Password Manager - Design Manual

Technical Introduction

The Password Manager is a Java desktop application built using JavaFX for the UI. It follows the Model-View-Controller (MVC) architectural pattern to separate concerns and improve maintainability. The system uses AES(Advanced Encryption Standard) encryption to secure sensitive data and provides persistent file-based storage.

Key Technical Features

- Secure password storage using AES encryption
- User authentication with a master password
- Encrypted data
- Password generation and health analysis
- Category-based organization and search functionality

User Stories

Authentication

- As a new user, I want to create an account with a master password
- As a returning user, I want to log in with my master password
- As a user, I want to log out securely

Password Management

- As a user, I want to add, view, update, and delete password entries
- As a user, I want to copy usernames and passwords to clipboard

Organization

- As a user, I want to categorize, search, and filter my passwords

Security Features

- As a user, I want to generate strong passwords
- As a user, I want to identify weak or duplicate passwords

Incomplete User Stories

- Change master password

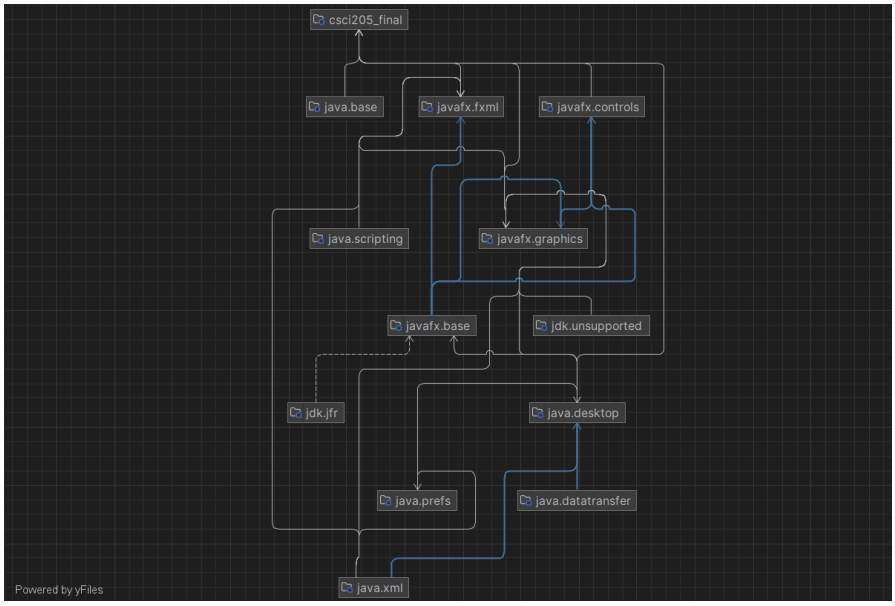
- Two-factor authentication
- Import/export functionality
- Cloud synchronization
- Mobile companion app

Object-Oriented Design

Core Design Principles

The application follows key OO principles:

- **Single Responsibility**: Each class has one primary purpose
- **Encapsulation**: Implementation details are hidden
- **Loose Coupling**: Minimal dependencies between components
- **High Cohesion**: Related functionality grouped together



Key Classes and Responsibilities

AuthenticationManager

- Manages user login/logout and session state
- Verifies credentials and loads user data
- Collaborates with: User, FileStorageManager, EncryptionManager

PasswordManager

- Manages operations for password entries
- Manages encryption/decryption of passwords

- Organizes passwords by category
- Collaborates with: AuthenticationManager, EncryptionManager, User

EncryptionManager

- Generates encryption keys
- Encrypts and decrypts sensitive data
- Collaborates with: FileStorageManager

FileStorageManager

- Saves and loads data to/from disk
- Manages file paths and I/O operations
- Collaborates with: EncryptionManager

User

- Stores username and master password
- Maintains a collection of password entries
- Organizes passwords into categories
- Collaborates with: PasswordEntry

PasswordEntry

- Contains website, username, and encrypted password
- Tracks metadata (creation/modification dates)
- Provides update methods

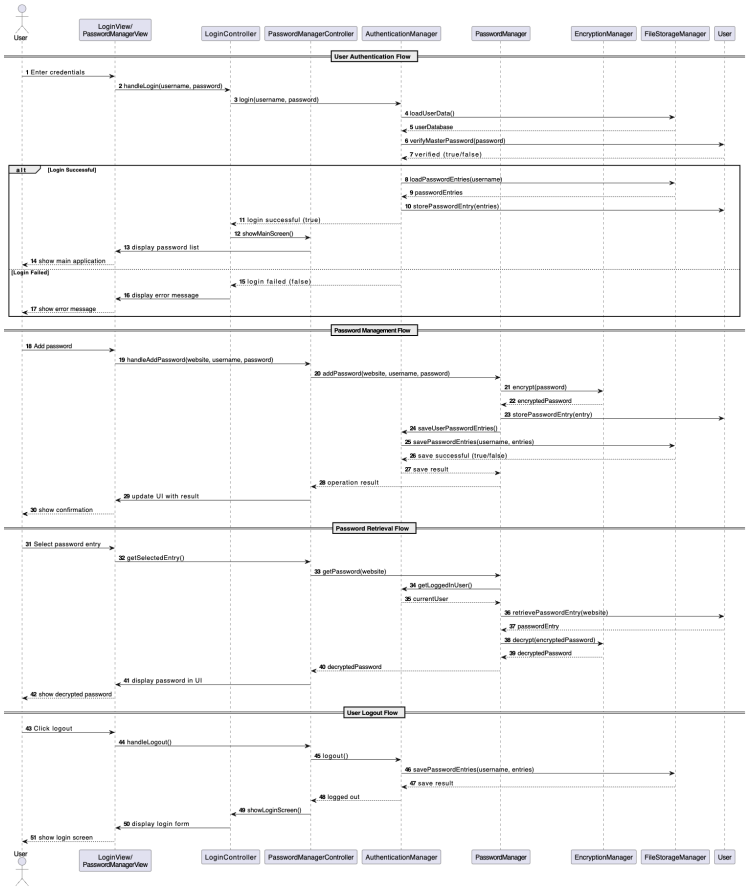
PasswordGenerator

- Creates random, secure passwords
- Ensures complexity requirements

PasswordHealthAnalyzer

- Identifies weak passwords
- Detects duplicate passwords
- Tracks password age

Component Interactions



Authentication Flow

1. LoginController captures credentials from LoginView
2. AuthenticationManager verifies against stored user data
3. On success, loads the user's password entries
4. Initializes PasswordManagerController with user data

Password Storage Flow

1. PasswordManagerController captures new password details
2. PasswordManager encrypts the password via EncryptionManager
3. New PasswordEntry is created and added to User's collection
4. Updated data is saved to disk via FileStorageManager

Password Retrieval Flow

1. PasswordManagerController requests a specific password
2. PasswordManager retrieves the encrypted entry

3. EncryptionManager decrypts the password
4. Decrypted password is displayed in the UI

Security Implementation

Encryption

The system uses AES (Advanced Encryption Standard) with 128-bit keys:

public String encrypt(String password) {

try {

Cipher cipher = Cipher.getInstance(ALGORITHM); cipher.init(Cipher.ENCRYPT_MODE, encryptKey); byte[] encryptedBytes = cipher.doFinal(password.getBytes());

return Base64.getEncoder().encodeToString(encryptedBytes);

} catch (Exception e) {

throw new RuntimeException("Error encrypting password", e);

}

}

Secure Storage

- Master passwords are never stored in plaintext
- All password entries are encrypted before storage
- Encryption key is secured in the user's home directory
- Sensitive data is cleared from memory after use

Data Persistence

Storage Structure

~/.passwordmanager/

├── encryption.key

├── userdata.dat

├── username1/ │

```
        └── passwords.dat
└── username2/
        └── passwords.dat
```

File Formats

- User data: Simple format with usernames and encrypted master passwords
- Password entries: CSV-like format with encrypted fields
- Key storage: Base64-encoded AES key

UI Implementation

The user interface follows the MVC pattern:

- **Models**: Data objects (User, PasswordEntry)
- **Views**: FX files and View classes
- **Controllers**: Logic classes that mediate between Model and View

Key UI Components

- LoginView/Controller: Handles authentication
- PasswordManagerView/Controller: Manages the main interface
- Custom dialogs for password operations
- Table views for password listing
- Detail panels for viewing password information

Future Enhancements

Security Enhancements

- Two-factor authentication
- Enhanced encryption options
- Secure password sharing

Feature Enhancements

- Cloud synchronization
- Browser integration
- Mobile companion app
- Import/export functionality

UI Improvements

- Customizable themes
- Improved accessibility
- Enhanced password strength visualization