MyPass – Secure Vault Manager

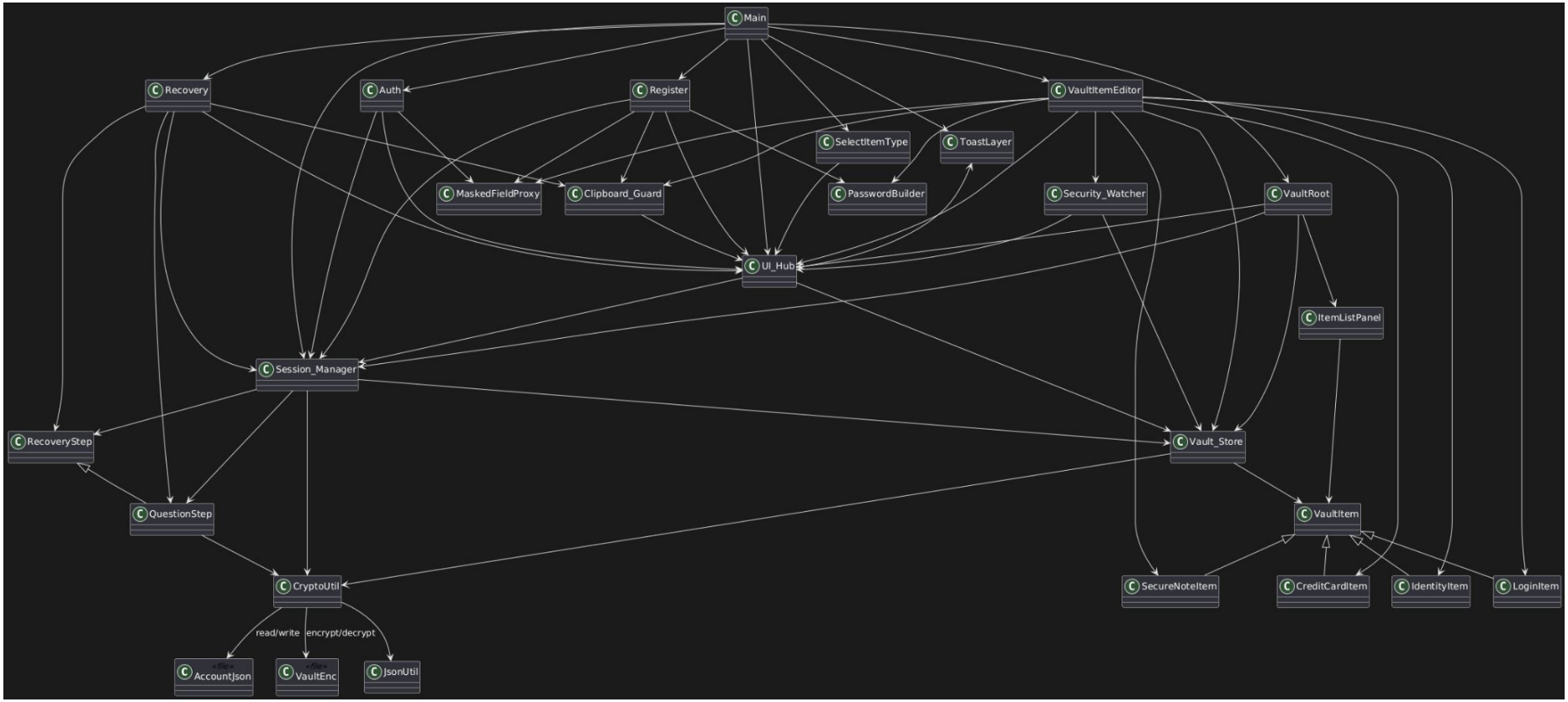
CIS 476 Term Project

Daniel Johnson

Godot 4.5

- A secure vault/password manager
- Encrypted storage
- Android/Desktop compatible
- Developed in Godot 4.5

High Level


Main
Recovery
Auth
Register
VaultItemEditor
SelectItemType
ToastLayer
MaskedFieldProxy
Clipboard_Guard
PasswordBuilder
Security_Watcher
VaultRoot
UI_Hub
ItemListPanel
Session_Manager
RecoveryStep
Vault_Store
QuestionStep
VaultItem
CryptoUtil
SecureNoteItem
CreditCardItem
IdentityItem
LoginItem
read/write
encrypt/decrypt
AccountJson
VaultEnc
JsonUtil

Project Architecture

- 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

Project Architecture

- 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

Project Architecture

- 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

Implemented Design Patterns

- Singleton - (Global Autoloads)
- Mediator - (UI Hub)
- Observer - (Item Updates/Security)
- Builder - (Password Builder)
- Proxy - (Masked Fields)
- Chain of Responsibility - (Password Recovery)
- Simple Factory - (Vault Items)

Database / Storage Model

- account.json: Decrypted

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

Database / Storage Model

- vault.enc: Schema (XOR encrypted)


```
[
 {
        "id": "login_1733020000",
        "item_type": "login",
        "display_name": "My Steam",
        "fields": {
        "username": "myname",
        "password": "hunter2",
        "url": "https://store.steampowered.com"
        }
 }
]
```

VaultItem Models

```
## Responsible for:
##  * Base data model for all vault items
##  * Serializing/deserializing fields to/from dictionaries

extends Resource
class_name VaultItem

var id: String = ""
var item_type: String = ""
var display_name: String = ""
var fields: Dictionary = {}

# Convert this item into a serializable Dictionary
func to_dict() -> Dictionary:
	return{
		"id": id,
		"item_type": item_type,
		"display_name": display_name,
		"fields": fields,
	}

# Populate this item from a Dictionary
func from_dict(data: Dictionary) -> void:
	id = data.get("id", id)
	item_type = data.get("item_type", item_type)
	display_name = data.get("display_name", display_name)
	fields = data.get("fields", fields)
```

VaultItem Models

```
extends VaultItem
class_name LoginItem

# Call base init and set values
func init() -> void:
	item_type = "login"
	display_name = "New Login"
	fields = {
		"username": "",
		"password": "",
		"url": "",
	}
```

```
extends VaultItem
class_name IdentityItem

# Call base init and set values
func init() -> void:
	item_type = "identity"
	display_name = "New Identity"
	fields = {
		"document_type": "",
		"id_number": "",
		"issuing_country": "",
		"expires_on": "",
		"social_security_number": "",
	}
```

```
extends VaultItem
class_name CreditCardItem

# Call base init and set values
func init() -> void:
	item_type = "credit_card"
	display_name = "New Card"
	fields = {
		"card_holder": "",
		"card_number": "",
		"expires_on": "",
		"cvv": "",
	}
```

```
extends VaultItem
class_name SecureNoteItem

# Call base init and set values
func init() -> void:
	item_type = "secure_note"
	display_name = "New Secure Note"
	fields = {
		"title": "",
		"body": "",
	}
```

XOR Encryption (CryptoUtil)

- XOR encryption here works by XOR-ing each byte of the plaintext with a repeating key byte; applying the same operation again with the same key reverses it and recovers the original data.
- Example using snippets from CryptoUtil

```
func _xor_with_key(data: PackedByteArray, key: PackedByteArray) -> PackedByteArray:
	var out := PackedByteArray()
	if key.is_empty():
		return data.duplicate()
	for i in data.size():
		out.append(data[i] ^ key[i % key.size()])
	return out


var key := "my-secret-key".to_utf8_buffer()
var plain_bytes := "Hello Vault".to_utf8_buffer()

var enc := _xor_with_key(plain_bytes, key)   # encrypt
var dec := _xor_with_key(enc, key)           # decrypt (same function)

print(plain_bytes.get_string_from_utf8())    # "Hello Vault"
print(enc)                                   # scrambled bytes
print(dec.get_string_from_utf8())            # "Hello Vault" again
```

XOR Encryption (CryptoUtil)

How XOR + Salt Work in MyPass

Salt (Random Bytes)

- Generated during account creation
- Stored in `account.json`
- Ensures every user has a unique encryption key

Key Derivation

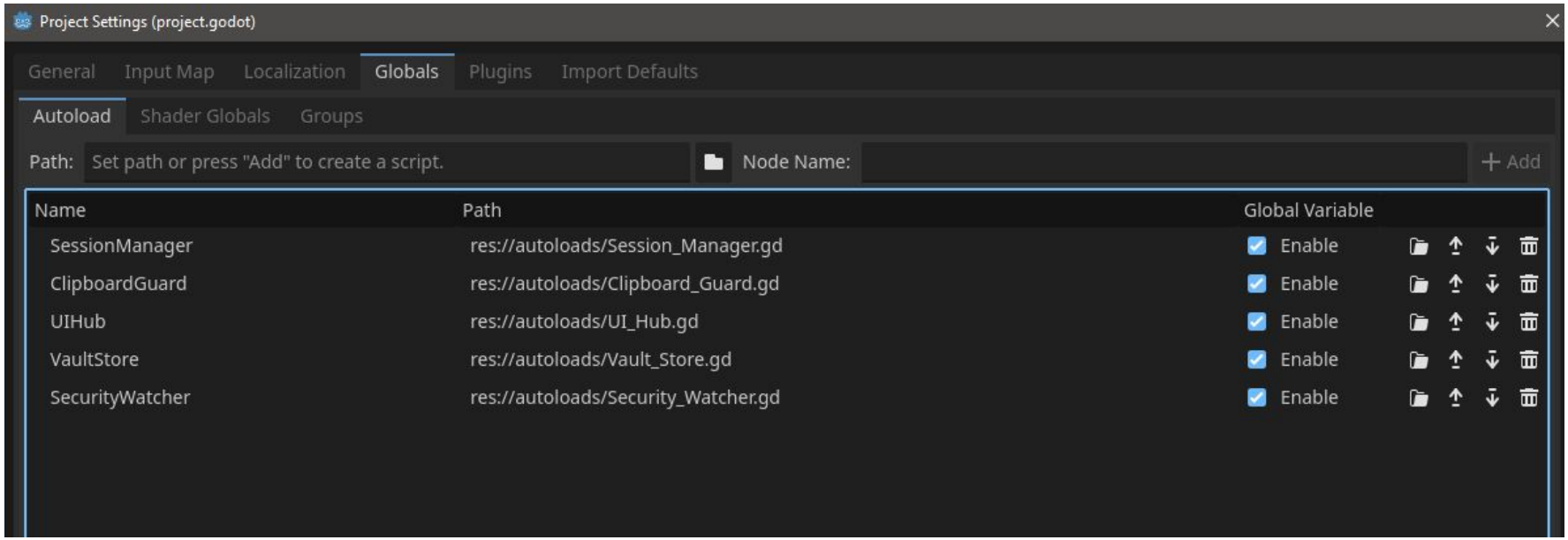
- Salt + master password
- Hashed with SHA-256
- Produces a stable 32-byte encryption key

XOR Encryption

- Vault data (JSON) converted to bytes
- XOR-ed with derived key
- Same function decrypts the vault
  (XOR is symmetric: $A \oplus B \oplus B = A$)

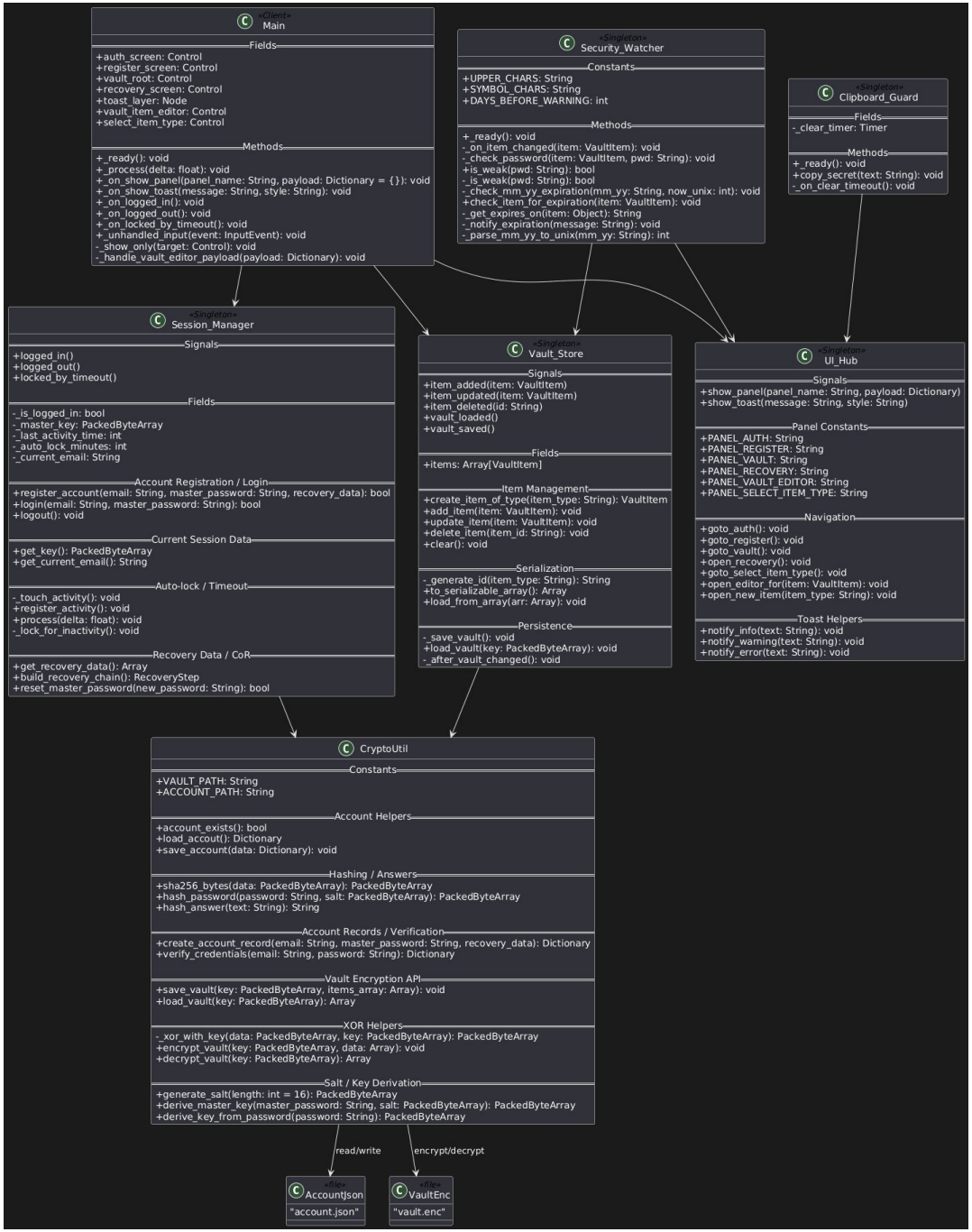
Singleton Autoloads

- Singleton implemented via Godot Autoload. Godot treats Autoloaded scripts as singleton. These singletons are also referred to as “Globals”



Singleton Autoloads

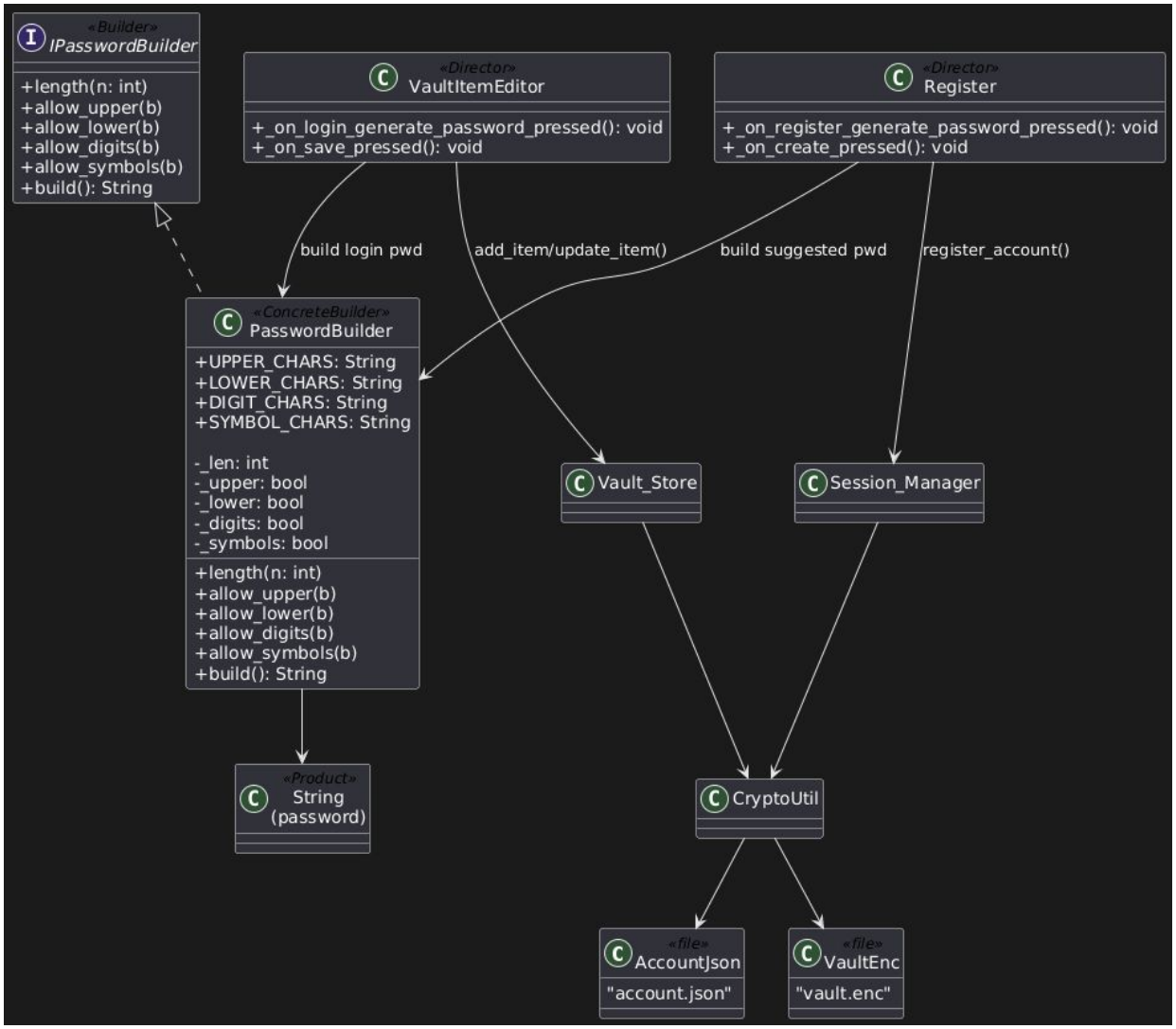
- Session_Manager
  - Logging in/out, derive encryption from master password, session timeout, expose data for account recovery
- Vault_Store
  - Manage in-memory Vault Items, save/load vault via CryptoUtil, signals on Item change
- UI_Hub
  - Central UI navigation, emit signals for panel changes and Toast
- Security_Watcher
  - Observe Vault_Store for changes, check for expiring passwords and weak passwords
- Clipboard_Guard
  - Copy sensitive text to clipboard, auto clear clipboard after timeout, send signal for Toast on clipboard clear


«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"

Builder Pattern (Password Builder)

- IPasswordBuilder (Interface)
  - Defines the contract for building passwords:
  - length(n), allow_upper(), allow_lower(), allow_digits(), allow_symbols(), and build().
  - Allows interchangeable password builders.
- PasswordBuilder (Concrete Builder)
  - Implements IPasswordBuilder to assemble a strong password.
  - Tracks configuration flags (upper/lower/digits/symbols) and length.
  - Constructs the final password using randomized character pools.
  - Guarantees minimum strength rules (≥10 chars, required symbol count).
- Register.gd (Director / Client)
  - Uses PasswordBuilder to generate suggested master passwords.
  - Configures builder with default settings, displays output to user.
  - Allows copying the generated password through Clipboard_Guard.
- VaultItemEditor.gd (Director / Client)
  - Uses PasswordBuilder to generate suggested passwords for Login items.
  - Lets the user copy the generated password and insert it into item fields.
- Output (Product)
  - Final password generated by builder.build():
  - Strong, randomized, meets strength checks, ready to use or store.

Builder Pattern (Password Builder)



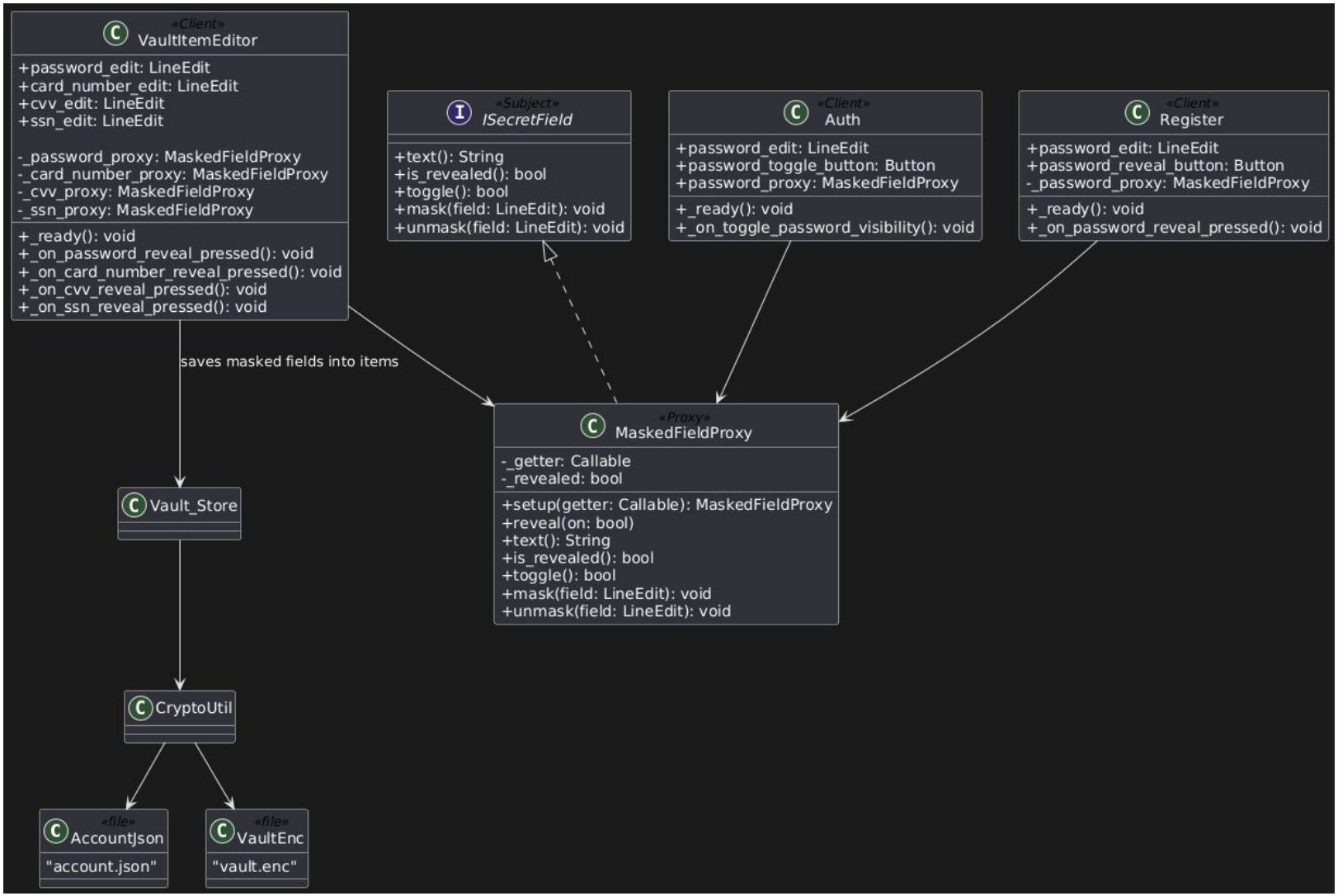
Proxy Pattern (Masked Fields)

- ISecretField (Interface)
  - Defines contract for secure field behavior:
  - text(), toggle(), is_revealed(), mask(), unmask().
  - Ensures any secret-handling logic is interchangeable and testable.
- MaskedFieldProxy (Proxy)
  - Implements ISecretField to sit between sensitive LineEdit fields and the UI.
  - Controls visibility of secret text (masked/unmasked).
  - Stores internal _revealed state and _getter callable to pull raw text safely.
  - Prevents direct exposure of sensitive data except when explicitly revealed.
- Auth.gd (Client)
  - Uses MaskedFieldProxy to toggle master password visibility on login screen.
  - Prevents accidental exposure of typed credentials.
  - Updates toggle button UI automatically.
- Register.gd (Client)
  - Uses MaskedFieldProxy to mask the master password during account creation.
  - Allows secure reveal/hide interactions for user verification.
  - Works with Clipboard_Guard for safe copying.

Proxy Pattern (Masked Fields)

- VaultItemEditor.gd (Client)
  - Uses multiple proxies: _password_proxy, _card_number_proxy, _cvv_proxy, _ssn_proxy.
  - Controls reveal/hide for each sensitive field depending on item type.
  - Ensures secret values never display or copy unless user explicitly chooses.
- Output (Protected Data Flow)
  - Sensitive fields remain masked by default.
  - Revealing values is explicitly controlled.
  - Accidental exposure is prevented through the proxy layer.

Proxy Pattern (Masked Fields)



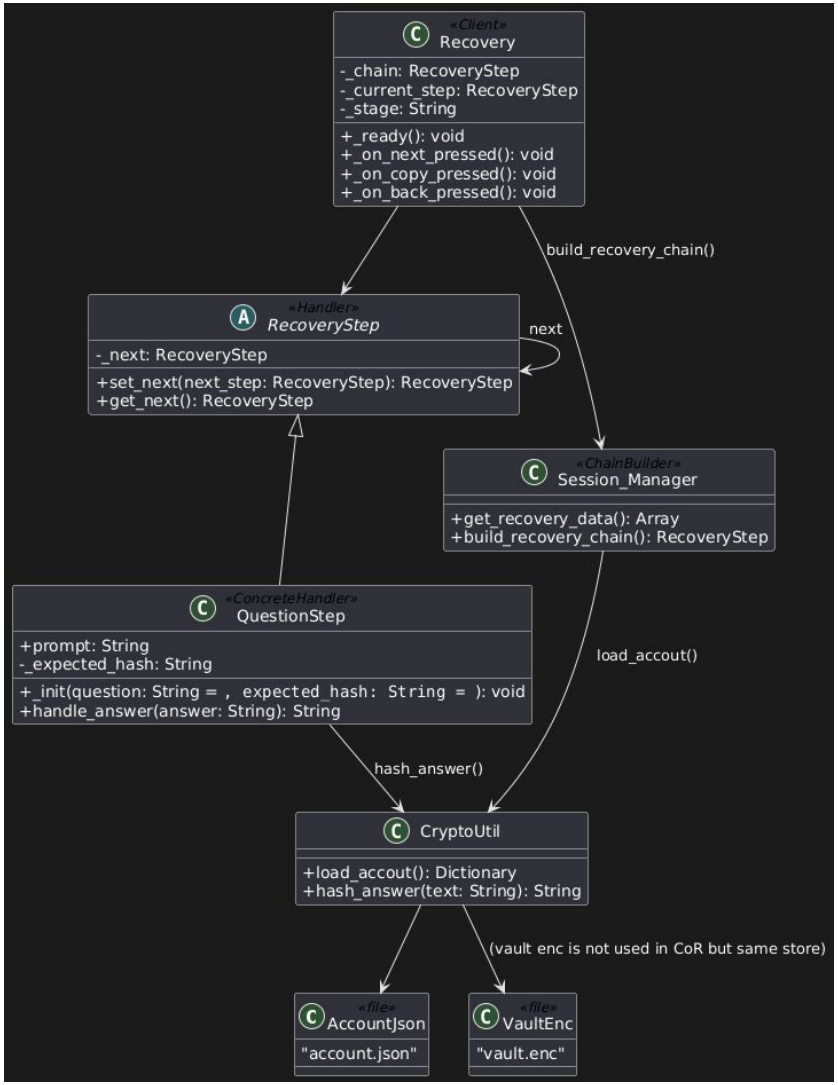
Chain Of Responsibility (Password Recovery)

- RecoveryStep (Handler)
  - Base class for recovery chain nodes.
  - Holds reference to _next step.
  - Provides set_next() and get_next() to link handlers into a chain.
- QuestionStep (Concrete Handler)
  - Stores prompt and _expected_hash.
  - Implements handle_answer() to compare hashed answer vs stored hash.
  - Returns "fail", "ok_next", or "ok_done" to control chain flow.
- Session_Manager (Chain Builder)
  - get_recovery_data() loads stored questions + answer hashes via CryptoUtil.
  - build_recovery_chain() creates QuestionStep objects and links them via set_next().
  - Returns the head RecoveryStep for the UI to drive.

Chain Of Responsibility (Password Recovery)

- Recovery.gd (Client)
  - Holds _chain and _current_step to walk the handlers in order.
  - On “Next”, calls handle_answer() on the current step and reacts to its return code.
  - If all questions pass, loads master_password_plain from account.json and reveals it.
  - Uses Clipboard_Guard and UI_Hub for secure copy + user feedback.
- Result (CoR Flow)
  - Each question is a handler in the chain.
  - Failure at any step stops the process with feedback.
  - Success at every step unlocks the recovery action (password reveal).

Chain Of Responsibility (Password Recovery)



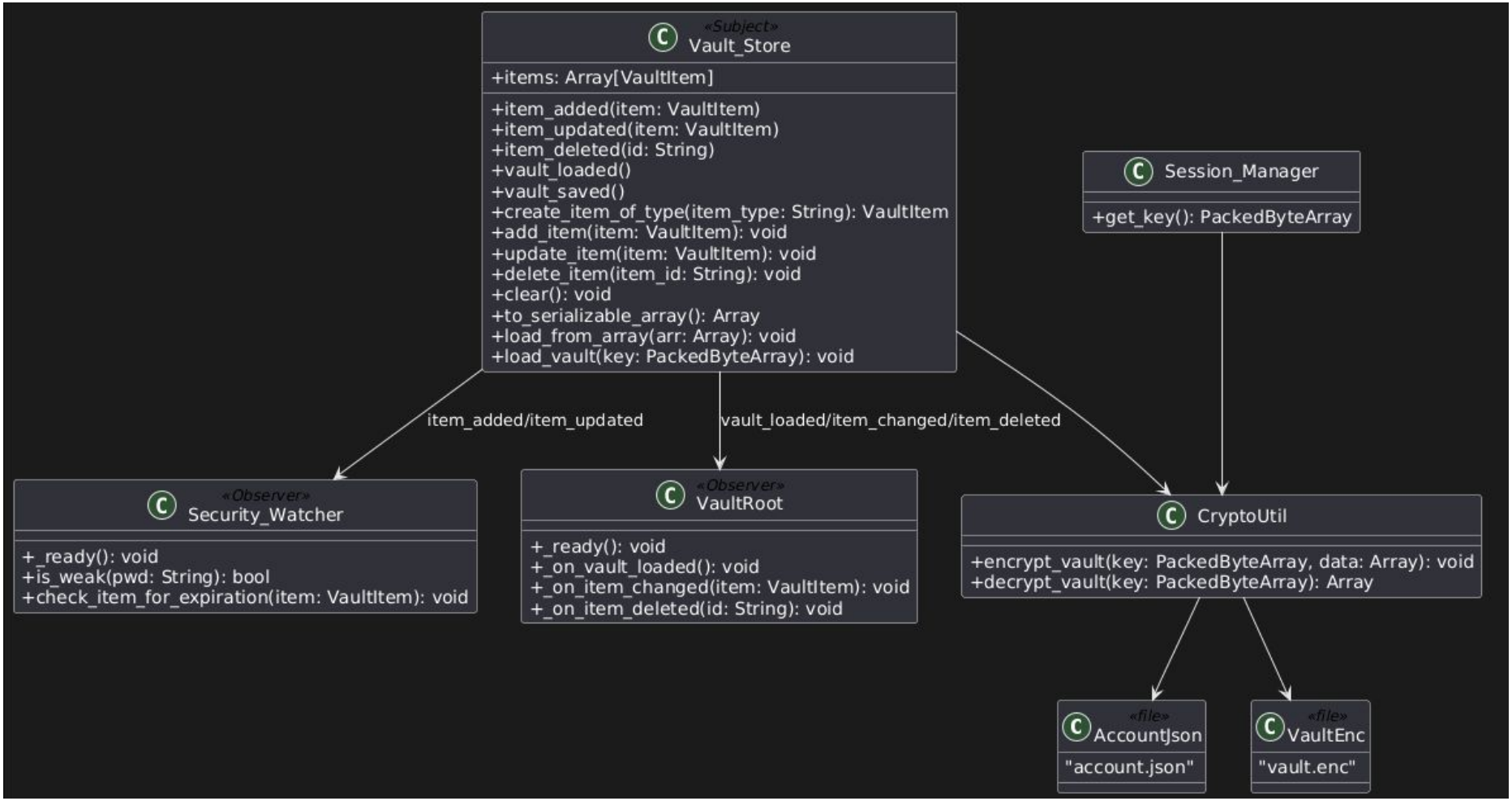
Observer (Item Updates / Security)

- Vault_Store (Subject / Observable)
  - Manage in-memory vault items and persistence.
  - Signals when data changes:
  - item_added, item_updated, item_deleted, vault_loaded, vault_saved.
  - Notifies all observers whenever the vault contents change.
- Security_Watcher (Observer – Security Rules)
  - Subscribes to Vault_Store.item_added and item_updated.
  - Checks new/updated items for weak passwords using is_weak().
  - Checks items with expires_on for soon-to-expire or expired data.
  - Uses UI_Hub to display security warnings as toasts.
- VaultRoot (Observer – UI Refresh)
  - Subscribes to vault_loaded, item_added, item_updated, and item_deleted.
  - Rebuilds the filtered item list when anything changes in the vault.
  - Updates status label (item counts, filter state).
  - Keeps the main vault UI always in sync with the underlying data.

Observer (Item Updates / Security)

- ItemListPanel (Passive Observer Target)
  - Receives filtered item arrays from VaultRoot.
  - Renders each VaultItem as a clickable button.
  - Emits item_selected to let other UI components react.
- Result (Decoupled Updates)
  - Vault_Store doesn’t know who is watching it.
  - Observers react automatically when vault state changes.
  - Security checks and UI refresh logic stay cleanly separated.

Observer (Item Updates / Security)



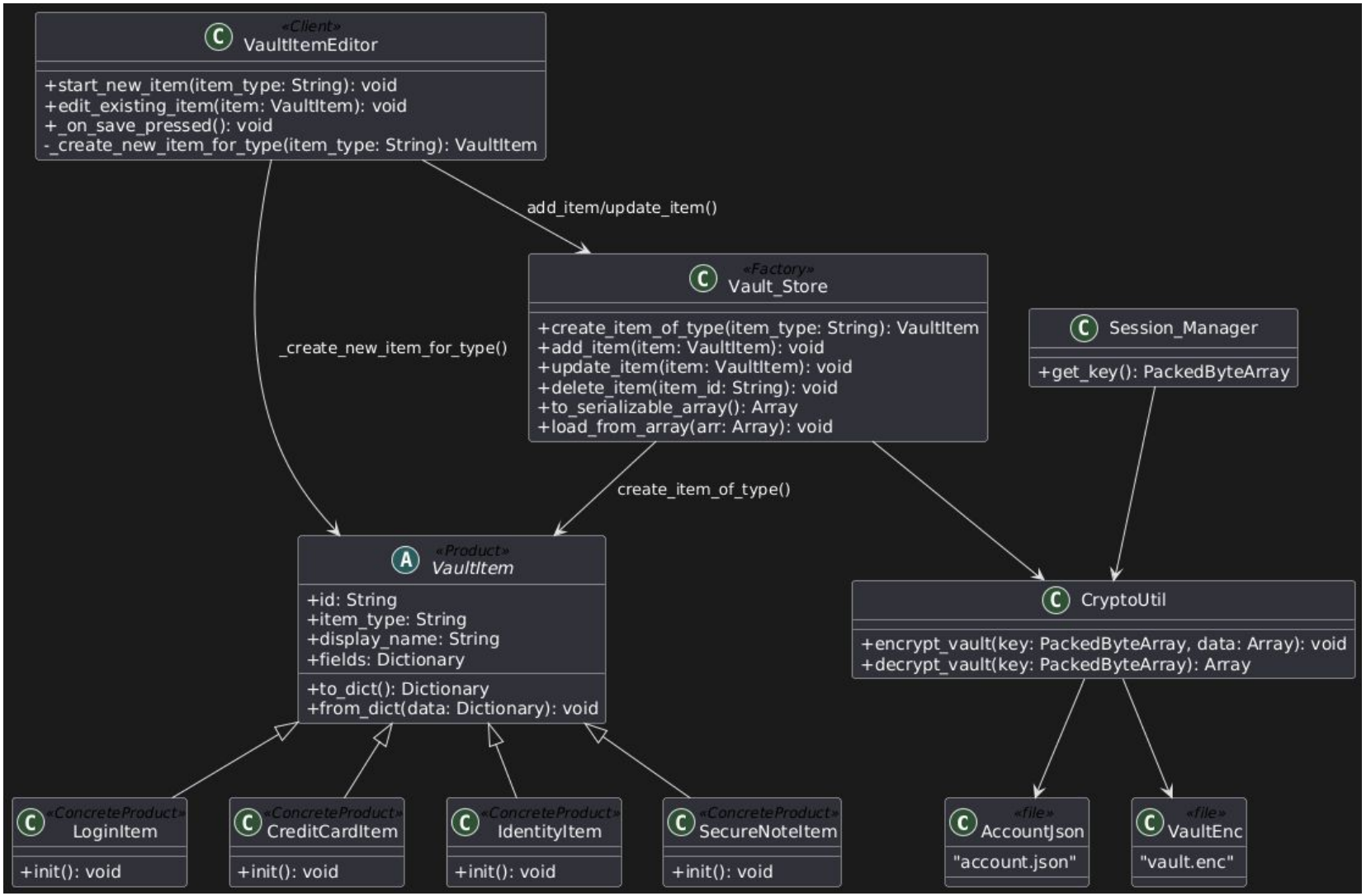
Factory (Vault Items)

- VaultItem (Base Product)
  - Abstract base class for all vault entries.
  - Defines shared fields: id, item_type, display_name, and fields{}.
  - Provides serialization helpers: to_dict() and from_dict().
- LoginItem, CreditCardItem, IdentityItem, SecureNoteItem (Concrete Products)
  - Each subclass sets its own item_type and default fields.
  - Represents a specific type of vault entry.
  - Used by UI screens and Vault_Store for type-specific editing & display.
- Vault_Store.create_item_of_type() (Simple Factory Method)
  - Takes an item_type: String and returns the appropriate subclass instance.
  - Implements the selection logic:
  - "login" → LoginItem.new()
  - "credit_card" → CreditCardItem.new(), etc.
  - Used for both new item creation and loading from decrypted vault data.

Factory (Vault Items)

- VaultItemEditor (Client)
  - Calls the Factory to create new items when the user selects a type.
  - Applies UI-entered field data to the created instance.
  - Saves the resulting item through Vault_Store.
- SelectItemType (Client / Front-end Trigger)
  - Lets the user choose which product type to create.
  - Passes item_type forward to the editor, which uses the factory.
- Result (Decoupled Object Creation)
  - UI never manually constructs subclasses.
  - All item creation is routed through a single factory function.
  - Adding a new item type only requires adding a new subclass and updating the factory method.

Factory (Vault Items)



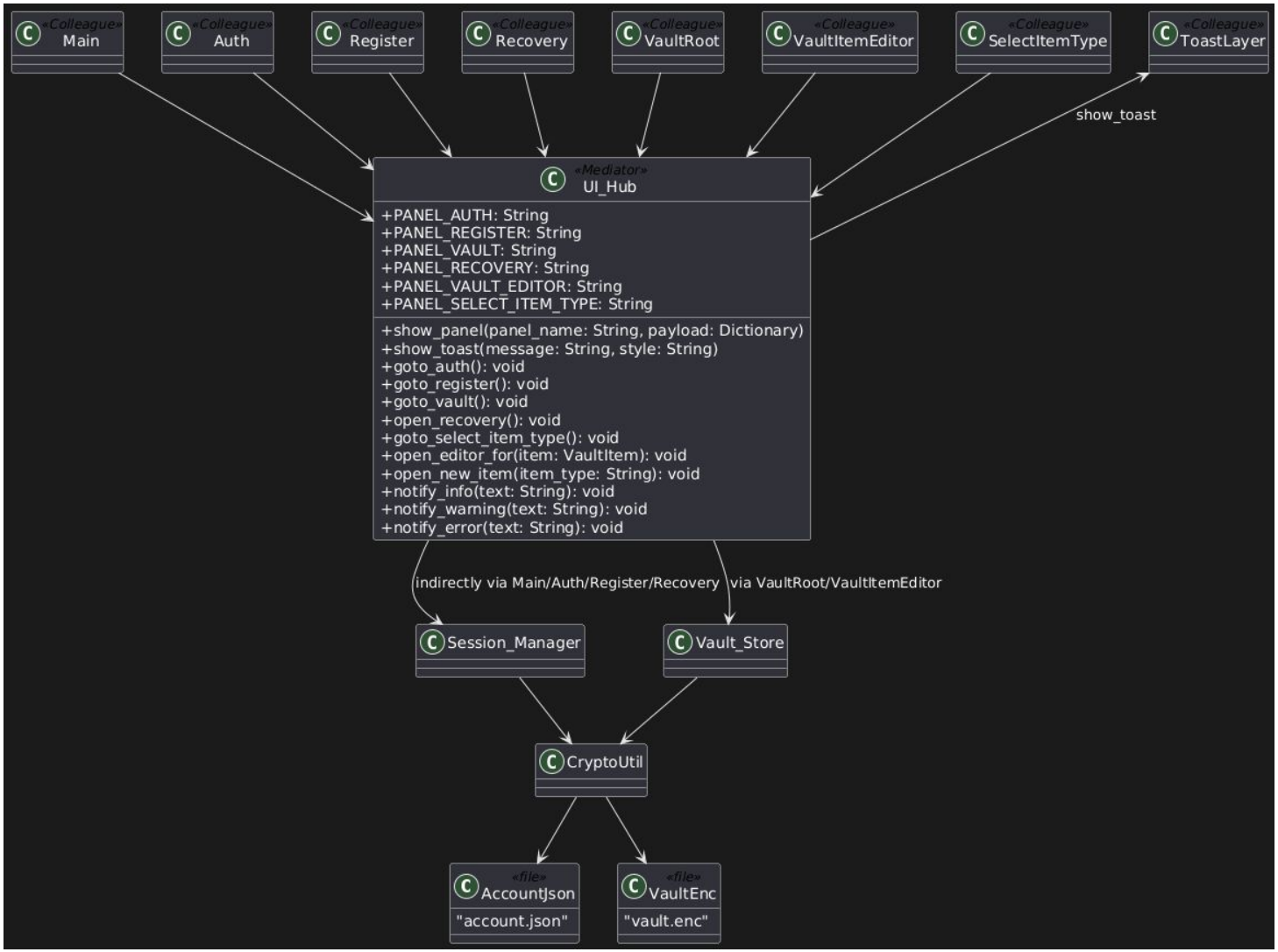
Mediator (UI)

- UI_Hub (Mediator)
  - Central communication hub for all UI screens.
  - Defines navigation signals: show_panel and show_toast.
  - Provides methods like goto_auth(), goto_vault(), open_editor_for(), and toast helpers.
  - Decouples screens from each other—no screen calls another directly.
- Main.gd (Concrete Mediator Controller)
  - Listens to UI_Hub signals and shows the correct screen.
  - Owns all major UI nodes (Auth, Register, VaultRoot, Recovery, etc.).
  - Implements _show_only() to control which panel is visible.
  - Relays user interactions to Session_Manager and UI_Hub.

Mediator (UI)

- Screens (Colleagues / Components)
  - Auth.gd, Register.gd, VaultRoot.gd, VaultItemEditor.gd, Recovery.gd, SelectItemType.gd
  - Each screen interacts only with the mediator, never with each other.
  - Trigger navigation by calling UI_Hub.goto_*() rather than referencing other scenes.
- ToastLayer (Colleague)
  - Shows toast notifications when UI_Hub emits show_toast().
  - Fully decoupled from UI logic or business logic.
- Result (Decoupled UI Architecture)
  - Screens never directly reference one another.
  - UI_Hub centralizes communication + navigation.
  - Main.gd handles the actual switching.
  - System is modular, scalable, and easy to modify.

Mediator (UI)


«Colleague»
Main
«Colleague»
Auth
«Colleague»
Register
«Colleague»
Recovery
«Colleague»
VaultRoot
«Colleague»
VaultItemEditor
«Colleague»
SelectItemType
«Colleague»
ToastLayer
show_toast
«Mediator»
UI_Hub
+PANEL_AUTH: String
+PANEL_REGISTER: String
+PANEL_VAULT: String
+PANEL_RECOVERY: String
+PANEL_VAULT_EDITOR: String
+PANEL_SELECT_ITEM_TYPE: String
+show_panel(panel_name: String, payload: Dictionary)
+show_toast(message: String, style: String)
+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
+notify_info(text: String): void
+notify_warning(text: String): void
+notify_error(text: String): void
indirectly via Main/Auth/Register/Recovery
via VaultRoot/VaultItemEditor
Session_Manager
Vault_Store
CryptoUtil
«file»
AccountJson
"account.json"
«file»
VaultEnc
"vault.enc"

Thank You

Please enjoy the demo :)