Backend Design Document

CSCI 3060U

Dr. Michael Miljanovic

Levi Willms, Daniel Hinbest, Syed Rizvi, Raje Singh

Run the Backend

System Requirements: Linux distribution running either natively or within WSL. For example, Ubuntu for WSL, Ubuntu virtual machine, or Ubuntu OS.

To run the backend system, assuming the frontend has previously run and produced output files, run `make run_backend`. Otherwise, if the frontend has NOT been run or there are changes to be compiled and run from the frontend, use `make run_system`.

`make run_backend`: Will compile all daily transaction files produced by the frontend in frontend storage with `create_tests_dtf.py` and assign each a new name in the format `daily_transaction_{date}.txt where the date is some time in the previous calendar day. **NOTE: This is only to be used for testing. Will not exist in production. The system will then compile all the daily transaction files into one merged daily transaction file.**

Finally, it reads the old user accounts, game collection, and available games files then parses the merged daily transaction file to make any changes. While parsing the daily transaction file it will store a local map of each of the other files and validate against them for any potential changes.

`make run_system`: Will clean all executable files in the frontend, clear all daily transaction files in storage, and clean all test outputs. Then it will run the backend system. See the `run_backend` description for further details.

Note: python3 must be installed on the host machine for compilation and running the system.

Updated files

The backend writes files to its own storage directory in `src/backend/storage`. This includes the backend's log file, merged daily transaction file and the new user accounts, game collection, and available games files. The old user accounts, available games, and game collection files are also stored in this directory. The system will be fully integrated with the new front end in the

implementation phase where the backend will automatically read and replace the files in the frontend storage.

System Procedure: Main
Description: Main is not a class but a system of methods. The purpose of the main procedure is to bring all components of the backend together to run the system.

Method	**Description**
main()	**Parameters:** none **Return type:** N/A **Inputs:** old_user_accounts.txt, old_available_games.txt, old_game_collection.txt, daily_transactions_*.txt. **Outputs:** merged_transactions_{date}.txt, users_map, games_map, game_collection_map, log.txt The main method that runs the system. It creates a file manager for the old files, merges the daily transaction files, and calls fileParser to write the new files.
fileParser(mergedDailyTransactionFile, log)	**Parameters:** string mergedDailyTransactionFile, string log **Return type:** N/A **Inputs:** merged_transactions_{date}.txt. **Outputs:** new_user_accounts.txt, new_available_games.txt, new_game_collection.txt, log.txt. A method that reads the merged daily transaction file sequentially and handles each transaction logic. Then writing to the new files.

Class: FileManager
Description: An abstract class for file management with functions to read or write from a file.

Method	**Description**
read()	**Parameters:** self, file_name **Return type:** N/A **Inputs:** N/A **Outputs:** N/A An abstract method that will serve for the base of reading data from a file and returning a list of the data objects.
write()	**Parameters:** list data **Return type:** N/A **Inputs:** N/A **Outputs:** N/A Writes a list of games to a new file. Does not return anything.

Class: AvailableGamesFileManager (Base: FileManager)
Description: Inherited from FileManager, AvailableGamesFileManager is a file manager that is designed for file management for the available games

Method	Description
read()	**Parameters:** self, string file_name **Return type:** List<Game> available_games **Input**: old_available_games.txt **Output**: N/A A read method that reads Game data from a file and returns Game data objects with the Game data read from the file
write()	**Parameters:** self, string file_name, List<Game> games **Return type:** N/A **Input**: N/A **Output**: new_available_games.txt Writes a list of games to the new available games file. Does not return anything.

Class: UserAccountsFileManager(Base: FileManager)
Description: Inherited from FileManager, UserAccountsFileManager is a file manager that is designed for file management for the user accounts.

Method	Description
read()	**Parameters:** self, string file_name **Return type:** List<User> user_accounts **Input**: old_user_accounts.txt **Output**: N/A A read method that reads User data from a file and returns User data objects with the User data read from the file.
write()	**Parameters:** self, string file_name, List<User> users **Return type:** N/A **Input**: N/A **Output**: new_user_accounts.txt Writes a list of users to the new user accounts file. Does not return anything.

Class: GameCollectionFileManager(Base: FileManager)
Description: Inherited from FileManager, GameCollectionFileManager is a file manager that is designed for file management for the game collections.

Method	Description
read()	**Parameters:** self, string file_name **Return type:** List<Collection> game_collections **Input**: old_game_collection.txt **Output**: N/A A read method that reads Collection data from a file and returns Collection data objects with the Collection data read from the file.

write()	**Parameters:** self, string file_name, List<User> users **Return type:** N/A **Input**: N/A **Output**: new_game_collection.txt Writes a list of game collections to the new game collection file. Does not return anything.

Class: DailyTransactionFileManager(Base: FileManager)
Description: Inherited from FileManager, DailyTransactionFileManager is a file manager that is designed for file management for the daily transactions.

Method	**Description**
read()	**Parameters:** self **Return type:** List<String> files **Input**: N/A **Output**: N/A A read method that finds the names of all daily transaction files from the previous calendar day and returns a list of the file names.
write()	**Parameters:** self **Return type:** string merged_filename **Input**: daily_transaction_{date}*.txt **Output**: merged_transactions_{date}.txt Reads all daily transactions passed to the method and writes them to a new, merged, daily transaction file for the whole calendar date.

Class: Collection
Description: This class is designed to represent a collection of games that would be available to a user.

Method	**Description**
__init__(self, game_name, owner)	**Parameters:** self, string game_name, string owner **Return type:** N/A **Input**: N/A **Output**: N/A A default constructor to create a new Collection object with default attributes
get_game_name(self)	**Parameters:** self **Return type:** string **Input**: N/A **Output**: N/A An accessor method to return the game name.
get_owner(self)	**Parameters:** none **Return type:** string

	Input: N/A **Output**: N/A An accessor method that returns the name of the game's owner
set_game_name(self, name_name)	**Parameters:** string game_name **Return type:** N/A **Input**: N/A **Output**: N/A A mutator method that assigns a value to the gameName attribute
set_owner(self, owner)	**Parameters:** string owner **Return type:** N/A **Input**: N/A **Output**: N/A A mutator method that assigns a value to the owner attribute
to_string(self)	**Parameters:** self **Return type:** string **Input**: N/A **Output**: N/A A method that converts the collection object to a human readable string.

Class: User
Description: This class is designed to represent a user account in the system.

Method	**Description**
__init__(self, username, user_type, available_credit)	**Parameters:** self, string username, string user_type, Float available_credit **Return type:** N/A **Input**: N/A **Output**: N/A A default constructor to create a new User object with default attributes
set_username(self, username)	**Parameters:** self, string username **Return type:** string username **Input**: N/A **Output**: N/A A method that sets the particular username for a user object.
set_user_type(self, string)	**Parameters:** self, string user_type **Return type:** N/A **Input**: N/A **Output**: N/A A mutator method that assigns a value to the user_type attribute.
add_credit(self, float)	**Parameters:** self, float added_credits **Return type:** N/A

	Input: N/A **Output**: N/A A mutator method that adds credits to a user.
remove_credit(self, float)	**Parameters:** self, float removed_credits **Return type:** N/A **Input**: N/A **Output**: N/A A mutator method that removes credits from a user.
get_username(self)	**Parameters:** self **Return type:** string username **Input**: N/A **Output**: N/A An accessor method to return the username.
get_user_type(self)	**Parameters:** self **Return type:** string **Input**: N/A **Output**: N/A An accessor method that returns the user type attribute.
get_credit(self)	**Parameters:** self **Return type:** float **Input**: N/A **Output**: N/A A method to return the credit amount of the user.
to_string(self)	**Parameters:** self **Return type:** string **Input**: N/A **Output**: N/A A method that converts the user object to a string.

Class: Game
Description: This class is designed to represent games that would be available to a user in the system.

Method	**Description**
__init__(self, name, seller, price)	**Parameters:** self, string name, string seller, float price **Return type:** N/A **Input**: N/A **Output**: N/A A default constructor to create a new Collection object with default attributes
set_name(self, name)	**Parameters:** self, string name **Return type:** N/A **Input**: N/A

	Output: N/A A mutator method to set the game name.
set_seller(self, seller)	**Parameters:** self, string seller **Return type:** N/A **Input**: N/A **Output**: N/A A mutator method that sets the owner of the game.
set_price(self, price)	**Parameters:** self, float price **Return type:** N/A **Input**: N/A **Output**: N/A A mutator method that assigns a value to the price attribute.
get_name(self)	**Parameters:** self **Return type:** string **Input**: N/A **Output**: N/A An accessor method that returns the name attribute.
get_seller(self)	**Parameters:** self **Return type:** string **Input**: N/A **Output**: N/A An accessor method that returns the seller attribute.
get_price(self)	**Parameters:** self **Return type:** float **Input**: N/A **Output**: N/A An accessor method that returns the price attribute.
to_string(self)	**Parameters:** string owner **Return type:** string **Input**: N/A **Output**: N/A A method that converts the game object to a string.

Class: Transaction
Description: Represents a transaction in the system. It handles different types of transactions based on the transaction code.

Method	**Description**
__init__(self, name, seller, price)	**Parameters:** self, dictionary users_map, dictionary games_map, dictionary game_collection_map, dictionary log_file **Return type:** N/A **Input**: N/A

	Output: N/A A default constructor to create a new Collection object with default attributes
handle_create(self, line)	**Parameters:** self, string line **Return type:** N/A **Input:** N/A **Output:** log.txt Handles the create transaction
handle_delete(self, line)	**Parameters:** self, string line **Return type:** N/A **Input:** N/A **Output:** log.txt Handles the delete transaction
handle_sell(self, line)	**Parameters:** self, string line **Return type:** N/A **Input:** N/A **Output:** log.txt Handles the sell transaction
handle_buy(self, line)	**Parameters:** self, string line **Return type:** N/A **Input:** N/A **Output:** log.txt Handles the buy transaction
handle_refund(self, line)	**Parameters:** self, string line **Return type:** N/A **Input:** N/A **Output:** log.txt Handles the refund refund transaction
handle_add_credit(self, line)	**Parameters:** self, string line **Return type:** N/A **Input:** N/A **Output:** log.txt Handles the add credit transaction
handle_transaction(self, line)	**Parameters:** self, string transaction_code, string line **Return type:** N/A **Input:** N/A **Output:** N/A Handles the transaction based on the transaction code