelectItemType
- Logout button

5.4 Vault Item Editor Screen

Features

- Different field groups depending on selected type
- Mask/unmask using Proxy pattern
- Copy buttons guarded by Clipboard_Guard
- Cancel/ Save / Delete



5.5 Recovery Screen

Features

- Dynamic question chain
- Auto reveal final recovered password
- Copy option via Clipboard_Guard