Auctionr: A CSCI3060 Application

Design Document

By:

Anthony Kouroupis
Alexander Minz
Omer Baskurt

The front end is split into 4 modules which will be detailed in this document.

Options

Options classes are for runtime settings such as file locations. The files for these classes are stored in the options folder of the src directory.

The EnvOptions implementation will read the values from environment variables. Constants for the variable names are declared in EnvOptions.h, and are as follows:

Name	Description
ENV_ACCOUNTS_FILE	Path to the Accounts file
ENV_ITEMS_FILE	Path to the Items file
ENV_LOG_FILE	Path to the daily log file

There is also a FileOptions class that will read the application's options from a file of key-value pairs wherein the keys and values are separated by a space, and the entries are separated by a new line. FileOptions does not support options which have a space in either the key or value.

The following table details the functions and purpose of the functions in the classes of the Options module. Every function in Options is implemented in every implementation of Options, to the same ends.

Class Methods

Class	Function	Decription
Options	AccountsFile()	Returns a path to the User Accounts file
Options	ItemsFile()	Returns a path to the Items file
Options	DailyLogFile()	Returns a path to the daily log file
FileOptions	LoadOptions(string file)	Loads the values for AccountsFile(), ItemsFile() and DailyLogFile() from a file

Model Storage

Handles the creating, reading, updating, and deleting of models in long-term storage. These classes are stored in the storage folder in the src directory of the front end. The abstract classes for the objects in the system are:

- UserStorage
- ItemStorage
- DailyLog

These classes exist to abstract the method of reading and writing these objects to and from long term storage from the rest of the code. Each class reads and writes the objects corresponding to their name. Commands only receive pointers to these objects in their abstract form. The existing implementations for these classes are:

- UserFileStorage: extends UserStorage

- ItemFileStorage: extends ItemStorage
- DailyLogFile: extends DailyLog

These implementations read/write to the files they are configured to in the format specified by the system requirements. They implement no functions beyond that which their corresponding abstract class define and fulfill the same purpose.

There is also two exceptions that can be thrown from the UserStorage and ItemStorage classes

- UserStorageException
- ItemStorageException

These exceptions are generalized errors for exceptions when reading from or writing to storage. The type of error can be told by the exception's error number which corresponds to a value in the ExceptionTypes struct defined in Exceptions.h

Class Methods

Class	Function	Description
DailyLog	void AddEntry(string entry)	Buffers the string entry to be written to the daily log file when the program exits
DailyLog	Void WriteChanges()	Called at the end of program execution. Writes all buffered lines to the daily log file in the order they were added in.
UserStorage	User* GetByName(string name)	Returns the user which has a username matching the name parameter. Throws a UserStorageException when there is no user found with that name.
ItemStorage	Item* GetByName(string name)	Returns the item listing which the name of the item being sold matches the name parameter. Throws an ItemStorageException when no item is found with that name.

Commands

The Command components are split into abstract and implementation components. The Command components are stored in the commands folder in the src directory of the front end. The main module will receive Commands from the Command module through the CommandFactory class. Each command handles a single transaction code from the requirements, except the NoCommand object which is returned when there is no matching command for the last user input.

Class Methods

Class	Function	Description
Command	GetInputs()	Prompt the user for input, and validate the input
Command	Execute()	Do any data or application state modification operations
Command	GetLogContents()	Return the log entry for this execution
Command	IsPermitted(User* user)	Returns true if the user's permissions are sufficient to run this command, false otherwise
Command	IsLoginRequired()	Returns true if this command requires that the user be logged in to run, false otherwise.

Command	PadString(string s, int size)	Utility function. Returns a string with the value of s, with trailing spaces added on the end until the length of the string is equal to size
Command	FlushStream(iostream stream)	Utility function. Resets the state variables of the given stream, and flushes the buffer for the stream.
Command	PromptUntilNumeric(string prompt)	Utility function. Will take input from the user, and if that input is not an integer, it will re-print *prompt* to the console and read input again.
NoCommand	IsLoginRequired()	Always returns false, this command doesn't do anything, and doesn't require that the user be logged in
MissingPermissions	Execute()	Outputs an error message telling the user that it doesn't have permission to execute the requested command
NotFound	IsLoginRequired()	Always returns false, this command doesn't require that the user be logged in as it runs when the user types an invalid command name
NotFound	NotFound(string commandName)	Initializes the name of the command to be output in Execute()
NotFound	Execute()	Outputs an error message specifying that the requested command doesn't exist
NotLoggedIn	Execute()	Outputs an error message that the user needs to be logged in to run that command
NotLoggedIn	IsLoginRequired	Returns false for obvious reasons
AddCreditCommand	AddCreditCommand(UserStorage*)	Initializes the command to search the given UserStorage
AddCreditCommand	GetInputs()	Gets the user name and amount of credit to add to the user, validates that the user exists and the amount of credit is non negative.
AddCreditCommand	GetLogContents()	Returns the daily log file entry for this action
Advertise	GetInputs()	Gets the item name, amount of days to auction the item for, and minimum bid amount from the user
Advertise	GetLogContents()	Returns the daily log file entry for this transaction
Advertise	IsPermitted(User* user)	Returns true if the user is any role except BUY_STANDARD
Bid	Bid(UserStorage*, ItemStorage*)	Initializes the userStorage and itemStorage properties that will be used to search for users and items
Bid	GetInputs()	Gets the item name, seller's name, and amount to bid from the user
Bid	GetLogContents()	Returns the log contents for the Bid command
Bid	IsPermitted(User*)	Returns true if the user's type is anything except SELL_STANDARD
CreateCommand	CreateCommand(UserStorage*)	Initializes the class to search the given UserStorage
CreateCommand	GetInputs()	Gets the user name, type, and initial credit for the user to create
CreateCommand	GetLogContents()	Returns the log file entry for the Create command
DeleteCommand	DeleteCommand(UserStorage*)	Initializes the command to search the provided UserStorage
DeleteCommand	GetInputs()	Gets the name and type of user to delete

DeleteCommand	GetLogContents()	Returns the Daily Log File contents for the Delete command
Login	Login(ApplicationState*, UserStorage*)	Initializes the login command with the given application state, and initializes the user storage to search for the user
Login	GetInputs()	Gets a username from the user, validates that the user exists
Login	Execute()	Updates the ApplicationState to be logged in by the user matching the username that was input in GetInputs()
Login	IsLoginRequired()	Always returns false for obvious reasons
Logout	Logout(ApplicationState*, DailyLog*)	Initializes the application state to use, and the daily log file to write to.
Logout	Execute()	Writes the contents of the daily log file provided, and
Refund	Refund(UserStorage*)	Initializes the userStorage to search for the users' names in
Refund	GetInputs()	Gets the buyer's name, seller's name, and amount to refund
Refund	GetLogContents()	Gets the daily log file entry for this command.
CommandFactory	CommandFactory(Options options)	Constructor for CommandFactory. Initializes commands using settings from Options in Create calls.
CommandFactory	Create(string input)	Returns the Command implementation corresponding to the input string. Returns an instance of NoCommand if there is no matching command for the input.

The Execute function can be empty for commands that only take and parse inputs into the format for the daily log file such as create and advertise. For functions such as login and logout which effect the application's state will do so in this function.

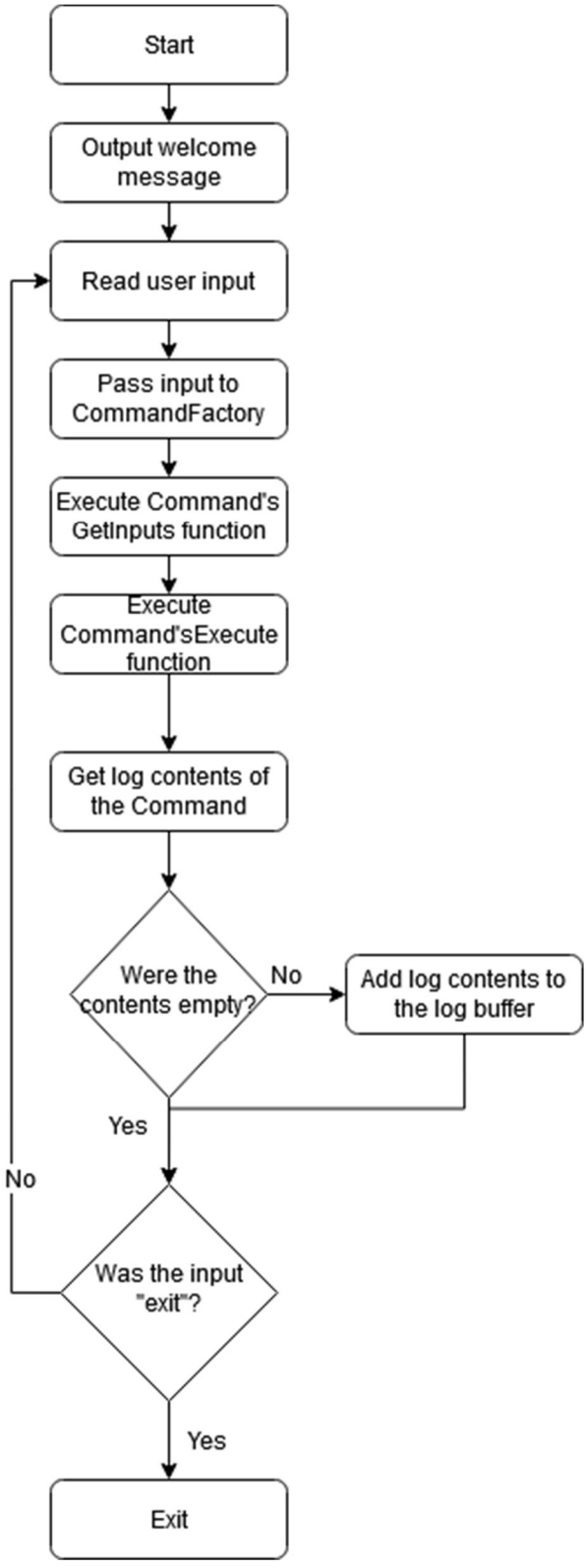
Main

The Main module is sourced directly in the src directory, and includes main.cpp and the ApplicationState class. This module runs the main loop, which calls the abstract classes to perform the application functionality. The flowchart for the main module can be found further below.

ApplicationState Methods

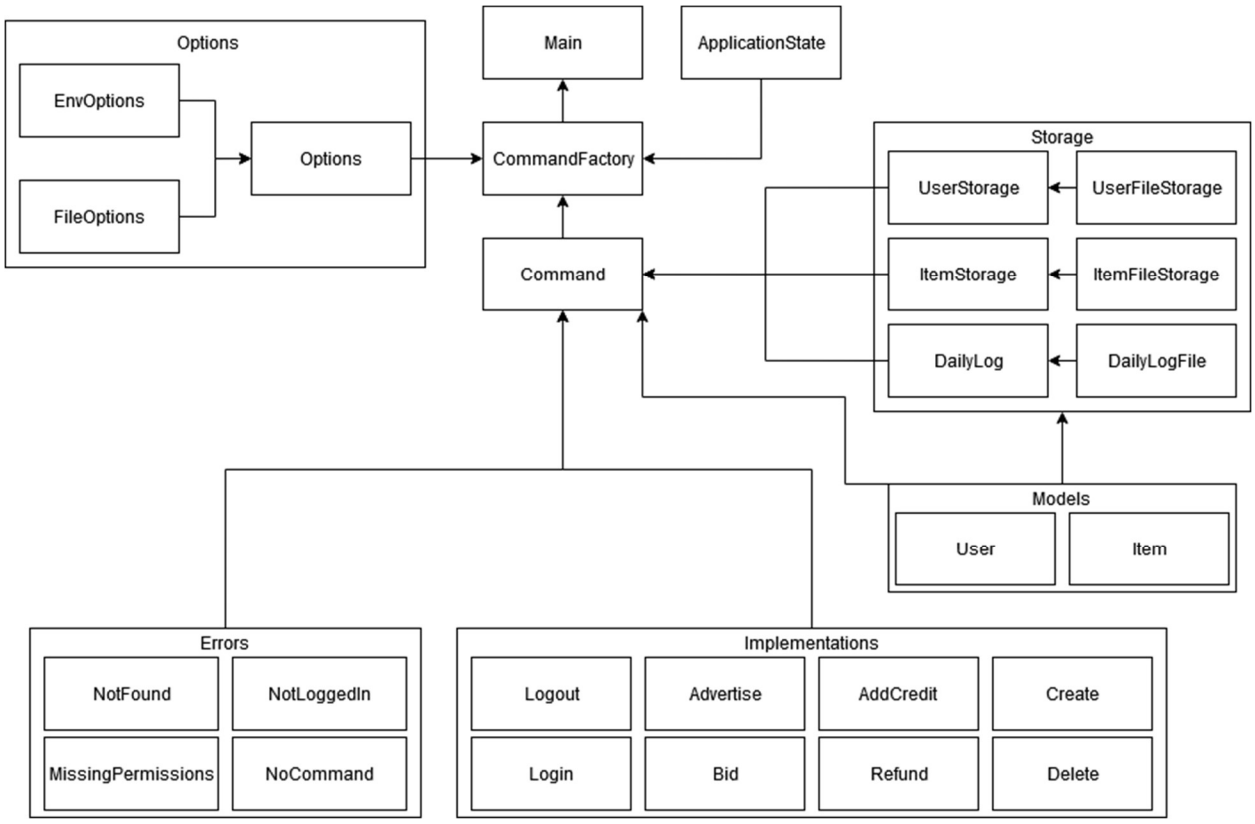
Class	Function	Description
ApplicationState	SetLoggedInUser(User*)	Sets the logged in user, IsLoggedIn will begin returning true until SetLoggedInUser(NULL) is called
ApplicationState	IsLoggedIn()	Returns true if a user has been set with SetLoggedInUser, false otherwise
ApplicationState	LoggedInUser()	Returns the currently logged in user

The behaviors of the application are abstracted from the main module through the Command objects. The main module dictates the order commands enact their functions in, and will write to the daily log file, as most commands don't directly influence the daily log file.



Module Interactions

Here is a diagram to visualize the relationships between the modules:



At the beginning, the Main module boostraps the options and application state dependencies into each other to ensure that the CommandFactory receives the initialization and state information it needs to return the proper commands.

Main will then pass its input into the CommandFactory to get the proper commands matching the user's input. One of the Error commands may be returned if the user has input a command they can't run, input it without being logged in, input a command that doesn't exist, or they have input no command at all. These commands differentiate the type of error in the users' input and give an appropriate error message. The CommandFactory will never return NULL, as use NULL should be avoided when possible.

The storage objects are passed to the Command objects on initialization, depending on the objects that the command needs to access. This is how objects will be read from long-term storage in the commands for validation and application state initialization.