CIS 476 – Software Architecture & Patterns

MyPass – Secure Password Vault Application

Student:

Bryce Chudzik ID: 7440 7773

University of Michigan–Dearborn

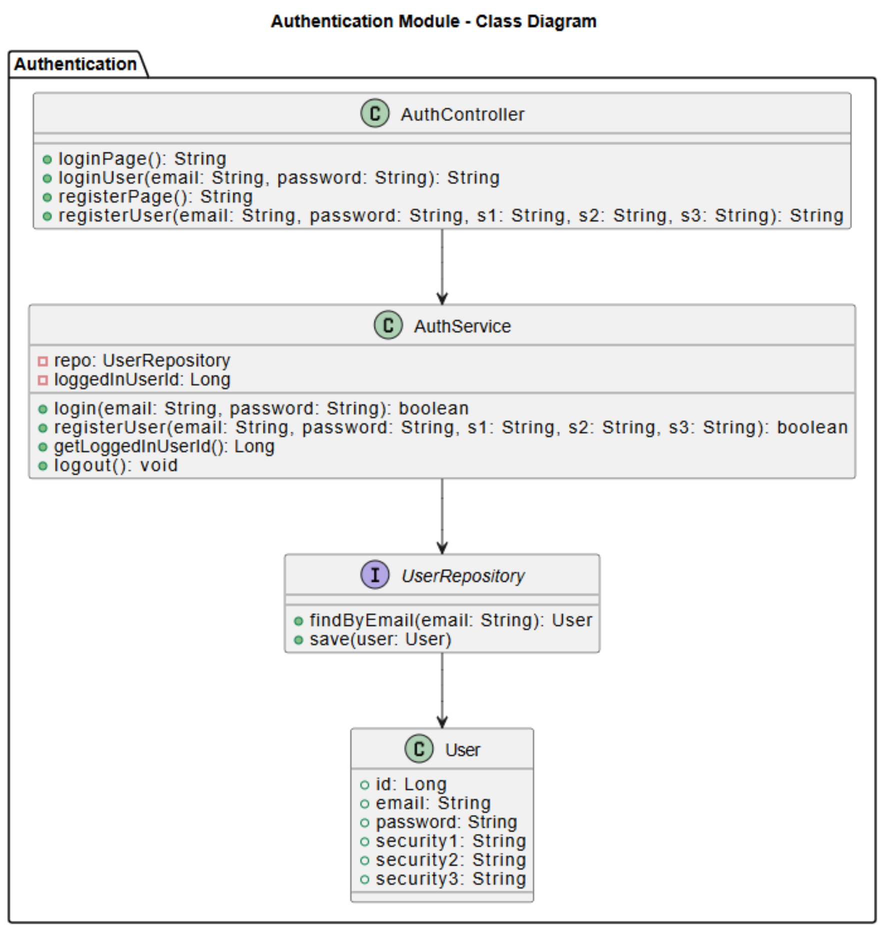
Instructor:

Professor Tommy

UML Diagrams

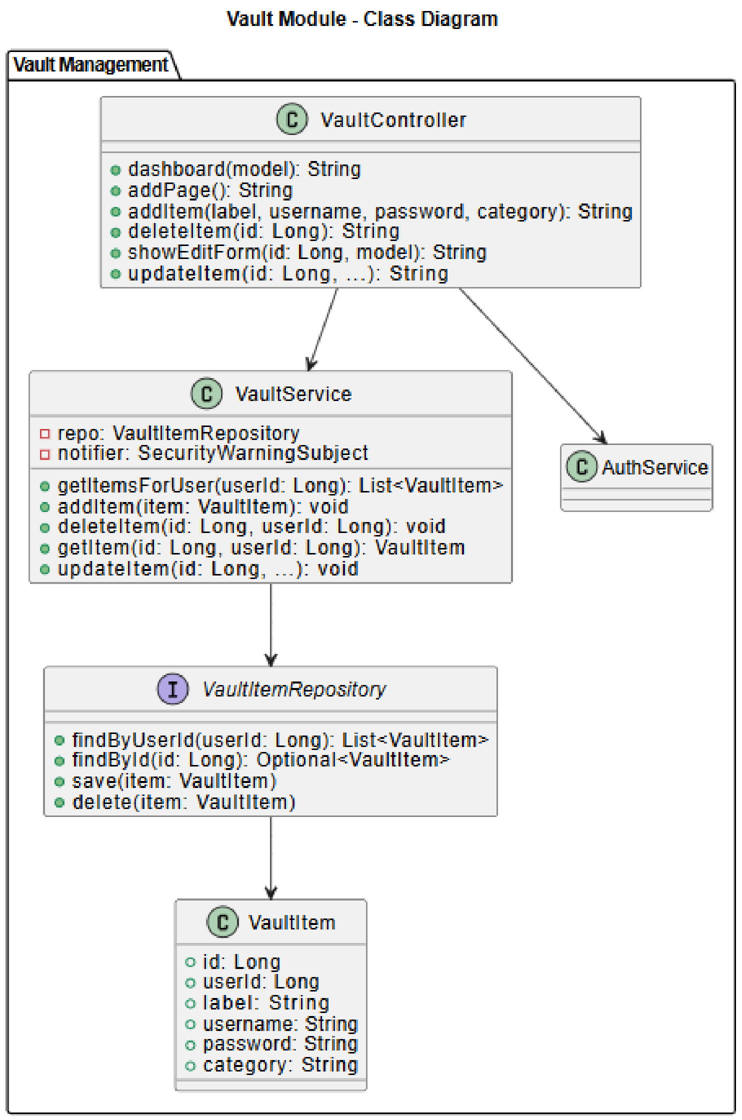
Class Diagram #1 – Authentication Module

The Authentication Module manages login, logout, and account creation.
`AuthController` handles HTTP requests for login and registration.
`AuthService` performs validation, password hashing (BCrypt), and manages session state by storing the logged-in user ID.
`UserRepository` persists users in the H2 database.
`User` represents a registered user with email, hashed password, and security questions.


Authentication Module - Class Diagram
Authentication
C AuthController
loginPage(): String
loginUser(email: String, password: String): String
registerPage(): String
registerUser(email: String, password: String, s1: String, s2: String, s3: String): String
C AuthService
repo: UserRepository
loggedInUserId: Long
login(email: String, password: String): boolean
registerUser(email: String, password: String, s1: String, s2: String, s3: String): boolean
getLoggedInUserId(): Long
logout(): void
I UserRepository
findByEmail(email: String): User
save(user: User)
C User
id: Long
email: String
password: String
security1: String
security2: String
security3: String

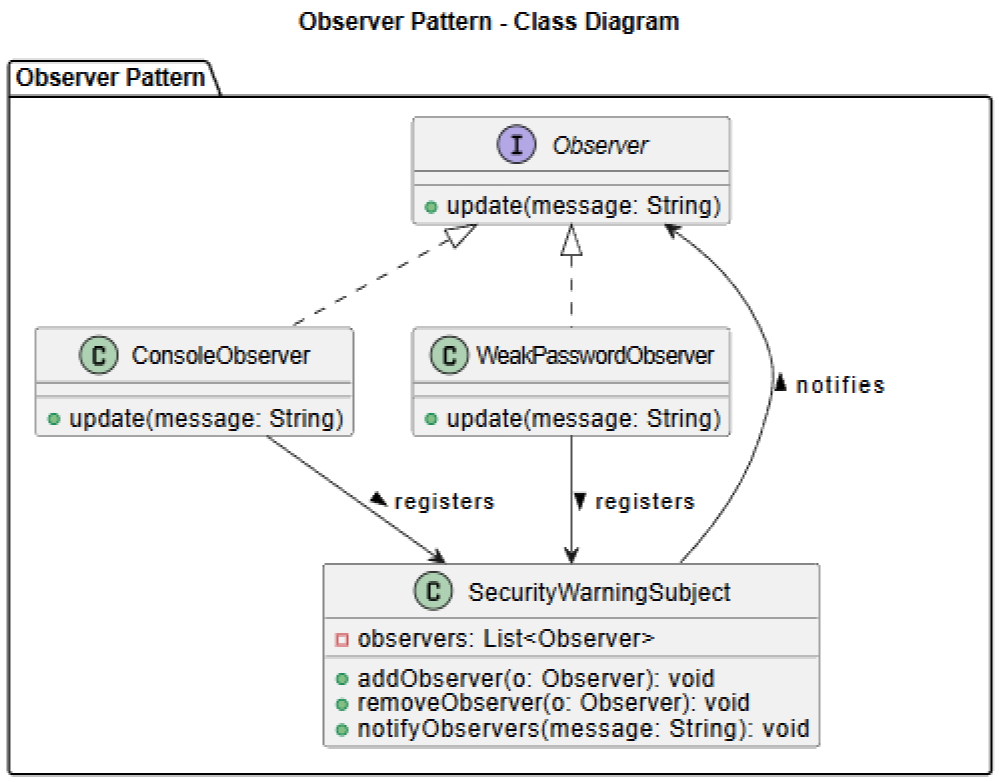
Class Diagram #2 – Vault Module

The Vault Module provides password-vault functionality, including viewing, adding, editing, and deleting stored credentials.
`VaultController` manages user interactions with the vault via dashboard pages and forms.
`VaultService` handles business logic and interacts with the repository.
`VaultItemRepository` accesses the vault table in the database.
`VaultItem` represents an individual stored credential entry.


Vault Module - Class Diagram
Vault Management
VaultController
dashboard(model): String
addPage(): String
addItem(label, username, password, category): String
deleteItem(id: Long): String
showEditForm(id: Long, model): String
updateItem(id: Long, ...): String
VaultService
repo: VaultItemRepository
notifier: SecurityWarningSubject
getItemsForUser(userId: Long): List<VaultItem>
addItem(item: VaultItem): void
deleteItem(id: Long, userId: Long): void
getItem(id: Long, userId: Long): VaultItem
updateItem(id: Long, ...): void
AuthService
VaultItemRepository
findByUserId(userId: Long): List<VaultItem>
findById(id: Long): Optional<VaultItem>
save(item: VaultItem)
delete(item: VaultItem)
VaultItem
id: Long
userId: Long
label: String
username: String
password: String
category: String

Class Diagram #3 – Observer Pattern Module

The Observer Pattern provides real-time monitoring for security-related events in the application.
`SecurityWarningSubject` maintains a list of observers and broadcasts notifications when vault items are added, updated, or deleted.
`ConsoleObserver` outputs security notices to the system console.
`WeakPasswordObserver` (optional extension) can detect weak or repeated passwords and issue warnings.
This pattern cleanly separates monitoring from core business logic.



Database Schema

The system uses a simple relational database with two tables: **Users** and **Vault_Items**. This schema supports authentication and secure storage of password entries for each user.

3.1 Tables

Users

Stores account and security-question data.

Field	Type	Description
`id`	BIGINT (PK)	Unique user ID
`email`	VARCHAR(255)	Login email (unique)
`password`	VARCHAR(255)	BCrypt-hashed password
`security1`	VARCHAR(255)	Security question #1
`security2`	VARCHAR(255)	Security question #2
`security3`	VARCHAR(255)	Security question #3

Vault_Items

Stores password-vault entries for each user.

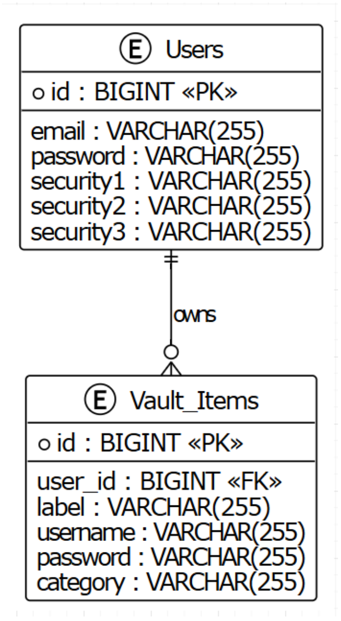
Field	Type	Description
`id`	BIGINT (PK)	Unique vault item ID
`user_id`	BIGINT (FK)	References Users(id)
`label`	VARCHAR(255)	Name of the saved credential

`username`	VARCHAR(255)	Username for the account
`password`	VARCHAR(255)	Stored password
`category`	VARCHAR(255)	Category (email, banking, etc.)

3.2 Relationship

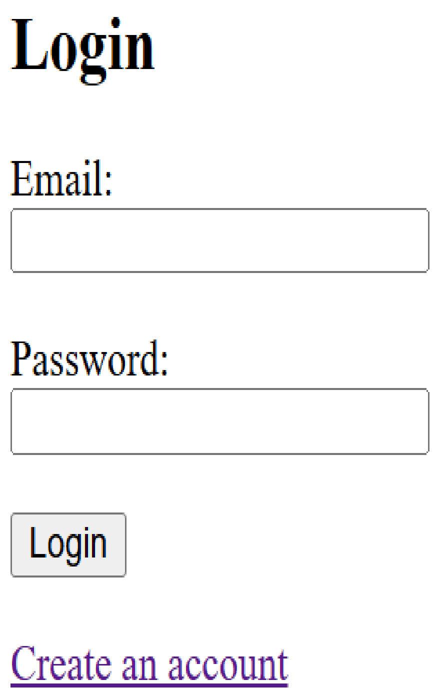
- **One User → Many Vault Items**
  Each vault entry belongs to exactly one user via the `user_id` foreign key.
  This ensures user data isolation and prevents cross-account access.

Schema



UI Screenshots

Login Page

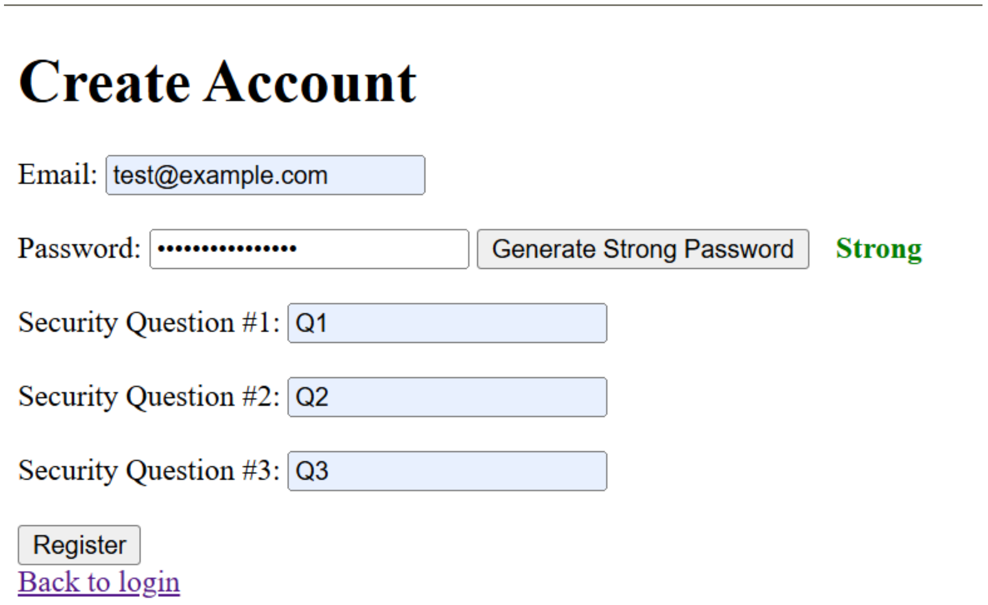


Login

Email:

Password:

Account Creation w/ Password Indicator and Password Generator



Create Account

Email: test@example.com

Password: ••••••••••••••• Generate Strong Password Strong

Security Question #1: Q1

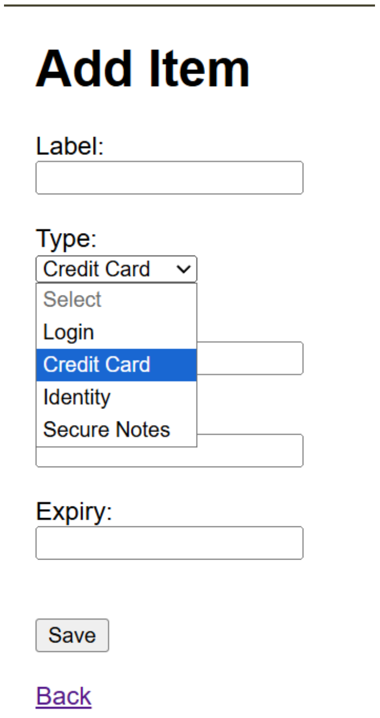
Security Question #2: Q2

Security Question #3: Q3

Register

Back to login

Adding Item w/ Custom Item Types and Password Strength Indicators



Add Item

Label:

Type:
Credit Card

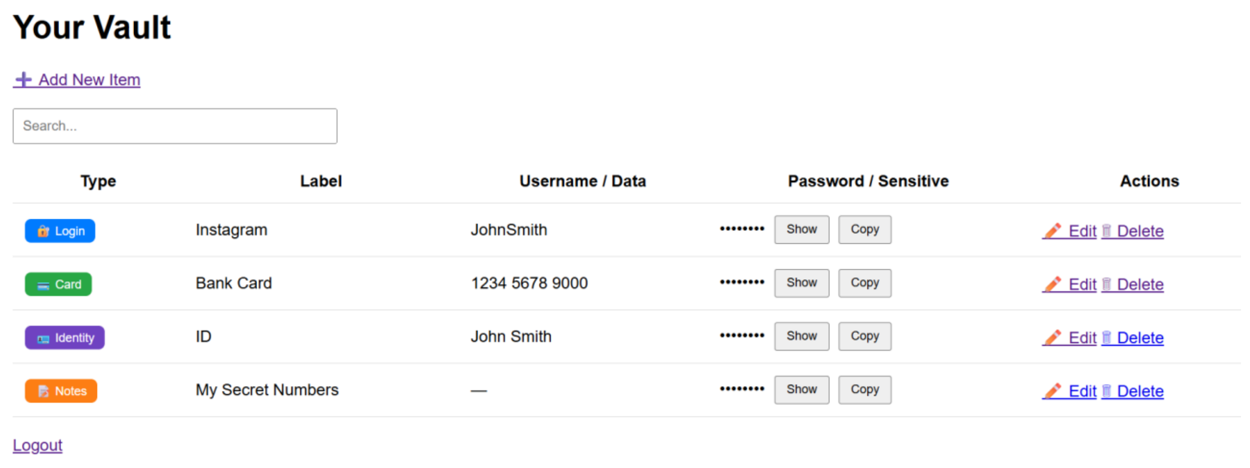
Select
Login
Credit Card
Identity
Secure Notes

Expiry:

Save

Back

Your Vault Dashboard w/ Actions



Your Vault

+ Add New Item

Search...

Type	Label	Username / Data	Password / Sensitive	Actions
Login	Instagram	JohnSmith	•••••••• Show Copy	Edit Delete
Card	Bank Card	1234 5678 9000	•••••••• Show Copy	Edit Delete
Identity	ID	John Smith	•••••••• Show Copy	Edit Delete
Notes	My Secret Numbers	—	•••••••• Show Copy	Edit Delete

Logout

References

Spring Boot. (2024). *Spring Boot Documentation*. Spring.io. https://docs.spring.io/spring-boot/

draw.io. (2024). *draw.io Diagramming Tool Documentation*. https://www.drawio.com/

Thymeleaf. (2024). *Thymeleaf Template Engine Documentation*. https://www.thymeleaf.org/