Onur Ucar

Professor Andy Harris

CSCI 24000

28 April 2020

**CSCI 24000 Final Project**

**Gaming Cart**

* **What is Gaming Cart?**

Gaming Cart is a checkout system for businesses which sell video games. It provides a software to video game shops to let them maintain various games in the system, displaying games according to the platform and category requested by customer, helps management to keep track of transactions, lets users -which are the people who buy video games from the store using Gaming Cart- request refund when needed, prints out the transactions every time after a purchase, prints out the previous orders of customers and much more..

* **Who is/are the target group?**

Target group is mainly businesses which sell video games, but it’s not limited to video games. The idea can be really easily implemented to other business fields, just by designing a new database and a new relationship between functions and tables of the database, which helps us to manipulate and store the we want.

**Why I decided to work on this particular project what could it be better?**

I’m really into data, databases and manipulating them. My CSCI23000 project was an employee management system, and I built that system using Python and SQLite. I was really enjoyed working on my previous project and it gave me the idea of working on a database one more time, using a new and fun language.

I was actually considering building a life simulation game, but I thought it would be better if I work on a database system, as I want to work on in my professional career. After a couple days full of research, I started to think on a project which involves database. I definitely can say that, thinking on a database design is one of the hardest, and it MUST be the first thing we should start with, just like the algorithm files of our regular programming assignments. The reason why is, great algorithm lies on a great database design.

* **What have I learned, what could it be better?**

Database projects require to think more wisely and deeply. A forgotten point may cost you hours and hours to fix the code you written, which actually works really well without the forgotten point. You start fixing with adding a column to your database, then try to change your both query and JAVA codes, then you realize the files you’ve been fixed are not the only place which your incomplete table involves… Consequently, starting a project without a picture of complete database is a terrible idea. My own forgetting experience was a bit tragic, I wanted to add a shopping cart system to my program, and I thought it was working really well until I started to test my program using different accounts. Unfortunately, it was not working like how I wish it would. Then I added just one column to “Shopping\_Cart” column in the table, which added more than six hours to my development process. Hopefully, it works well right now. These were my experiences and thoughts about a programming project which involves a database and its design. But what have I learned technically?

First thing I realized one more time is Java is so powerful and there are many ways to accomplish a single job! For instance, we can create a table using an external jar file, we can create a multidimensional array list putting the data in it, then we can access the data using loops and counting variables and much more... This project also helped me to visualize and create connections between classes better. I also remembered how important it is to split to problem into smaller pieces (LoggedIn.java is an example of splitting the problem into smaller pieces and then solving it.) Having a two connection to a database at the same time is not a good programming practice, and never should be done. It locks out the database. Alternatively, we can access to the database using another JDBC connection afterwards, or by creating a new class and connecting the database separately. Making sure that we closed the connections to the database is also another key point. I will work on my project a little bit more, and either add a hashing system to passwords or use a .config file to keep usernames and password secure.

One of my other plans was printing out PDF files as receipts, but I used regular .txt extension instead. The reason why is, “.txt is” much more user friendly in this situation. Text files take up less space and easier to access. I was also planning to use Apache POI library to use Excel files, but I ran out of time, unfortunately.

**How I organized my program?**

After I started to work on this project, I drew a diagram which shows me the connections between frames (tabs) that I’ll be using in the GUI side. I added new components to my diagram after I built a new class/frame. Please see the attached image in the algorithm section below.

* **Database Design**
  + Games

This table contains video games, and it’s mainly used in Games.java. We’re able to add, edit and delete video games in Games.java. Since G\_Platform and G\_Category variables are reading and writing into combo boxes, it’s not possible to have unknown categories or platforms in the store.

* + - GameID : Primary key
    - G\_Name : Game name
    - G\_Platform : Game platform (connected to the Platforms table)
    - G\_Category : Game category (connected to the Category table)
    - Metascore : Metacritic store out of 100
    - Price : Price in USD
    - Stock : Shows that how many left in the stock
    - Image
  + Categories
    - CategoryID : ID (PK)
    - CName : Category Name
  + Platforms
    - PlatformID : ID (PK)
    - Platform\_Name : Platform name
    - Product\_Code
    - Processor
    - CPU
    - GPU
    - Memory
    - Consumption
  + Orders
    - OrderID : PK
    - UserID : The user who bought the item
    - TransactionNumber : Unique receipt number
    - ReceiptDate : Date
    - GameID :Bought game
    - Refund : Refund status (May be “REQ” for requested, “APPROVED” or “REJECTED”
  + Users
    - UserID : PK
    - Username
    - AdminCheck : Gets “1” for admins, “0” for the customers.
  + Shopping\_Cart

This table contains every item which left in the shopping cart, then reads them using Java and SQL languages.

* CartID : PK
* GameID : Game bought
* G\_Name : Game name
* Price
* UserID : User who purchased the following game.
* **Algorithm**
* **—NOTE—Admin Account**
* **Username: onur**
* **Password: ucar**

**A screenshot of a cell phone

Description automatically generated**

**—NOTE—Admin Account**

**Username: onur**

**Password: ucar**

* Connect the program to a database, using JDBC.
* Create a signup page for users to enroll to the system if it’s their first time. (Check for the empty text fields, check whether given passwords match or not, check whether the username given exists or not.)
* If username and password given match with the data in the database, let the user in.
  + If user has the value of “0” in adminCheck column in the database, get him to the customer main menu. (Signup page assigns “0” to adminCheck value of new users.)
  + User will be able to see his previous orders (doesn’t matter if it’s refunded or not), buy a new video game and change his password.
    - Let the user choose a platform and/or category in the main menu (combo boxes will read the Category and Platform tables from the database) using combo boxes.
    - Save the given inputs and display the games which has the given values in the related columns.
    - If user clicks on the ID, confirm him if he really wants to add the following item into his shopping cart.
    - If he says yes, add the clicked item into shopping cart, if user is done with the shopping cart, ask him to confirm his purchases one more time and display the total amount.
      * After he virtually paid the total amount, subtract one from the clicked game’s stock and get him to the main menu.
  + Once the user bought an item from the store, his receipt will be saved into the file.
  + After the user purchased an item, he can see his transactions in “Your Orders” tab. Also, he’ll be able to “ASK” for a refund, clicking the order. (Requested refunds may be approved or rejected in admin dashboard.)
    - User is allowed to prints out all his orders in this tab.
  + If user wants to change his password, ask for the confirmation, check for the old password and check for the new text fields to see if they match.
  + If user has the value of “1” in adminCheck column in the database page, get him to the admin dashboard
    - Admin is allowed to add a new game, edit a game, delete a game, add a platform, edit a platform, add a category, delete a category, add items into the stock, see the pending refunds, see the refunded items and sold items.
    - If admin confirms a refund request, the customer who requested the refund will see “APPROVED” in his refund status.
* **Some Screenshots**

**A screenshot of a cell phone

Description automatically generated**

Wrong username or password error in login

**A screenshot of a cell phone

Description automatically generated**

Existing username error

**A screenshot of a cell phone

Description automatically generated**

Empty text field(s) error in signup

**A screenshot of a cell phone

Description automatically generated**

Another password related information error in signup page

**Stock System**

**A screenshot of a cell phone

Description automatically generated**

Let’s buy The Last of Us.. (Currently 4 in stock)

**A close up of a logo

Description automatically generated**

After we purchased it. (3 left)

**A screenshot of a cell phone

Description automatically generated**

Let’s go ahead and request refund for the game we just purchased (because why not)

**A screenshot of a cell phone

Description automatically generated**

Let’s confirm the refund request. (This screen shot from admin dashboard)

****

It has changed back to 4.

A screenshot of a cell phone

Description automatically generated

Prints out the orders we purchased until today. (Note that, there is also a receipt system in the program. Please check the folder after you purchased a video game, it’ll generate a receipt with a unique receipt number.)