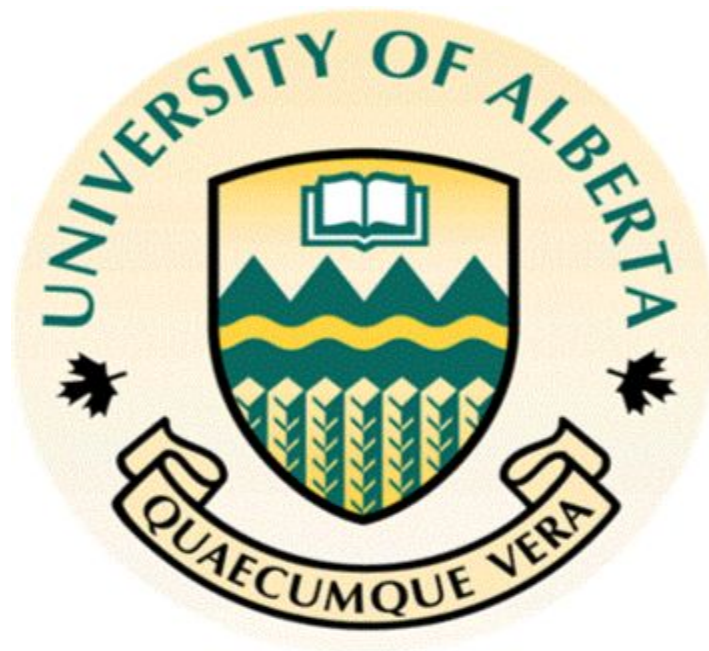


CMPUT 291 (Lab EH02) - File and Database Management
Department of Electrical and Computer Engineering
University of Alberta



Mini Project #1 (Group Project)

Authors: Jarrod Los & Cameron Hildebrandt

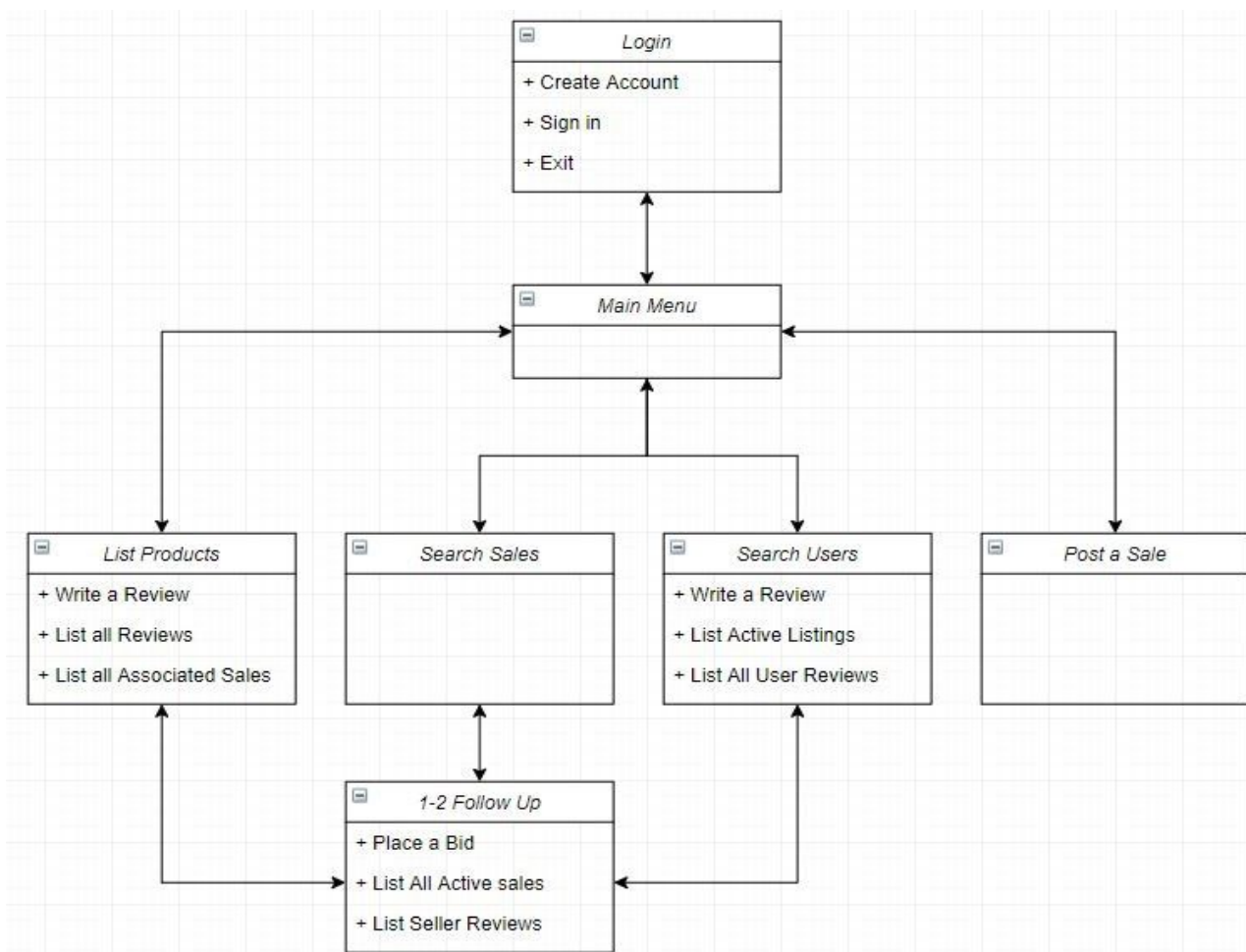
Submission Date: March 11th, 2020 (5pm)

Report:

[12 marks total] Report includes

- (a) A general overview of the system with a small user guide
- (b) A detailed design of the software
- (c) Testing strategy
- (d) Group work strategy (see the next bullet)

GENERAL OVERVIEW: The program designed consists of three main sections, first the login section where a user may create a new account or sign in with a previously created one. Once signed in the user may select between 1 of 4 options; **List products**, **Search for sales**, **Post a sale**, and **Search for users**. The last main section is the 1-2 follow up where a user may investigate a product, sale or user further to pull up a more detailed overview which offers actions such as writing a review, creating a bid, seeing similar products and so on. Note that you can traverse back and forth between any state, exit or logout.



DETAILED DESIGN: Brief summary of each methods functionality and relation.

`def restartProgram():` Used to sign out at any time and is called by the customIn method

`def getPath():` Gets the path to the database that was passed in from the terminal

`def connect(path):` Points at the data base passed as an argument into the terminal

`def init():` Initialize a beginning message and some variables

`def customIn(prompt = ""):` Over writes the input method to listen for a .logout, .quit

`def VerifyNew(email):` Checks if an email exists for new accounts

`def VerifyExisting(email, password):` Checks the email and password (Case sensitive)

`def CheckAccount():` Calls VerifyExisting based on case of signing in

`def CreateAccount():` Calls VerifyNew based on case of new account or signing in

`def checkSignInCmd():` Calls CheckAccount and CreateAccount to handle logic on login

`def viewReviews(user):` Calls a query using the user to pull all reviews

`def placeBid(sid, selectedSale, resPrice, maxBid):` Creates a bid with the provided info

`def showActiveListingsListProduct(product):` Shows all active listings of the product

`def showActiveListings(user):` Shows all active listings posted by the user

`def createProdReview(rating, text, pid):` Creates a review for the provided product

`def createSellerReview(rating, text, email):` Creates a review for the provided seller

`def createSale(edate, descr, cond, rprice, pid):` Creates a sale from the provided input

`def listReviews(pid):` Lists all reviews associated with the product id number

`def listSales(pid):` Lists all sales associated with the product id number

`def getCurrentDate():` Gets the current date, references the createReview Method

`def listProducts():` Lists all available products that have an active sale

`def searchSale():` Lists all active sales (Uses 1-2 follow up)

`def searchUser():` Lists all users (Uses 1-2 follow up)

`def mainMenu():` Used to call all the methods together in a concise fashion

TESTING STRATEGY:

Broke down each task into a series of edge cases, and took into account the data restrictions from the attributes of the tables. This involves testing a lot of redundant inputs with slight changes to check data ranges, size restrictions, data types, incorrect characters as well as the actual data flow and navigation. We roughly have about 10 bugs give or take as it stands with certain edge cases or data types that can cause an error, or the flow of the app can be interrupted by trying to exit, logout or go back causing unexpected errors. With enough testing and time these would of been fixable, but did not affect the average users input and you would have to be aware of the edge case and go out of your way to break the code, so we believe it is at a tolerable level given the amount of states that the program has (Roughly 22 different operating states the program can be in).

GROUP WORK STRATEGY: Broke down by order provided on eclass (All completed)

The project breakdown follows that similarly to eclass, where we both tackled the login screen where Jarrod handled the interface to SQL and the database, and Cameron set up the interface and UI for the python. Pushing into the project, the the main tasks (1, 2, 3, 4, 5) were split by there similarity in queries. This was done to prevent redundant solutions. Overall about 16 hours each were spent on the project bring it to a grand total of 32 man hours to complete!

- Login: Split between Jarrod and Cameron (Approx. 3 hours each)
- Main Menu: Cameron (Approx. 1 hour)
- Task #1 (Main Query): Jarrod (2 hour)
- Task #1A: Jarrod (2 hour)
- Task #1B: Jarrod (2 hour)
- Task #1C: Cameron (½ hour)
- Task #2: Cameron (4 hours)
- Task #3: Cameron (3 hours)
- Task #4: Jarrod (2 hours)
- Task #5 (Main Query): Cameron (3 hours)
- Task #5A: Jarrod (2 hour)
- Task #5B: Cameron (2 hours)
- Task #5C: Jarrod and Cameron (1 hour)
- Lab Report: Jarrod and Cameron (2 Hour)

We used github as well as a google drive to keep reference of progress, and handle merging problems. The tasks were split up by their similarities to each other such that previous work could be efficiently reapplied where possible instead of re-deriving the same solution!