An Extreme Course Project

CSCI 3060U – Winter 2024

Project Teams

You are to form a team of three or four people from the same lab section to design, implement, document, and deliver a two-part software product. All phases to follow the Extreme Programming philosophy as much as it applies – in particular,

- continuously maintained test suites as requirements and quality control
- pair programming of all code
- simplest possible solution to every problem
- continuous redesign and re-architecting
- automation in testing and integration
- frequent integration and complete releases

Every two weeks (or so, see the schedule below) you will deliver concrete evidence of your team's progress as required by project assignments.

Project Phases

The project will be done in six phases, each of which will be an assignment. Phases will cover steps in the process of creating a quality software result in the context of an Extreme Programming process model.

Assignments will be on the quality control aspects of requirements, rapid prototyping, design, coding, integration and analysis of the product you are building. Throughout the project, you should keep records of all evidence of your product quality control steps and evolution, to make the marketing case that you have a quality result at the end of the course.

Do not implement any features or additions to your project that are not explicitly listed by the requirements, your client (TA), or the phase assignment documents. Also, do not 'work ahead' on the project by attempting to complete future phases in advance, such as creating parts of the Back End before the Front End deadline. You may lose marks if you are working far ahead of schedule, especially for phases that are intended for you to test your code!!

Your final products will be tested and evaluated.

Project Schedule

The course project consists of six assignments, with separate handouts for each one. Assignments are scheduled to be due as follows:

- Phase #1: Front End Requirements

  Date Due: Friday, February 2, 2024

- Phase #2: Front End Rapid Prototype

  Date Due: Friday, February 16, 2024

- Phase #3: Front End Requirements Testing

  Date Due: Friday, March 1, 2024

- Phase #4: Back End Rapid Prototype

  Date Due: Friday, March 15, 2024

- Phase #5: Back End Unit Testing

  Date Due: Thursday, March 28, 2024

- Phase #6: Integration and Delivery

  Date Due: Wednesday, April 10, 2024

Assignment/Phase Hand-Ins – GitHub & Canvas

While working on a given assignment/phase of the project you are expected to maintain a GitHub project and regularly commit updates to source code, tests and all project documents. Your GitHub project must be non-public and should include your teaching assistant as a team member.

Unless otherwise specified, all assignments/phases *must* be handed in through Canvas by **8:00pm** on the due date. For all submissions indicate clearly your team name and all member names and on every hand in.

Late assignments/phases will be accepted through Canvas during the first 48 hours after the submission deadline with a cumulative 1% penalty per hour late. After 48 hours, late assignments/phases will not be accepted. Exceptions to this policy may be made for valid reasons (e.g., medical).

As part of the assessment of each assignment/phase, your teaching assistant will review your submitted Canvas materials as well as review your GitHub project activity. We will issue grade adjustments if it is clear, based on the Github project, that group work has not been evenly distributed (for example, if only one group member has contributed to the repo). We will compare the repo to a peer feedback form each member must fill out at the end of each phase, and adjust marks accordingly.

Project Requirements

The product you are to design and build is a Digital Distribution Service for Games (similar to Steam). The system consists of two parts:

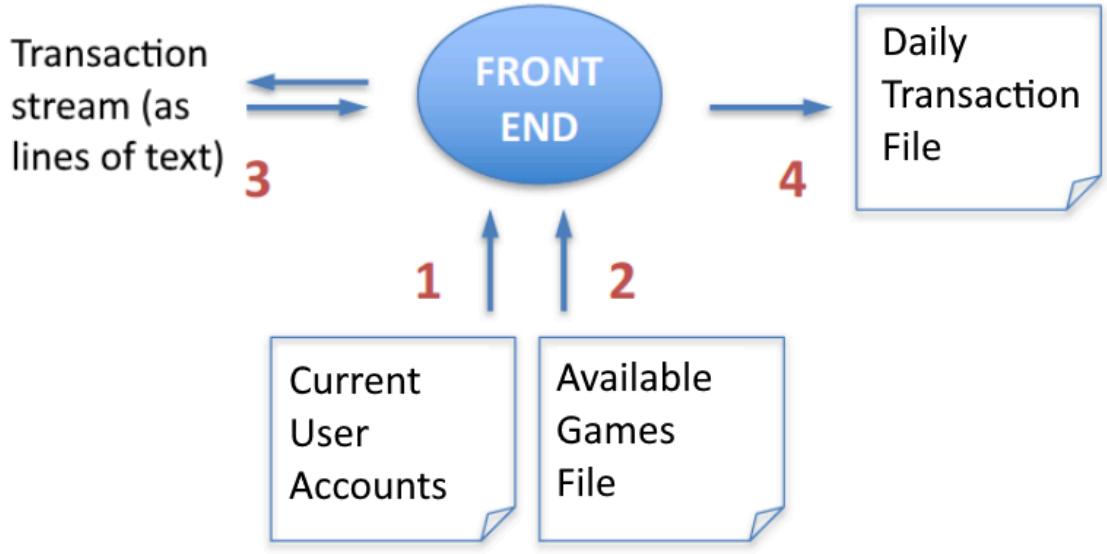
- the Front End, a point of purchase terminal for buying and selling games (written in C++)
- the Back End, an overnight batch processor to maintain and update a master games file intended to run each day at 12:00am (written in your choice of Python or Java)

Both parts will be run as console applications, that is, they are to be invoked from a command line and use text and text file input/output only (this is an important requirement for assignments later in the project, so don't ask for exceptions).

In true XP fashion, you must be prepared to adapt to changes in these requirements. Throughout this course, your client (represented by the Project Client account on Slack, and your TA in the labs) will issue periodic changes to the project. You should design your code in such a way that you can easily modify your program in the event of any sudden change.

THE FRONT END

The Front End reads in a file containing information regarding current user accounts in the system (1) and a file of games available for purchase (2), it processes a stream of game purchasing and advertising transactions one at a time (3), and it writes out a file of game purchasing and advertising transactions at the end of the session (4).



Informal Customer Requirements for the Front End

The Front End handles a sequence of transactions, each of which begins with a single transaction code (word of text) on a separate line. The Front End must handle the following transaction codes:

login	- start a Front End session
logout	- end a Front End session
create	- add a user with the ability to buy/sell games (privileged transaction)
delete	- remove a user (privileged transaction)
sell	- put a game up for sale
buy	- purchase a game available for sale
refund	- issue a credit to a buyer's account from a seller's account (privileged transaction)
addcredit	- add credit into the system for the purchase of accounts

Transaction Code Details

login - start a Front End session

- before processing a login transaction, the Front End reads the current user accounts file.
- should ask for the username
- after the username is accepted, reads in the available games file (see below) and begins accepting transactions
- Constraints:
    - no transaction other than login should be accepted before a login
    - no subsequent login should be accepted after a login, until after a logout
    - after a non-admin login, only unprivileged transactions are accepted
    - after an admin login, all transactions are accepted

logout - end a Front End session

- should write out the daily transaction file (see below) and stop accepting any transactions except login
- Constraints:
    - should only be accepted when logged in
    - no transaction other than login should be accepted after a logout

create – creates a new user with buying and/or advertising privileges.

- should ask for the new username (as a text line)
- then should ask for the type of user (admin or full-standard, buy-standard, sell-standard)
- should save this information to the daily transaction file
- Constraints:
    - privileged transaction - only accepted when logged in as admin user
    - new user name is limited to at most 15 characters
    - new user names must be different from all other current users
    - maximum credit can be 999,999

delete - cancel any games for sale and remove the user account.

- should ask for the username (as a text line)
- should save this information for the daily transaction file
- Constraints:
    - privileged transaction - only accepted when logged in as admin user
    - username must be the name of an existing user but not the name of the current user
    - no further transactions should be accepted on a deleted user's available inventory of games.

sell – put up a game for sale

- should ask for the game name (as a text line)
- should ask for a price for the game in dollars (e.g. 15.00)
- should save this information to the daily transaction file
- Constraints:
    - Semi-privileged transaction - only accepted when logged in any type of account except standard-buy.
    - the maximum price for a game is 999.99
    - the maximum length of a game name is 25 characters
    - All games must have unique names
    - no further transactions should be accepted on a new game for sale until the next session.

buy – purchase a game available for sale

- should ask for the game name and the seller's username
- should add the new game to the user's collection
- should credit the seller with an amount equal to the game's price and deduct that amount from the buyer
- should save this information to the daily transaction file
- Constraints:
    - Semi-privileged transaction - only accepted when logged in any type of account except standard-sell.
    - game name must be an existing game
    - the buyer must have enough money to buy the game
    - the buyer must not already have a copy of the game in their collection

refund - issue a credit to a buyer's account from a seller's account (privileged transaction)

- should ask for the buyer's username, the seller's username and the amount of credit to transfer.
- should transfer the specified amount of credit from the seller's credit balance to the buyer's credit balance.
- should save this information for the daily transaction file
- Constraints:
    - Buyer and seller both must be current users

add credit - add credit into the system for the purchase of accounts

- In admin mode, should ask for the amount of credit to add and the username of the account to which the credit is being added.
- In a standard account, should ask for the amount of credit.
- should save this information to the daily transaction file
- Constraints:
    - In admin mode, the username has to be an existing username in the system.
    - A maximum of $1000.00 can be added to an account in a given session.

General Requirements for the Front End

The Front End should never crash or stop except as directed by transactions.

The Front End cannot depend on valid input - it must gracefully and politely handle bad input of all kinds (note: but you can assume that input is at least lines of text).

Daily Transaction File

At the end of each session, when the logout transaction is processed, a daily transaction file for the day is written, listing every transaction made in the session.

Contains variable-length text lines of the form:

```
XX_UUUUUUUUUUUUUUU_TT_CCCCCCCCC
```

Where:

XX
: is a two-digit transaction code: 01-create, 02-delete, 06-add credit, 00-end of session

UUUUUUUUUUUUUUU
: is the username (buyer if two users in the transaction)

TT
: is the user type (AA=admin, FS=full-standard, BS=buy-standard, SS=sell-standard)

CCCCCCCCC
: is the available credit

_
: is a space

```
XX_UUUUUUUUUUUUUUU_SSSSSSSSSSSSSSS_CCCCCCCCC
```

Where:

XX
: is a two-digit transaction code: 05-refund

UUUUUUUUUUUUUUU
: is the buyer's username

SSSSSSSSSSSSSSS

is the seller's username

CCCCCCCCC

is the refund credit

_

is a space

```
XX_IIIIIIIIIIIIIIIII_SSSSSSSSSSSS_PPPPPP
```

where:

XX

is a two-digit transaction code: 03-sell.

IIIIIIIIIIIIIIIII

is the game name

SSSSSSSSSSSSS

is the seller's username

PPPPPP

is the price

_

is a space

```
XX_IIIIIIIIIIIIIIIII_SSSSSSSSSSSSS_UUUUUUUUUUUUU_PPPPPP
```

where:

XX

is a two-digit transaction code: 04-buy.

IIIIIIIIIIIIIIIII

is the game name

SSSSSSSSSSSSS

is the seller's username

UUUUUUUUUUUUU

is the buyer's username

PPPPPP

is the game's price

_

is a space

Constraints:

- alphabetic fields are left justified, filled with spaces
  (e.g. Jane_Doe_____ for account holder Jane Doe)
- unused numeric fields are filled with zeros
  (e.g., 0000)
- In a numeric field that is used to represent a monetary value, ".00" is appended to the end of the value
  (e.g. 00110.00 for 110)
- unused alphabetic fields are filled with spaces (blanks)
  (e.g., _____)
- the sequence of transactions ends with an end of session (00) transaction code

Current User Accounts File

Consists of fixed length (28 characters) text lines in the form:

```
UUUUUUUUUUUUUUU_TT_CCCCCCCCC
```

where:

`UUUUUUUUUUUUUUU`
: is the username

`TT`
: is the user type (AA=admin, FS=full-standard, BS=buy-standard, SS=sell-standard)

`CCCCCCCCC`
: is the available credit

`_`
: is a space

Constraints:

- every line is exactly 28 characters (plus newline)
- alphabetic fields are left justified, filled with spaces
  (e.g., User001_____ for username "User001")
- unused numeric fields are filled with zeros
  (e.g., 0000)

- in a numeric field that is used to represent a monetary value, if the value is only in dollars, then ".00" is appended to the end of the value (e.g. 110.00 for 110)
- unused alphabetic fields are filled with spaces (blanks) (e.g., _____)
- file ends with a special user named END with an empty user type and the credit field empty.

Available Games File

Consists of fixed length (49 characters) text lines in the form:

```
IIIIIIIIIIIIIIIIIIIIIII_SSSSSSSSSSSSSS_PPPPPP
```

where:

IIIIIIIIIIIIIIIII
: is the game name

SSSSSSSSSSSSS
: is the seller's username

PPPPPP
: is the price of the game.

_
: is a space

Constraints:

- every line is exactly 42 characters (plus newline)
- alphabetic fields are left justified, filled with spaces (e.g., Stardew_Valley_____ for game "Stardew Valley")
- unused numeric fields are filled with zeros (e.g., 0000)
- in a numeric field that is used to represent a monetary value, if the value is only in dollars, then ".00" is appended to the end of the value (e.g. 110.00 for 110)
- unused alphabetic fields are filled with spaces (blanks) (e.g., _____)
- file ends with a special game for sale named END with all other fields empty.

Game Collection File

Consists of fixed length (42 characters) text lines in the form:

```
IIIIIIIIIIIIIIIIIIIIIIII_SSSSSSSSSSSSSS
```

where:

IIIIIIIIIIIIIIIIII
: is the game name

SSSSSSSSSSSSSS
: is the owner's username

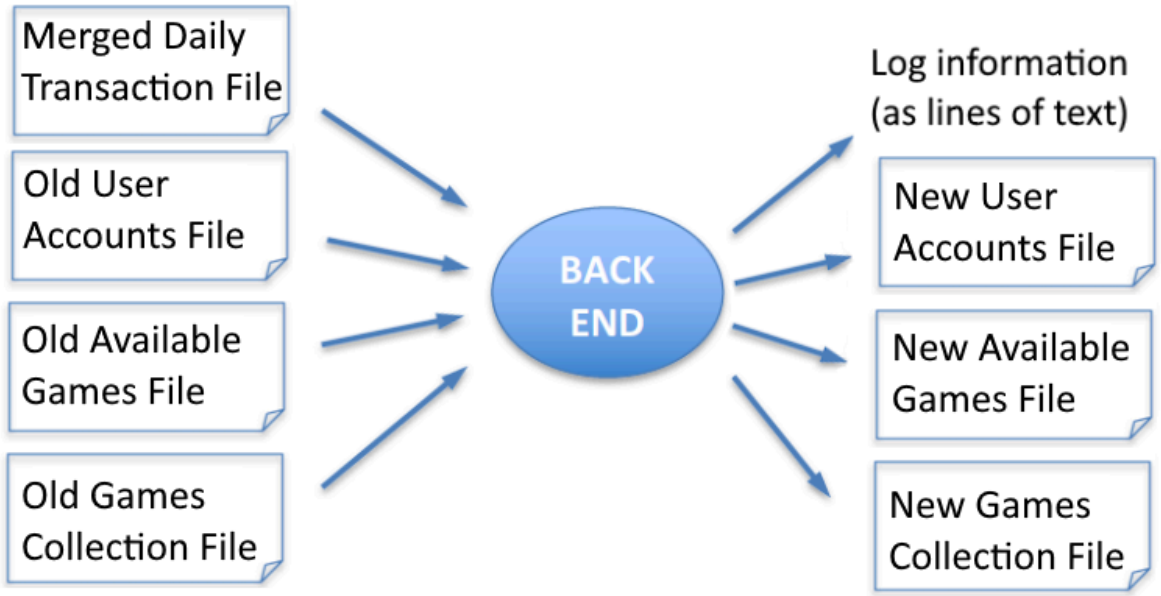
_
: is a space

Constraints:

- every line is exactly 42 characters (plus newline)
- alphabetic fields are left justified, filled with spaces (e.g., Stardew_Valley_____ for game "Stardew Valley")
- unused numeric fields are filled with zeros (e.g., 0000)
- file ends with a special game named END with all other fields empty.

THE BACK END

The Back End reads in the previous day's User Accounts File, Available Games File, and Games Collection File and then applies all of the daily transactions from a merged set of daily transaction files to these files to produce a new Current User Accounts File, Available Games File, and Games Collection File for tomorrow's Front End runs.



Informal Customer Requirements for the Back End

The Back End reads the Merged Daily Transaction File (see below) and applies all transactions to the Old Current User Accounts File, Old Available Games File, and Old Games Collection File to produce the New Current User Accounts File, the New Available Games File, and the New Games Collection File.

The Back End enforces the following business constraints, and produces a failed constraint log on the terminal as it processes transactions from the merged daily transaction file.

Constraints:

- no user should ever have a negative account balance
- a newly created user must have a name different from all existing users

General Requirements for the Back End

The Back End should assume it will have the correct input format on all files, and need not check for bad input (e.g. a corrupt file). However, if by chance it notices bad input, it should immediately stop and log a fatal error on the terminal.

Back End Error Recording

All recorded errors should be of the form: `ERROR: <msg>`

- For failed constraint errors, `<msg>` should contain the type and description of the error and the transaction that caused it to occur.
- For fatal errors, `<msg>` should contain the type and description and the file that caused the error.

The Merged Daily Transaction File

The Merged Daily Transaction File is the concatenation of any number of Daily transaction files output from Front Ends, ending with an empty one (one containing no real transactions, just a line with a 00 transaction code).