INTE2512 OBJECT-ORIENTED PROGRAMMING
FINAL PROJECT

SHINOBU STORE

DO KHOA NGUYEN - S3978796

TRAN THE QUANG MINH - S3979562

NGUYEN SON TUNG - S3979348

BUI ANH MINH - S3979513

NGUYEN MINH KHOA - S3980823

Table of Contents

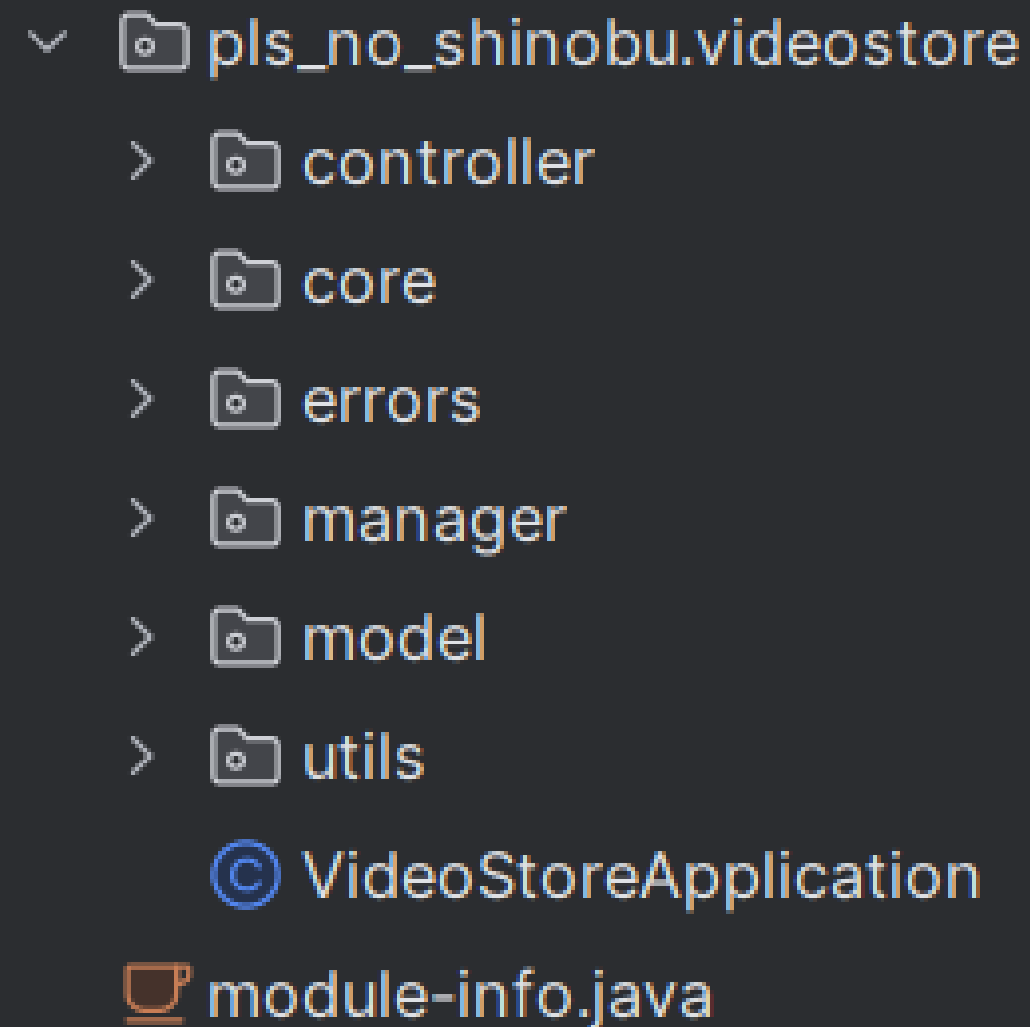1. ## Backend

   - Folder structure
   - Model
   - Manager
   - Core

2. ## GUI

   - Folder structure
   - welcome.fxml
   - signup.fxml
   - signin.fxml
   - userDashboard.fxml
   - adminDashboard.fxml
   - updateUser.fxml
   - addItem.fxml
   - updateItem.fxml

Backend - Folder structure

The folder structure:

```
∨ ⊡ pls_no_shinobu.videostore
  > ⊡ controller
  > ⊡ core
  > ⊡ errors
  > ⊡ manager
  > ⊡ model
  > ⊡ utils
    Ⓒ VideoStoreApplication
  ☕ module-info.java
```

- controller: contains controllers for fxml files.
- core: contains classes for core operations.
- errors: contains classes for exceptions specified for this application.
- manager: contains classer for managing multiple model objects.
- model: contains classes for models.
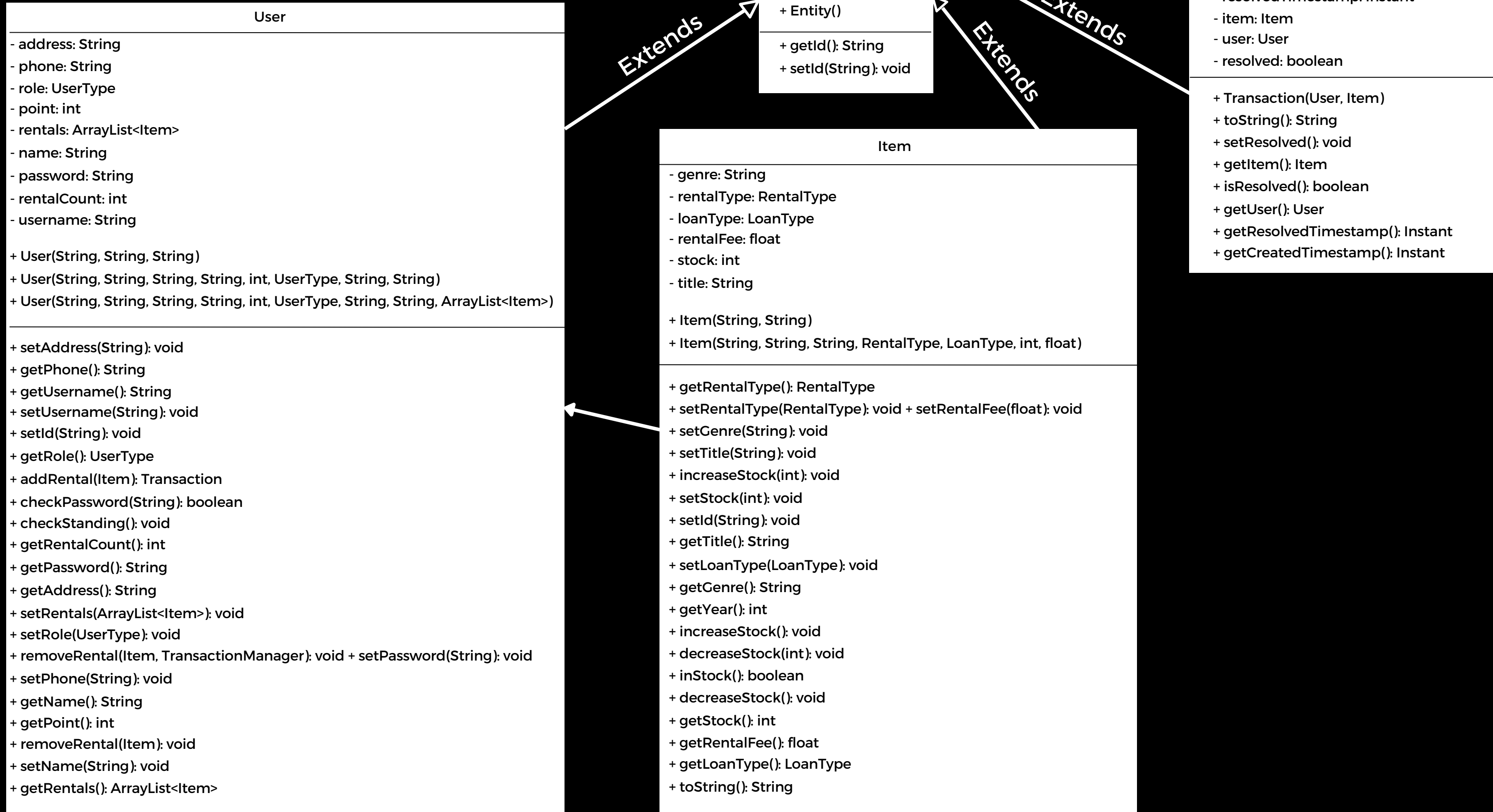- utils: contains classes for utilities for parsing users and items from CSV files and password handling.

Backend - Model

We started with implementing classes for:

- **User**: Include basic informations, auth informations and rental informations.
- **Item**: Include the item details and the stock with its methods.
- **Transaction**: Store transaction data such as the User, Item, its status and create-resolve timestamp in Instant.

These classes are inherited from the **Entity** class which requires an ID that can be extended.

Backend - Model

Entity
- id: String

+ Entity(String)
+ Entity()

+ getId(): String
+ setId(String): void

User
- address: String
- phone: String
- role: UserType
- point: int
- rentals: ArrayList<Item>
- name: String
- password: String
- rentalCount: int
- username: String

+ User(String, String, String)
+ User(String, String, String, String, int, UserType, String, String)
+ User(String, String, String, String, int, UserType, String, String, ArrayList<Item>)

+ setAddress(String): void
+ getPhone(): String
+ getUsername(): String
+ setUsername(String): void
+ setId(String): void
+ getRole(): UserType
+ addRental(Item): Transaction
+ checkPassword(String): boolean
+ checkStanding(): void
+ getRentalCount(): int
+ getPassword(): String
+ getAddress(): String
+ setRentals(ArrayList<Item>): void
+ setRole(UserType): void
+ removeRental(Item, TransactionManager): void + setPassword(String): void
+ setPhone(String): void
+ getName(): String
+ getPoint(): int
+ removeRental(Item): void
+ setName(String): void
+ getRentals(): ArrayList<Item>

Item
- genre: String
- rentalType: RentalType
- loanType: LoanType
- rentalFee: float
- stock: int
- title: String

+ Item(String, String)
+ Item(String, String, String, RentalType, LoanType, int, float)

+ getRentalType(): RentalType
+ setRentalType(RentalType): void + setRentalFee(float): void
+ setGenre(String): void
+ setTitle(String): void
+ increaseStock(int): void
+ setStock(int): void
+ setId(String): void
+ getTitle(): String
+ setLoanType(LoanType): void
+ getGenre(): String
+ getYear(): int
+ increaseStock(): void
+ decreaseStock(int): void
+ inStock(): boolean
+ decreaseStock(): void
+ getStock(): int
+ getRentalFee(): float
+ getLoanType(): LoanType
+ toString(): String

Transaction
- createdTimestamp: Instant
- resolvedTimestamp: Instant
- item: Item
- user: User
- resolved: boolean

+ Transaction(User, Item)
+ toString(): String
+ setResolved(): void
+ getItem(): Item
+ isResolved(): boolean
+ getUser(): User
+ getResolvedTimestamp(): Instant
+ getCreatedTimestamp(): Instant
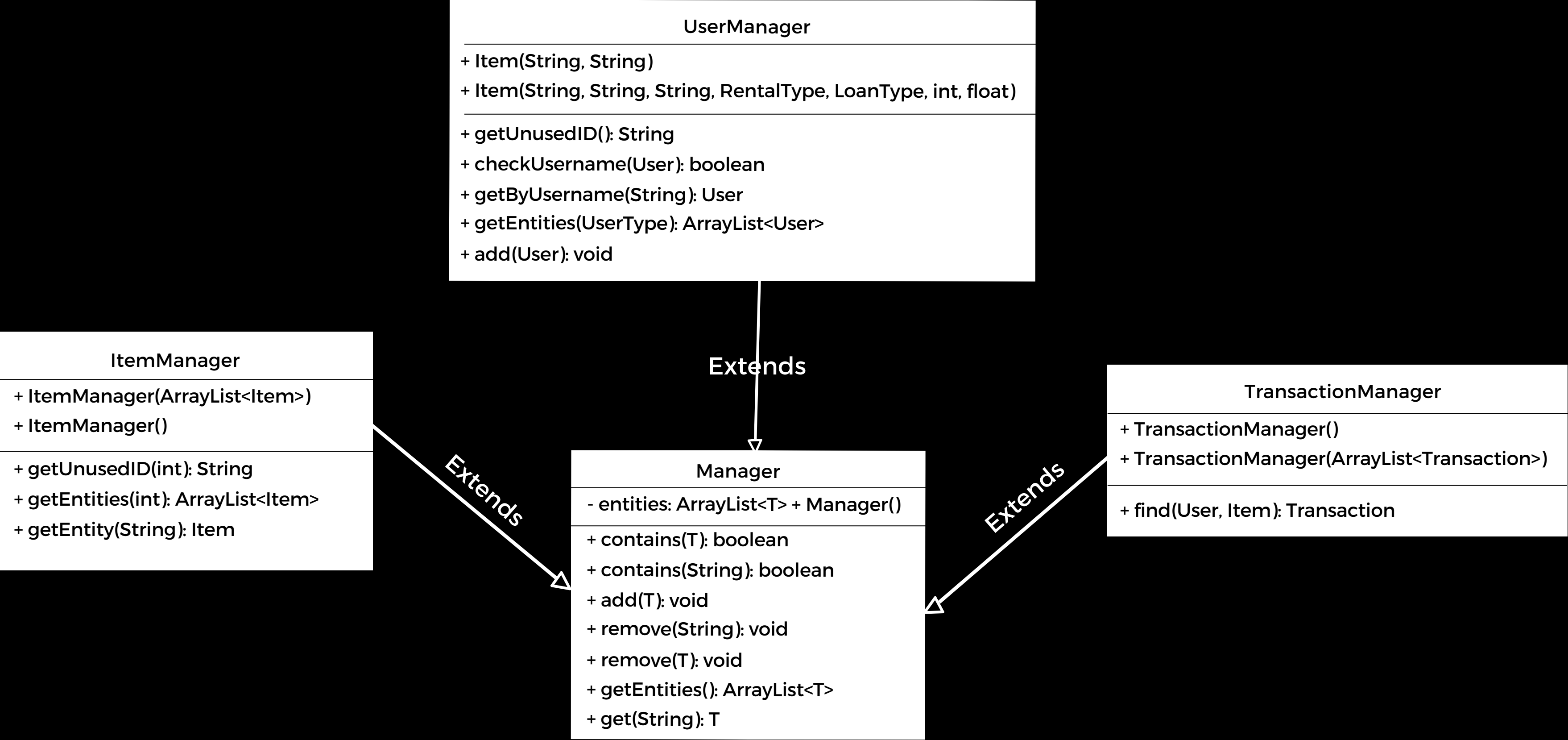
Extends
Extends
Extends

Backend - Manager

We then implement classes for managing multiple objects of the models, in which we can create, delete or to provide information:

- **UserManager**: Class for manage multiple User.
- **ItemManager**: Class for manage multiple Item.
- **TransactionManager**: Class for manage multiple Transaction.

These classes are inherited from the **Manager** class, which is an abstract class for implementing Entity manager.
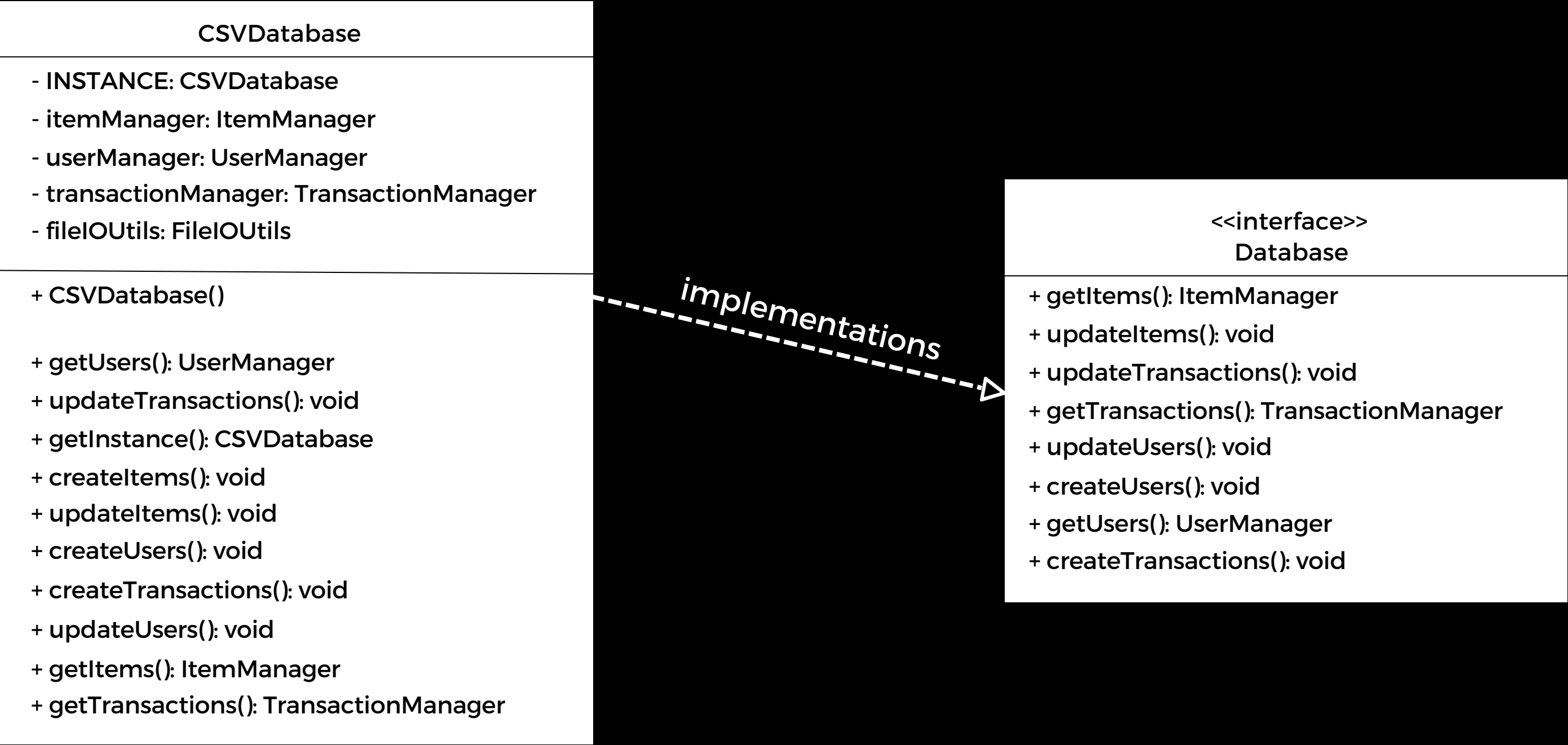
Backend - Manager

UserManager

+ Item(String, String)

+ Item(String, String, String, RentalType, LoanType, int, float)

+ getUnusedID(): String

+ checkUsername(User): boolean

+ getByUsername(String): User

+ getEntities(UserType): ArrayList<User>

+ add(User): void

ItemManager

+ ItemManager(ArrayList<Item>)

+ ItemManager()

+ getUnusedID(int): String

+ getEntities(int): ArrayList<Item>

+ getEntity(String): Item

Manager

- entities: ArrayList<T> + Manager()

+ contains(T): boolean

+ contains(String): boolean

+ add(T): void

+ remove(String): void

+ remove(T): void

+ getEntities(): ArrayList<T>

+ get(String): T

TransactionManager

+ TransactionManager()

+ TransactionManager(ArrayList<Transaction>)

+ find(User, Item): Transaction

Extends

Extends

Extends

Backend - Core

We then implement classes for the main operations of the application:

- CSVDatabase: implements the Database interface, handle CRUD operations for the CSV database.
- Session: provide credentials for sign in and log out.

Backend - Core

CSVDatabase

- INSTANCE: CSVDatabase
- itemManager: ItemManager
- userManager: UserManager
- transactionManager: TransactionManager
- fileIOUtils: FileIOUtils

+ CSVDatabase()

+ getUsers(): UserManager
+ updateTransactions(): void
+ getInstance(): CSVDatabase
+ createItems(): void
+ updateItems(): void
+ createUsers(): void
+ createTransactions(): void
+ updateUsers(): void
+ getItems(): ItemManager
+ getTransactions(): TransactionManager

implementations ----▷

**<<interface>>
Database**

+ getItems(): ItemManager
+ updateItems(): void
+ updateTransactions(): void
+ getTransactions(): TransactionManager
+ updateUsers(): void
+ createUsers(): void
+ getUsers(): UserManager
+ createTransactions(): void

GUI - Folder structure

```
∨ 🗀 pls_no_shinobu.videostore
    > 🗀 css
    > 🗀 img
    </> addItem.fxml
    </> adminDashboard.fxml
    </> signin.fxml
    </> signup.fxml
    </> updateItem.fxml
    </> updateUser.fxml
    </> userDashboard.fxml
    </> welcome.fxml
```

- css: contains css files
- img: contains image files
- 8 fxml files
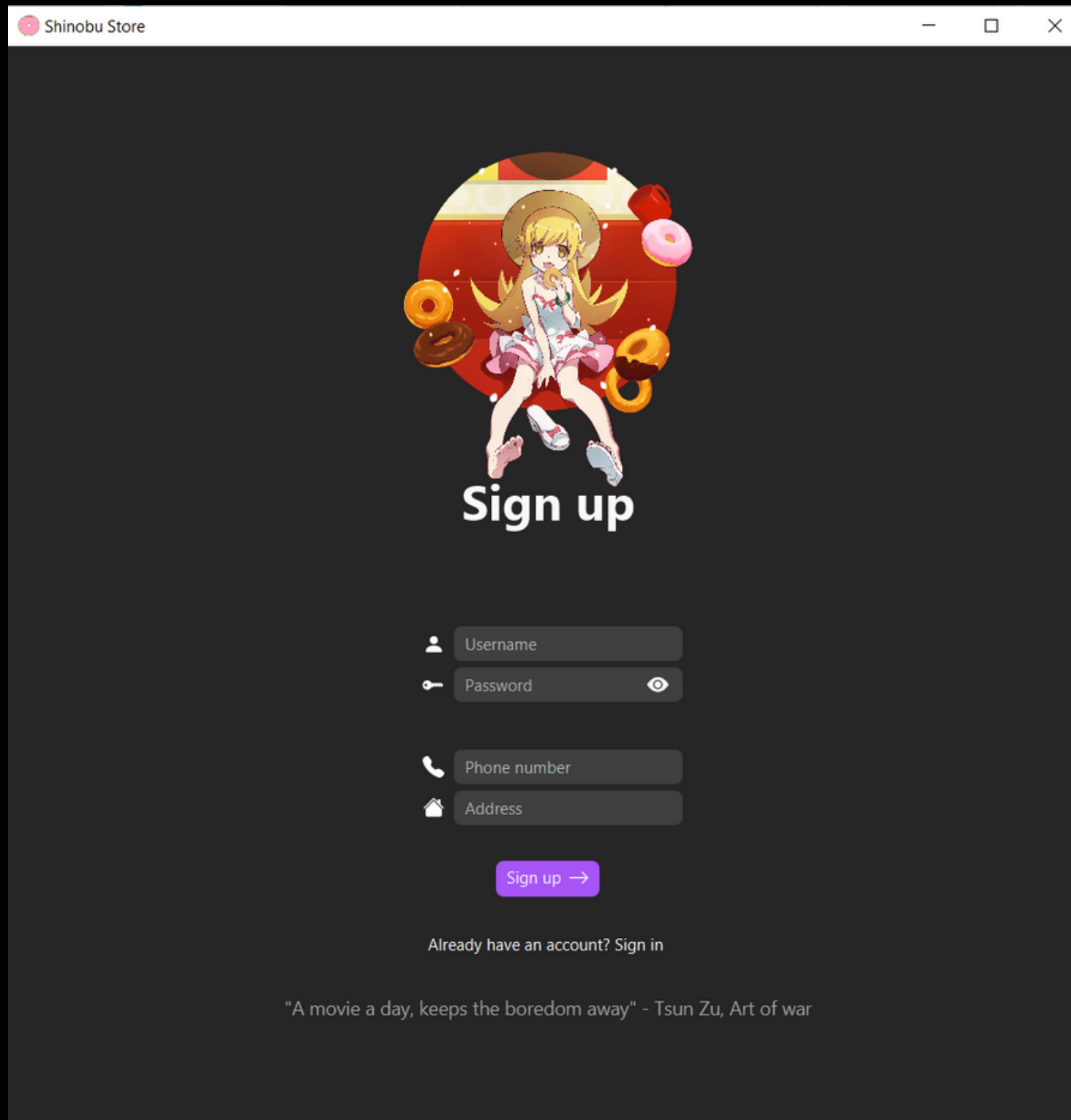
GUI - welcome.fxml
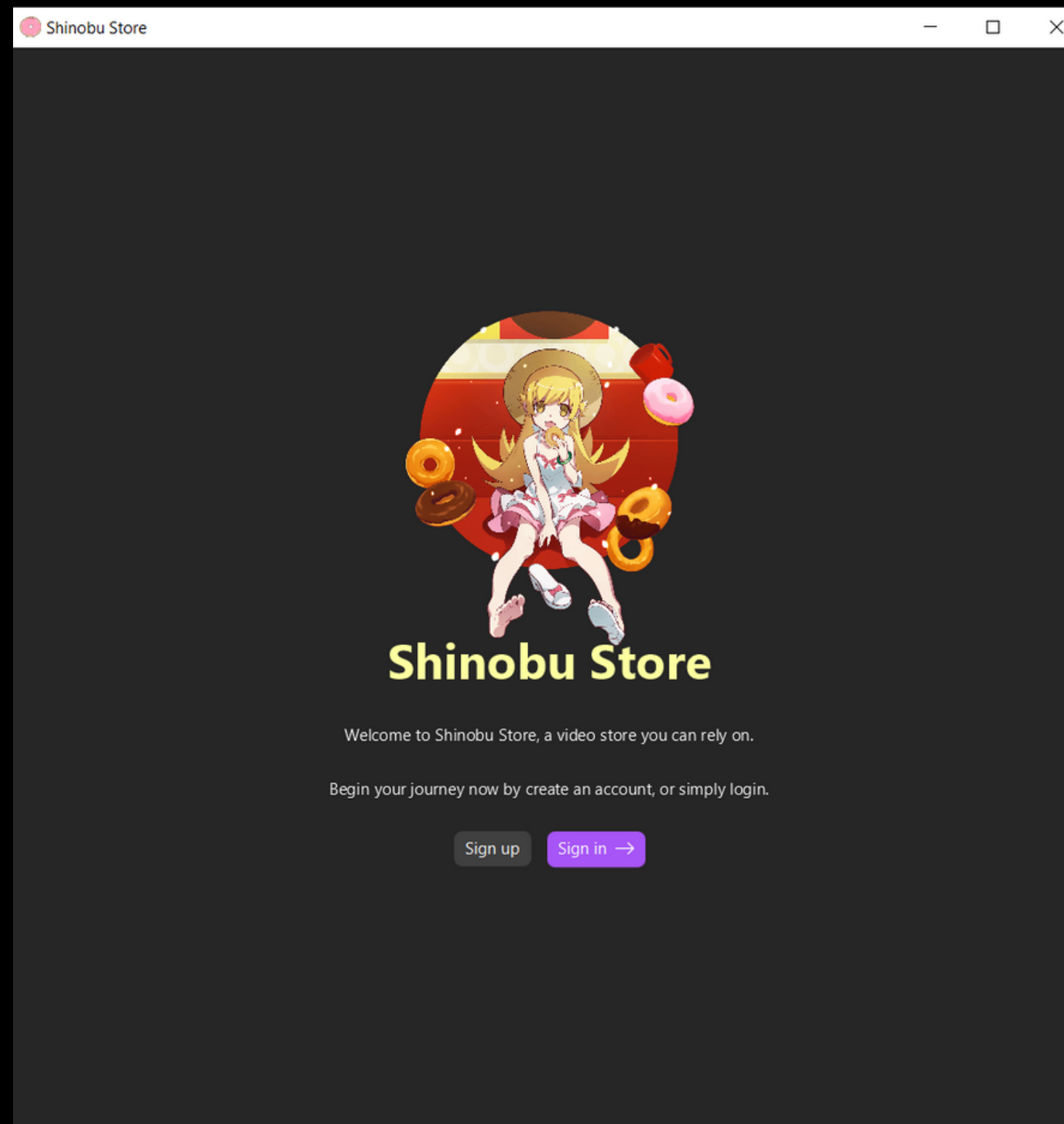
Features:
- Index page of the Shinobu Store
- Buttons link to Sign in and Sign up pages

GUI - signup.fxml

Features:
- Text fields for the user to input their information.
- Button to add the user to the database then log the user in.
- All inputs are validated
- Button link to Sign in page.

GUI - signin.fxml
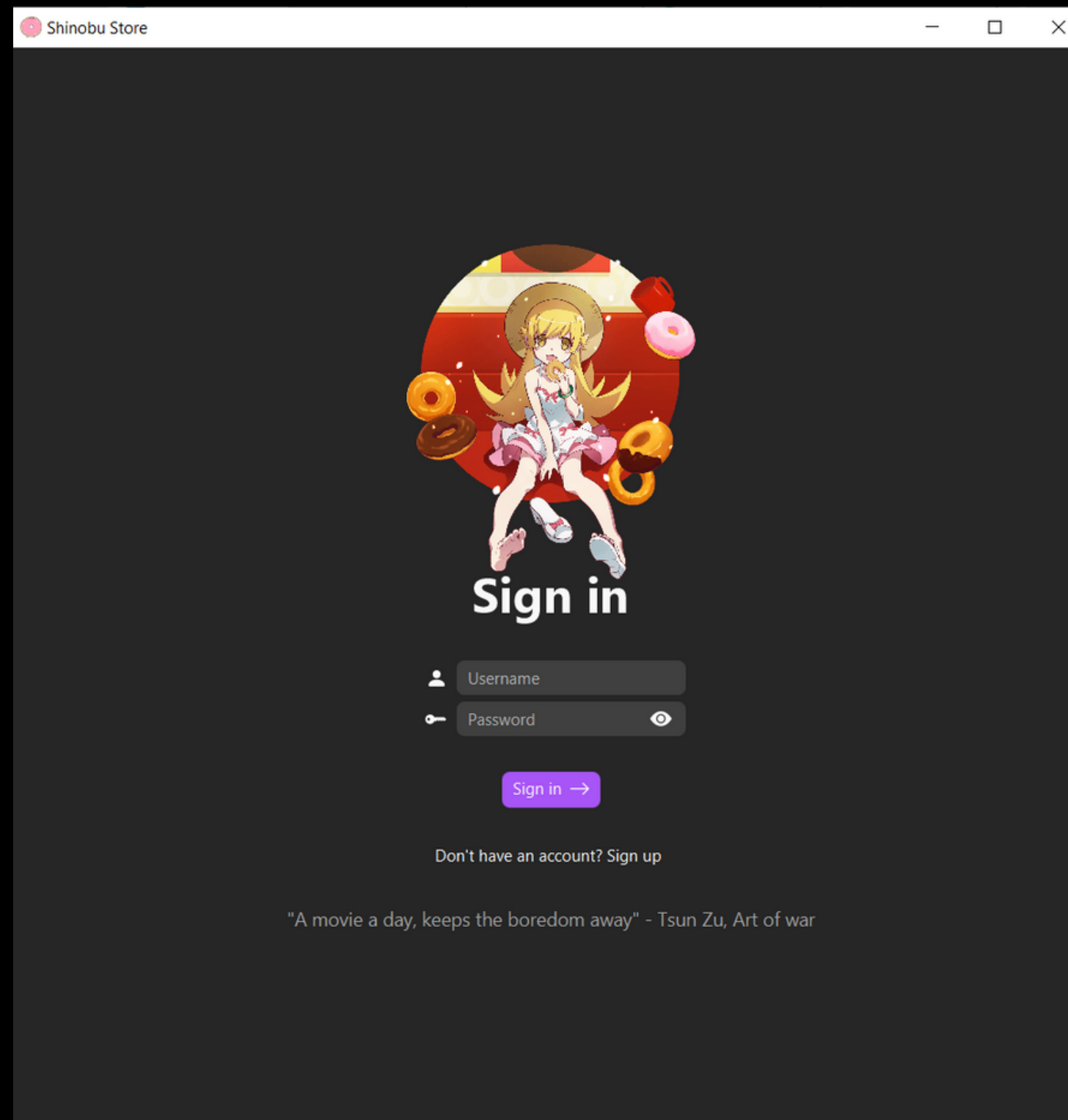
Features:
- Text fields for the user to input their username and password.
- An icon for toggling visibility of the password field
- Button to log in and create a session if the provided informations are correct.

GUI - userDashboard.fxml
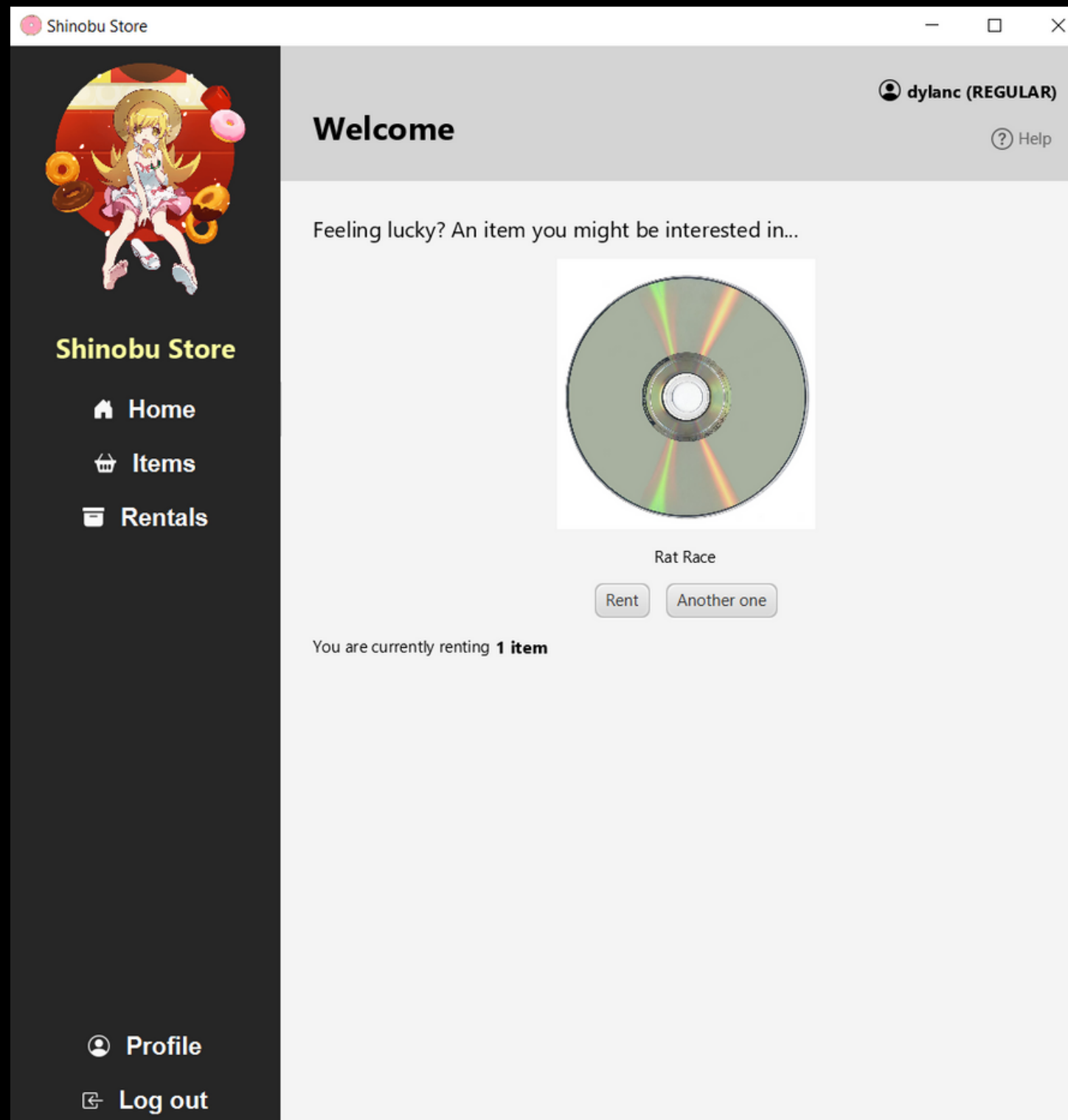
Features:
- Welcome page that recommend a random item to the user. Also display the number of item the user currently renting.
- Sideboard with buttons allows the user to navigate the app.
- Display the username and their membership on the top right.
- 'Help' button.

GUI - userDashboard.fxml
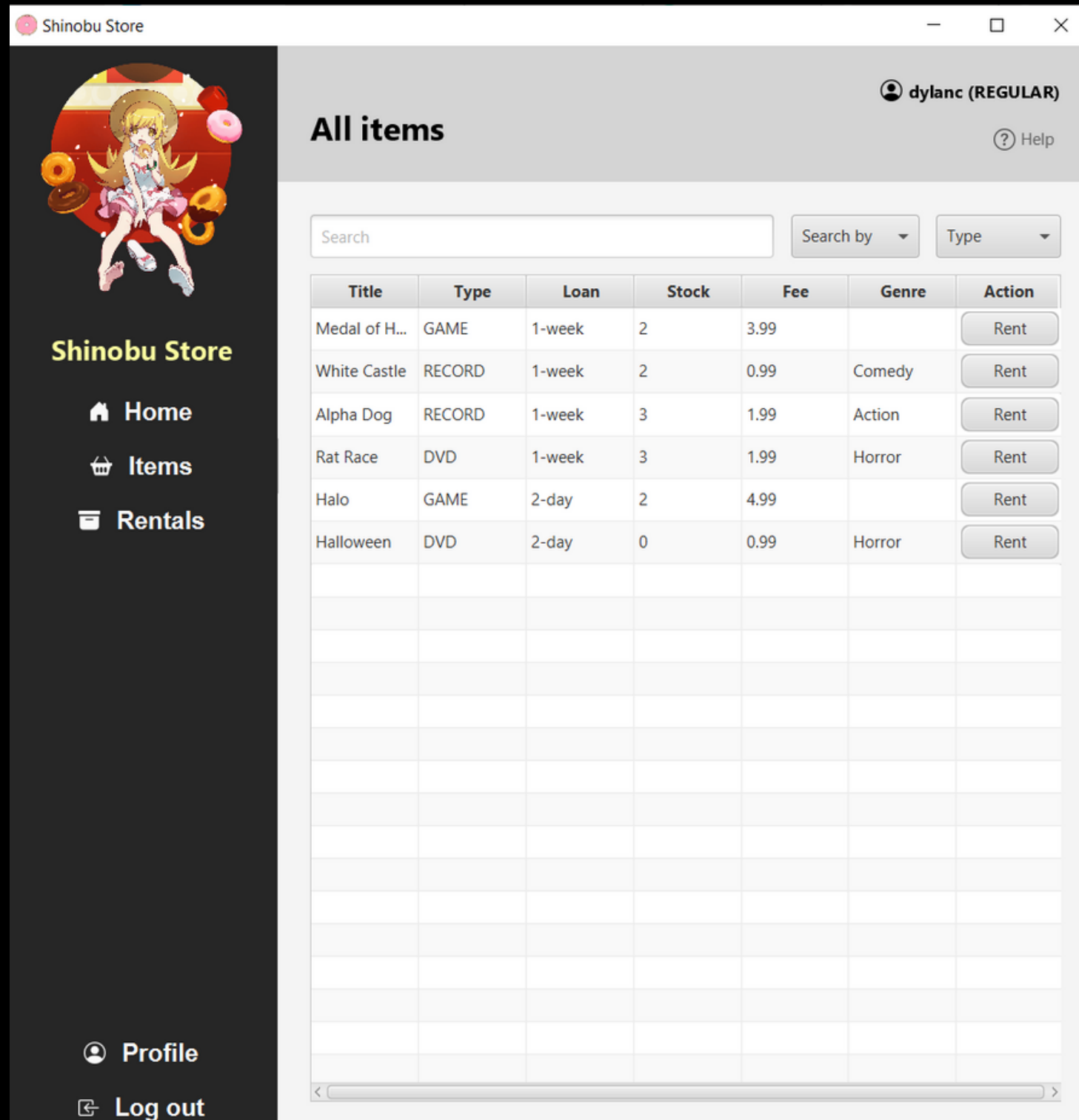
Features:
- This page corresponds with the 'Items' button, displays all items currently in the store.
- Search bar, allows the user to search for items by either IDs or names.
- Filter function, displays the items by their type.

GUI - userDashboard.fxml
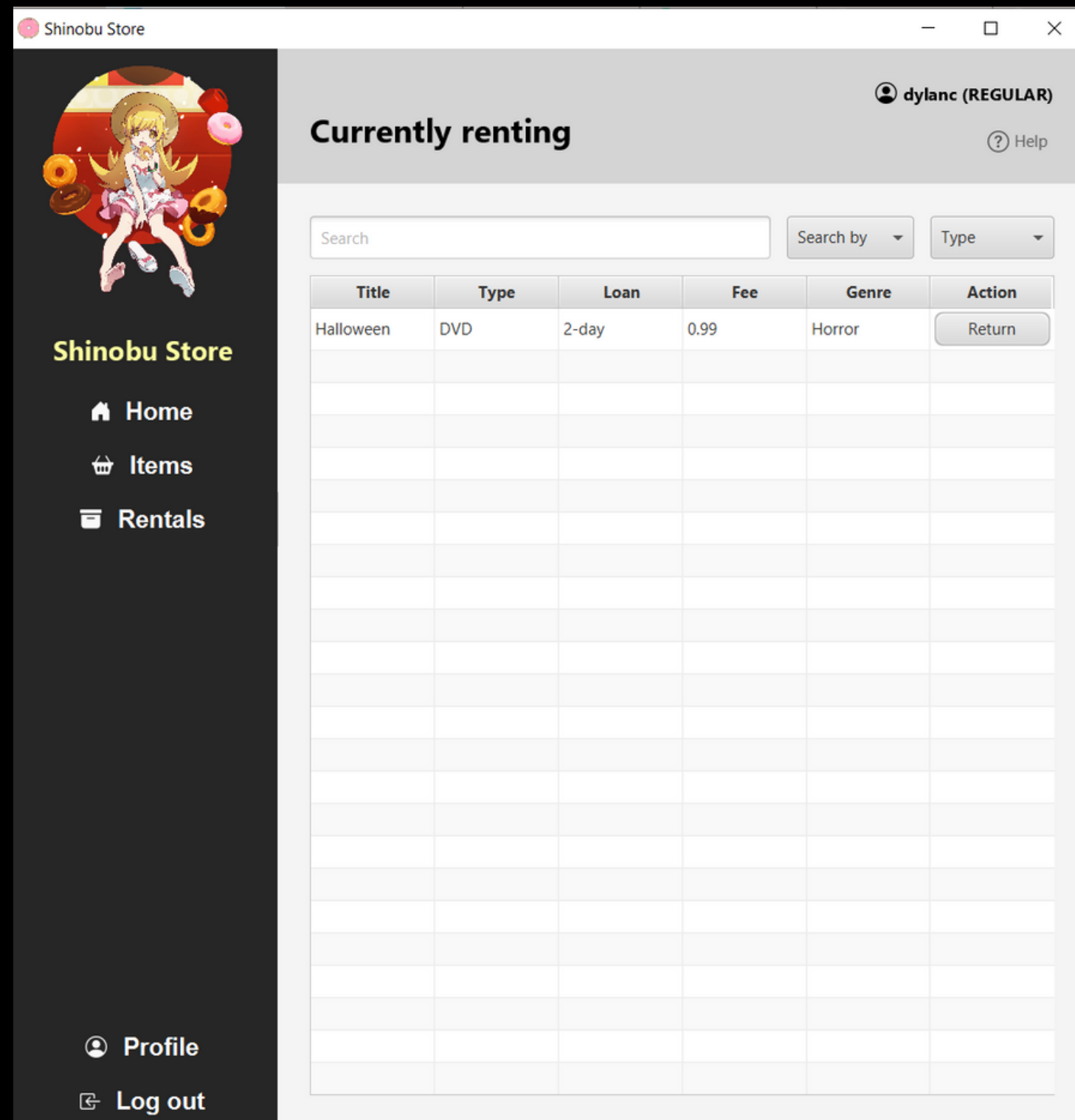
Features:
- This page corresponds with the 'Rentals' button, displays all the user's current rented item.
- Same search and filter function as the last page.
- Button that allows the user to return the item.

GUI - userDashboard.fxml
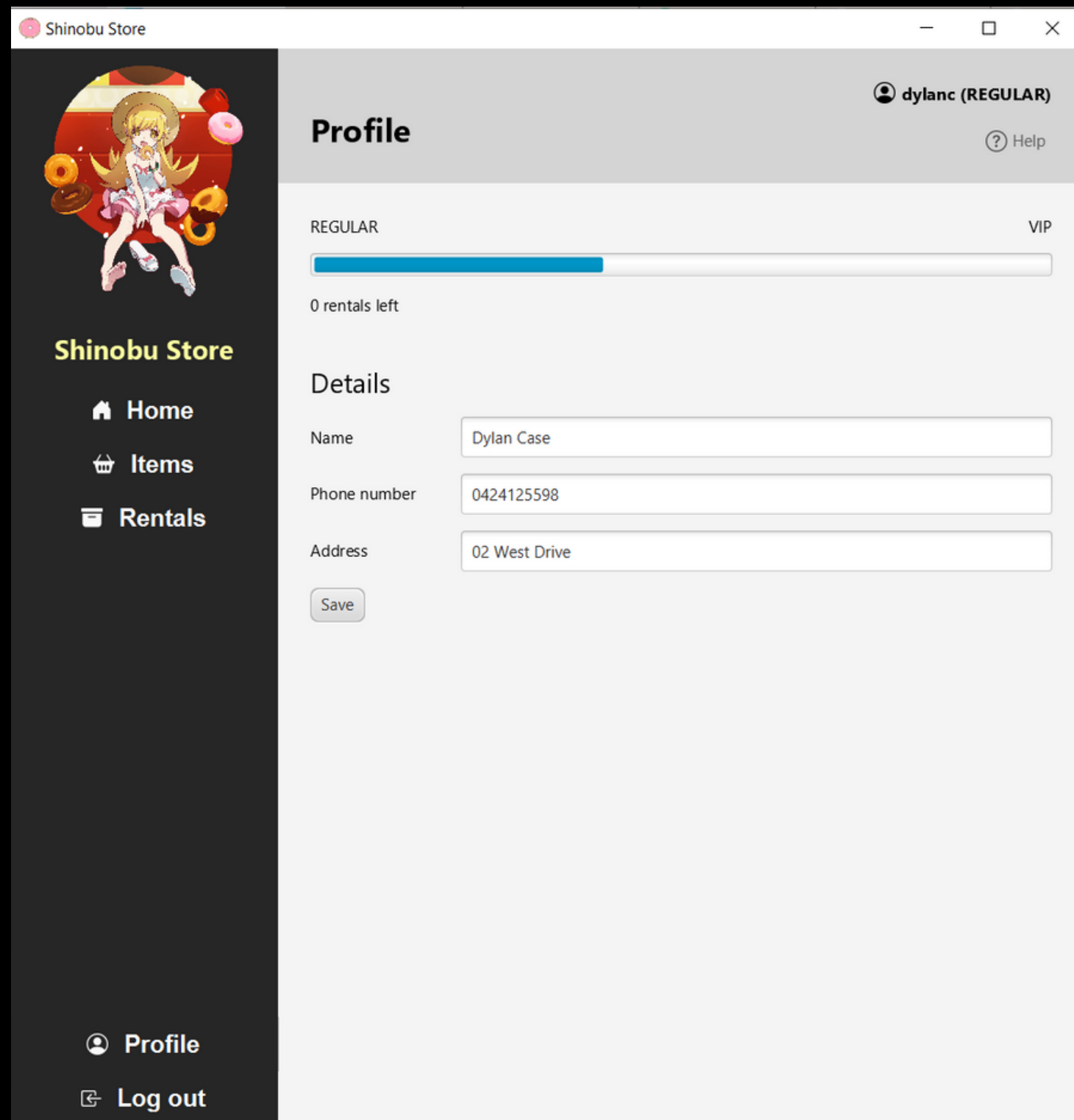
Features:
- This page corresponds with the 'Profile' button, displays the user's info and their current membership progression.
- Allows the user to edit their information.

GUI - userDashboard.fxml
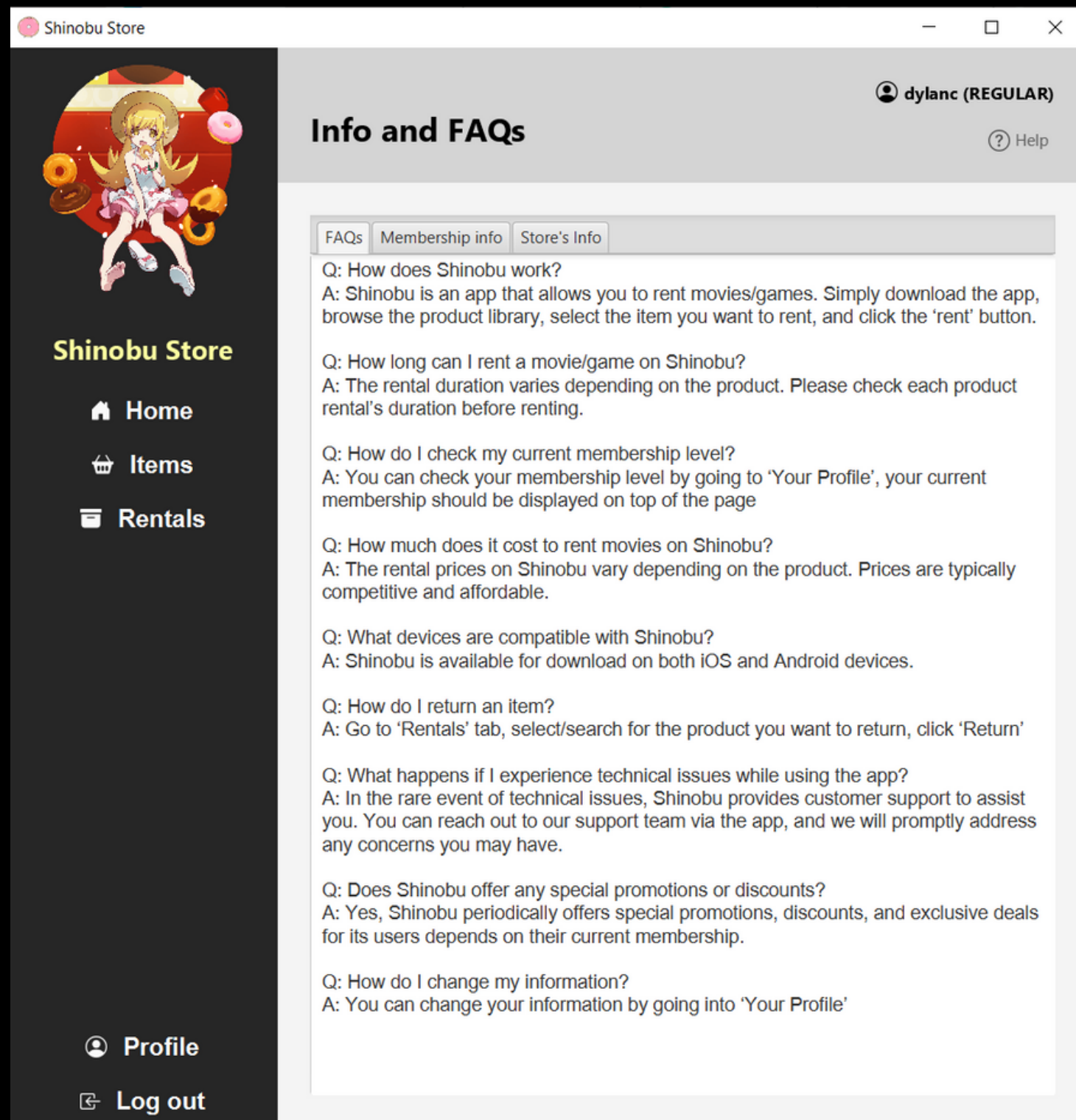
Features:
- This page corresponds with the 'Help' button, which displays FAQs, membership rules and store information.
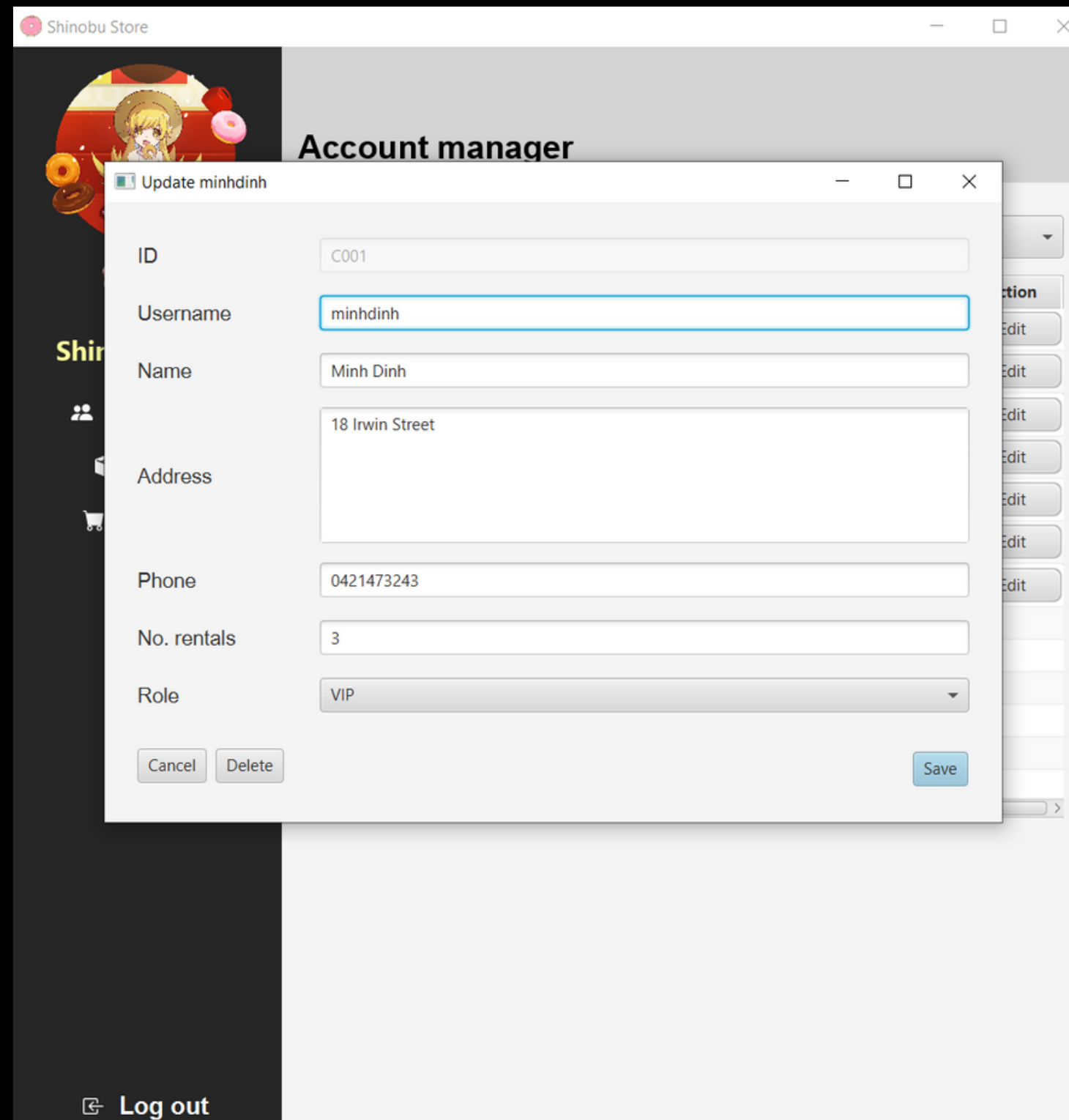
GUI - adminDashboard.fxml
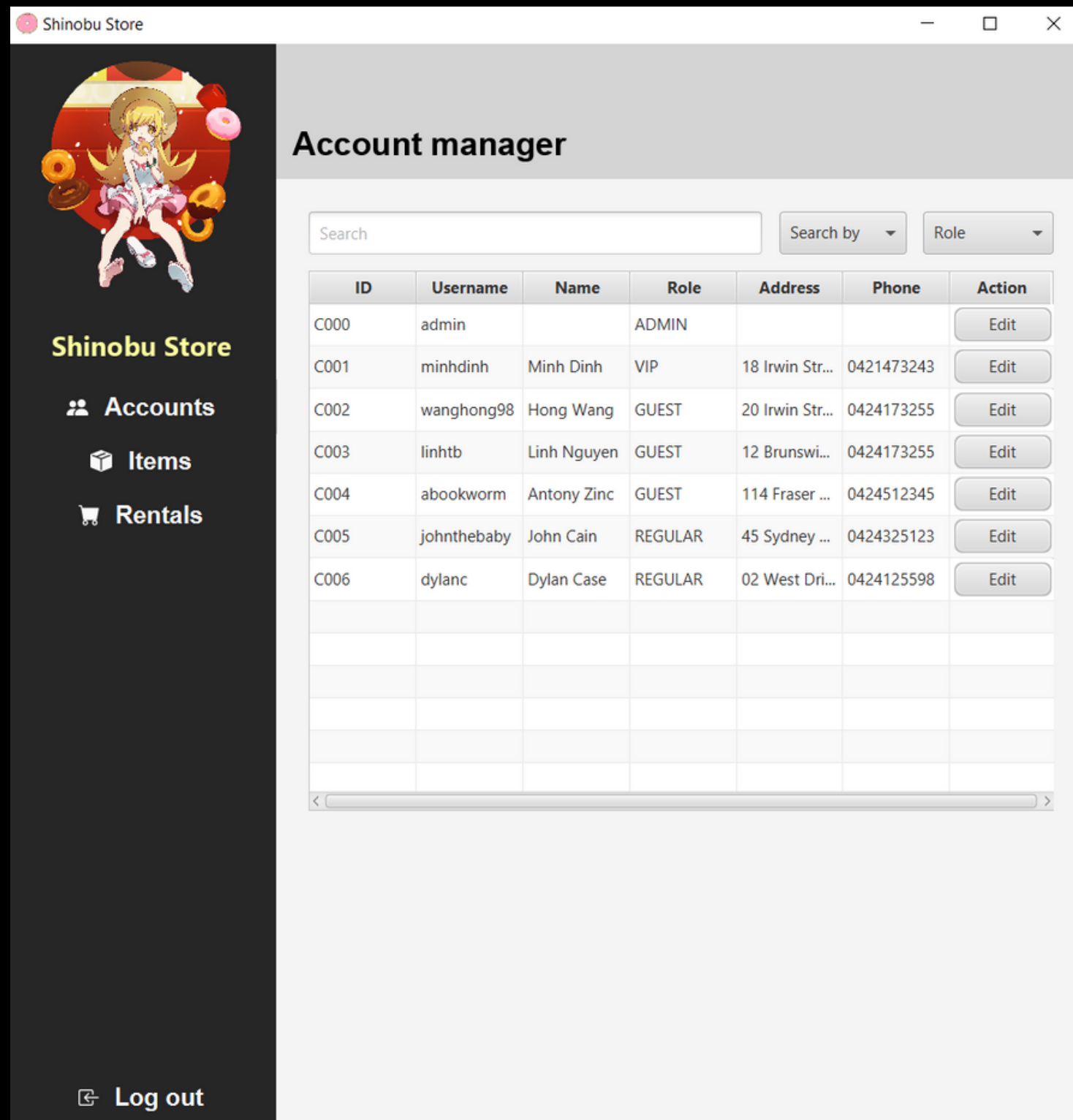
Features:
- Sideboard with buttons to let the admin navigates the app.
- This page correspond with the 'Accounts' button, displays all the existing accounts.
- Search bar that allows the admin to search for users by name or ID.
- Filter function which displays the users by their membership level.
- Button to edit the user's info.

GUI - updateUser.fxml

Features:

- Pop up when the admin clicked the 'Edit button', user's info can be updated here or delete the user out of the database.
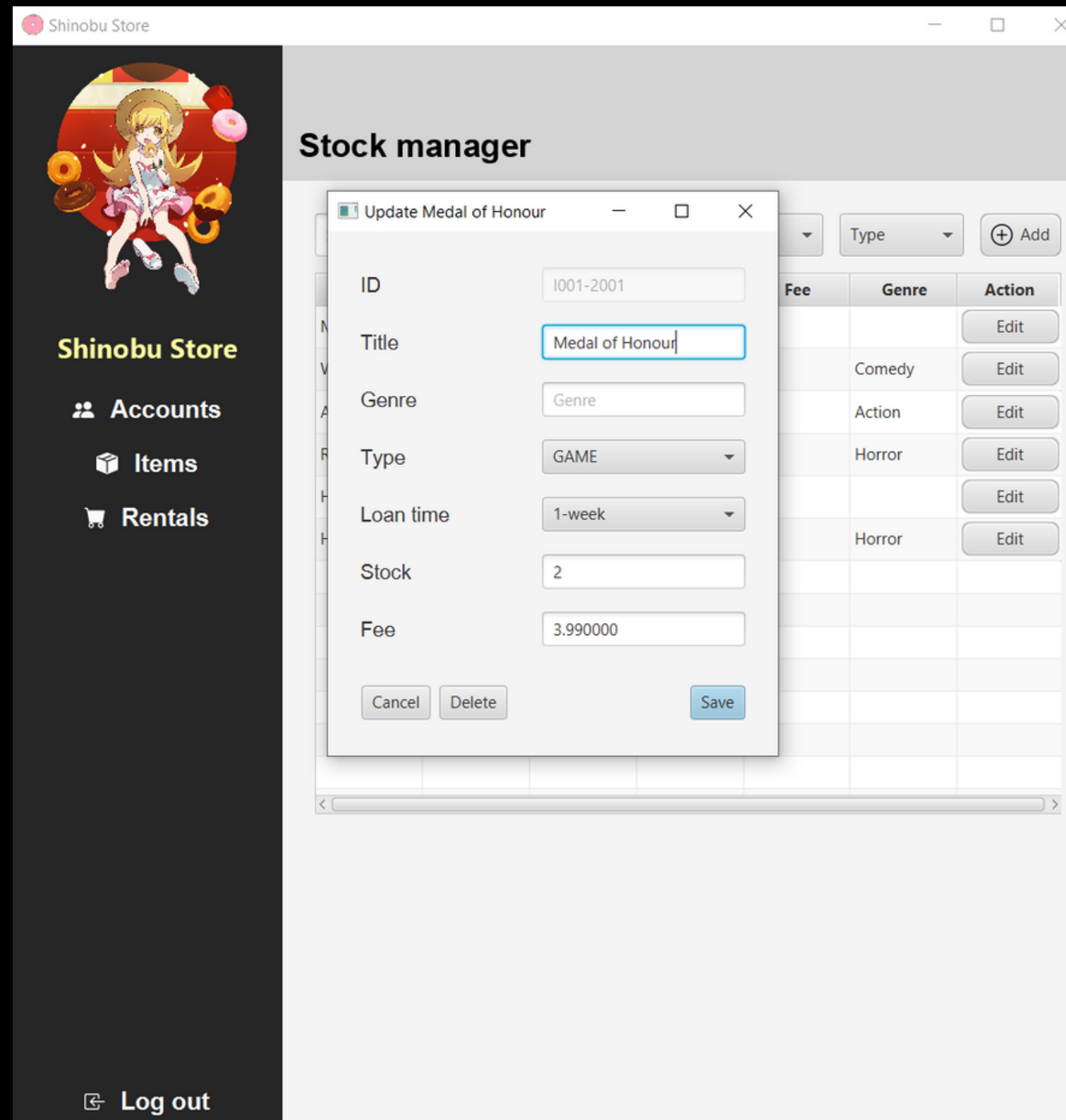
GUI - adminDashboard.fxml
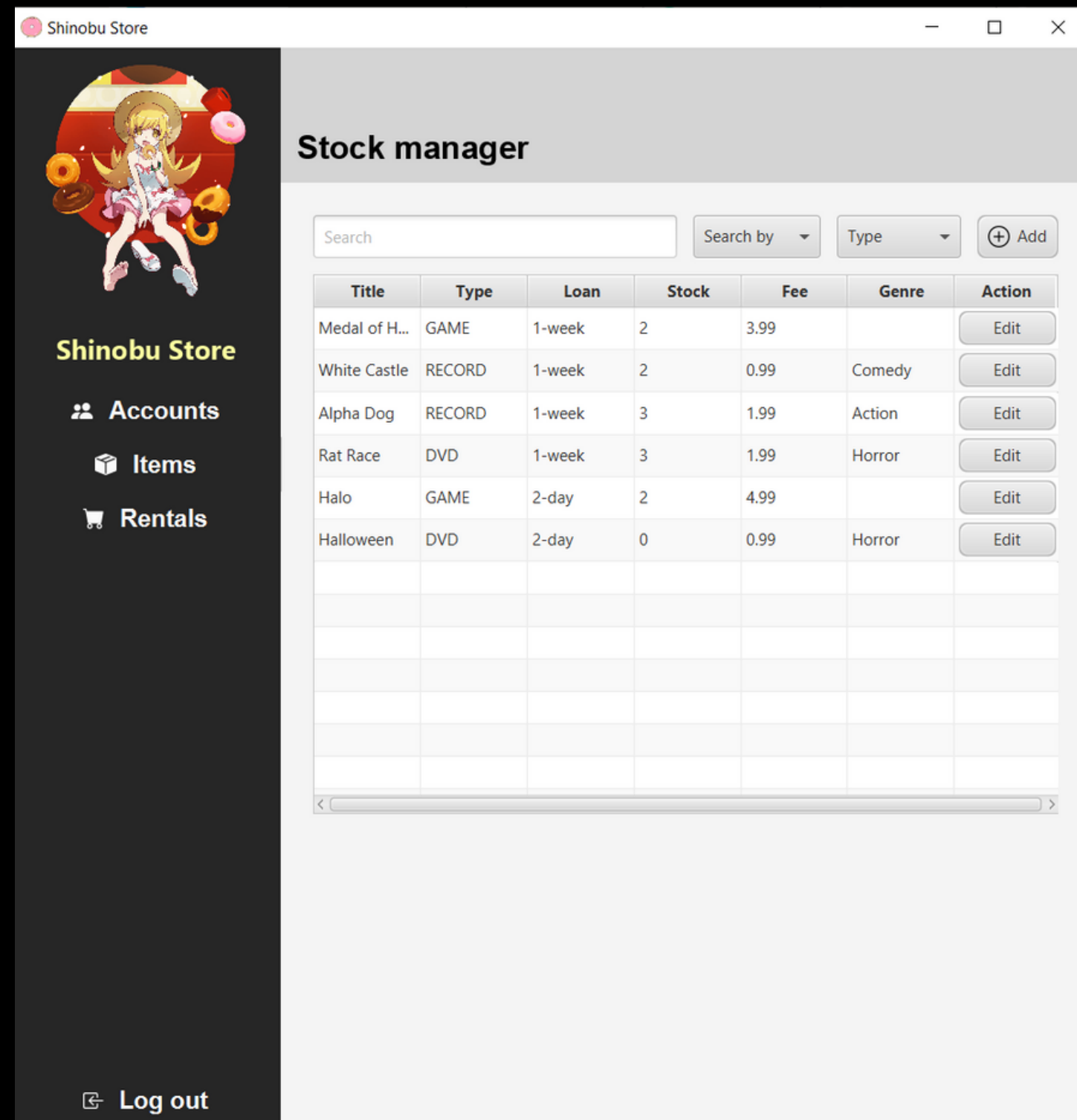
Features:
- This page corresponds with the 'Items' button, allows the admin to manage the store's stock.
- Same search and filter as the user.
- Button to add more item into the stock.
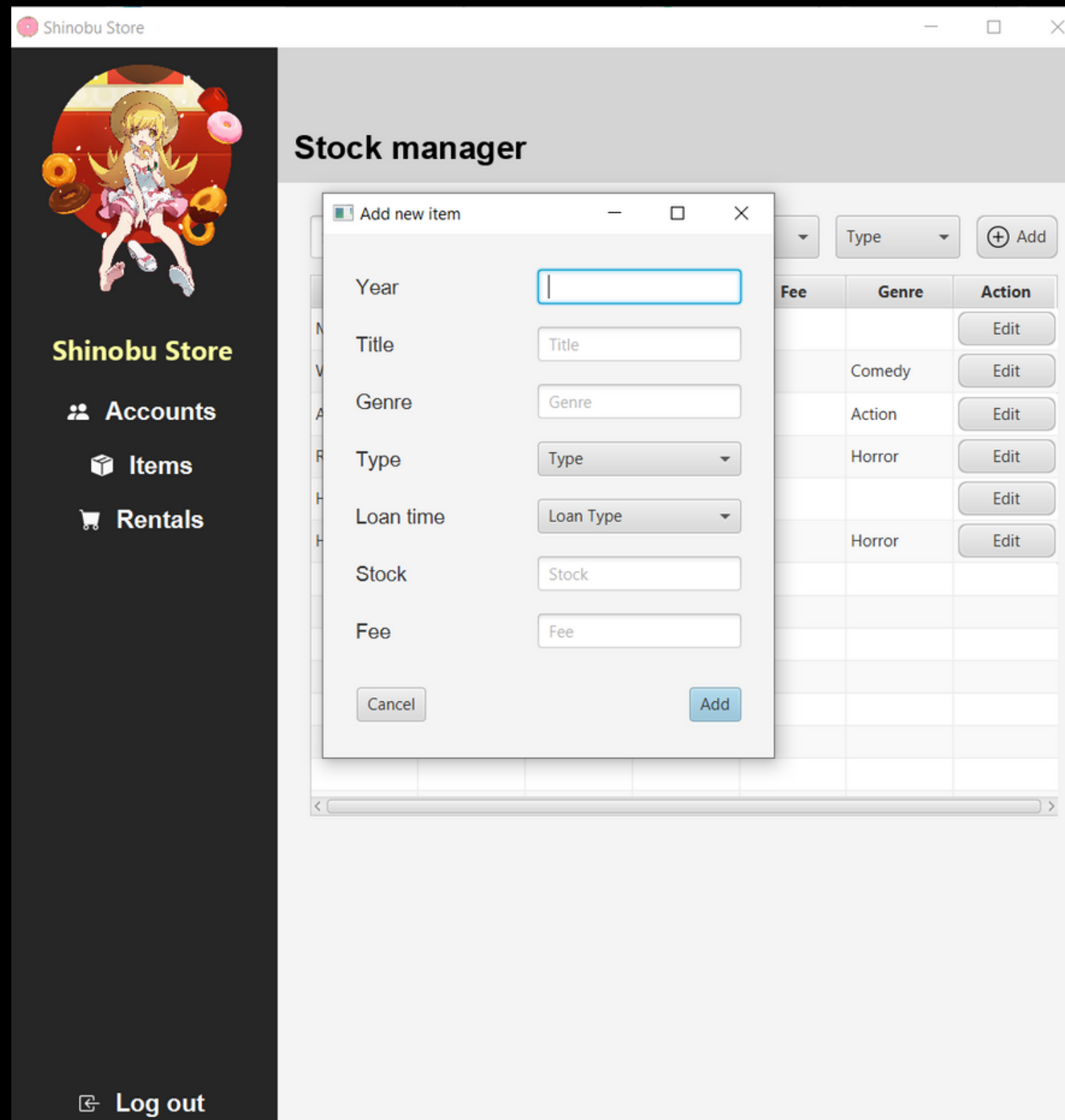- Button to edit each item's info.

GUI - updateItem.fxml

Features:
- Pop up when the 'Edit' button is clicked, allows the admin to update the item's info or delete them from the database.
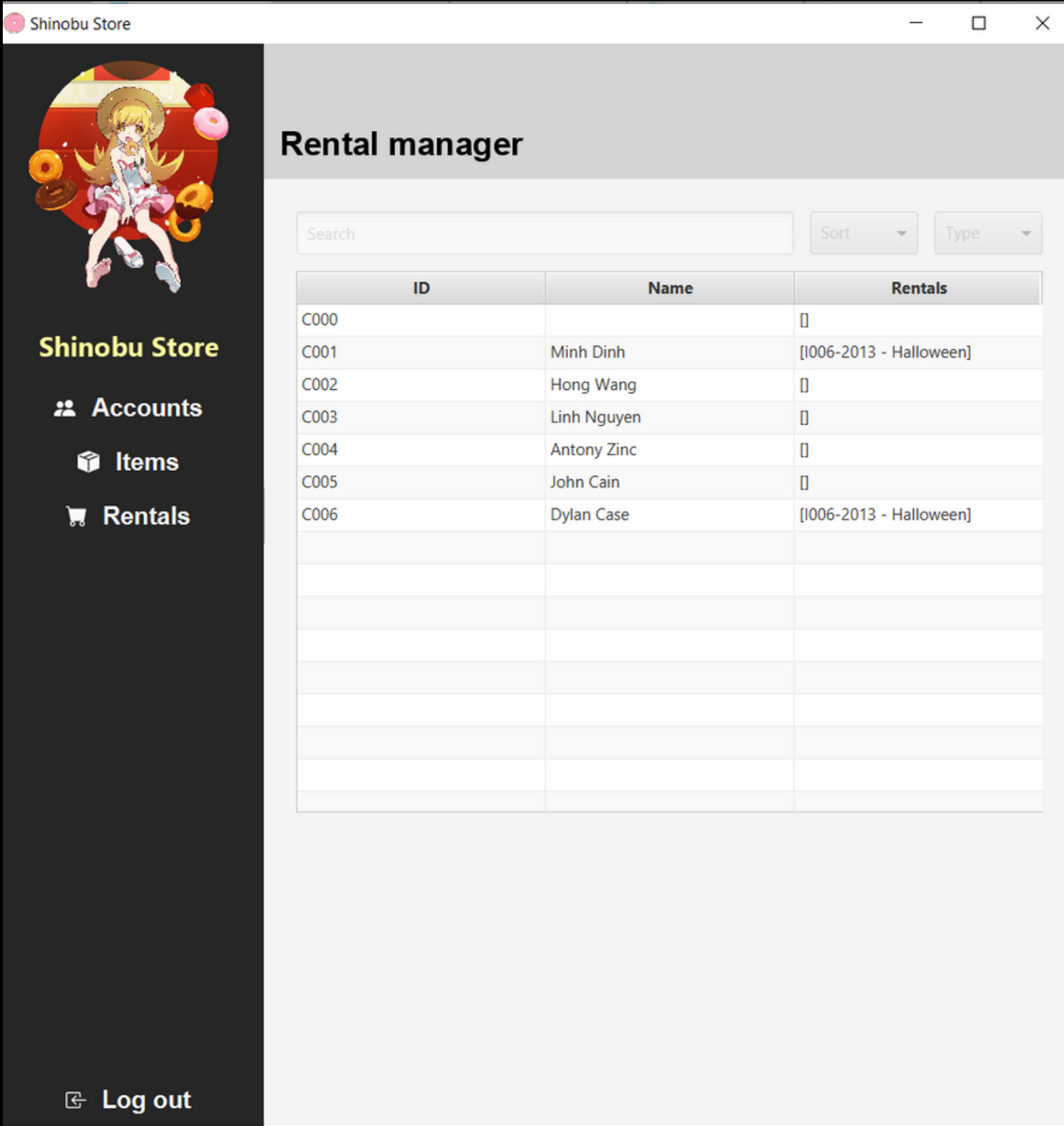
GUI - addItem.fxml

Features:

- Pop up when the 'Add' button is clicked, allows the admin to add more item along with their info into the database.

GUI - adminDashboard.fxml

Features:
- This page corresponds with the 'Rentals' button, displays all users that currently have any item rented.

THANK YOU