**System Implementation: README**

1.) Database Implementation

1.1.) Database Creation

Class attributes were taken from the original Class diagram and placed into a tabular format (Connolly & Begg, 2014). The attributes had to be slightly modified to align with normalisation and relationship requirements (Connolly & Begg, 2014; University of Essex, 2021a). For instance, instead of having a singular name column, “firstName” and “lastName” columns were created to fulfil the atomic value requirement (Connolly & Begg, 2014; University of Essex Online 2021a). Furthermore, various tables were created to align with normalisation dependency requisites (Connolly & Begg, 2014). Additionally, to form relationships primary and foreign keys were assigned (Connolly & Begg, 2014). To prevent having to reuse multiple attributes across various tables, where certain primary keys pertaining to a particular table needed to be represented in a child table, IDs were created in parent tables and used in child tables to define unique credentials (Connolly & Begg, 2014).

The Database was implemented in a MySQL Relational Database Management System (DBMS) using Structured Quey Language (SQL) (MySQL, N.D). A schema was created first, thereafter selected for use, and then populated with the various designed structures (University of Essex Online, 2021b). Eleven tables were created, with each having a purpose as presented in the points below:

1. “CustomerDetails” is a parent table defining a customer’s main unique and generally unchanging characteristics (Connolly & Begg, 2014).
2. “LoginDetails” is a child table representing the login credentials of a user, and contains a “loggedIn” attribute which specifies whether a user is logged in or not. It is linked to its parent “CustomerDetails” via its foreign key “customerID” (Connolly & Begg, 2014).
3. “CustomerFinancialDetails” contains attributes able to store a customer’s bank details for purchase purposes. It links back to the “CustomerDetails” table via the “customerID” field (MySQL, N.D).
4. “CustomerAddress” represents the address details of a customer to be used for delivery of purchased goods.
5. “Products” is a parent table representing details of an individual item that will be for sale. An item has a unique ID attached to it, which automatically increments each time a new item is added to the database (MySQL, N.D).
6. “Stock” is a child table of “Products” and contains all the serial numbers pertaining to a particular product, allowing for a type of item to be more granularly tracked by entering the unique serial number for each item in stock. This table also allows the stock count for a particular item to be calculated (University of Essex Online, 2021b).
7. “Orders” contains all orders that are placed by customers. It contains which customer made the order, the product that was ordered, the warehouse that needs to provide the product, and the delivery option selected by the customer.
8. “Warehouses” is a parent table representing a particular warehouse in the database. This table was created so that multiple warehouses could be facilitated by the information system, allowing for future expansion (Sommerville, 2016).
9. “WarehouseProducts” is a child table of both “Warehouses” and “Products” reflecting which warehouse contains which products. A product is assigned a location within the warehouse via this table.
10. “Stores” is a parent table which identifies various stores that are available to the customer via the front-end. This table was created so that multiple stores could be facilitated by the information system, allowing for future expansion (Sommerville, 2016).
11. “StoreWarehouses” represents which warehouses, and therefore products, are available to particular stores.

Once the structures had been created, confirmation was performed using the SQL “SHOW DATABASES”, “USE *DATABASE*”, “SHOW TABLES”, and “SHOW COLUMNS FROM *TABLE*”, commands (University of Essex Online, 2021b; Connolly & Begg, 2014).

1.2.) Database Population

Once the database had been created, it was populated with specific initial data to enable the system to be tested (Brookshear & Brylow, 2018). The data was inserted according to below:

1. The "Products” table was filled with various products.
2. Serial numbers were entered into the “Stock” table relating to the various items entered into the “Products” table.
3. A main warehouse’s details was inserted into the “Warehouses” table.
4. Products were allocated to the main warehouse in the “WarehouseProducts” table.
5. A main store was created in the “Stores” table.
6. The main warehouse was allocated to the main store in the “StoreWarehouses” table.

Note that “NULL” was used in columns that had auto-increment set, to instruct the DBMS to perform the incrementation (MySQL, N.D). Both the schema setup and the data population were performed in Codio via the “Store\_database\_Setup.sql” file. Once the data had been populated confirmation was performed using the SQL “SELECT \* FROM *TABLE*” command (University of Essex Online, 2021b).

2.) Application Implementation

2.1) Philosophy

To produce the relevant Python code, the design laid out in the Class Diagram was followed (Phillips, 2018). Functional and domain specific methods and attributes were tied as best as possible to relevant classes to fulfil the principle of encapsulation (Phillips, 2018). However, due to normalisation and the impedance mismatch between the Python code and the structures pertaining to the Relational Database Management System (RDBMS), class attributes do not necessarily directly reflect the database tables (Connolly & Begg, 2014). Furthermore, certain attributes initially thought of in the design phase as existing permanently in computer memory with the application in the form of objects, were made transient as they would now be pushed to the database once fully processed by the application. In hindsight, it may have been beneficial to first create the database design with an Entity Relationship Diagram, before designing the application with the aid of a Class Diagram (Connolly & Begg, 2014). Additionally, by moving data to the database the principle of private attributes was broken, as they could be fetched anywhere in the code using the correct SQL statements (University of Essex Online, 2021c). However, during coding of the system the paradigm of encapsulation was strictly followed (University of Essex Online, 2021c). Various security concerns like storing passwords as salted and hashed values was not considered, neither was aspects of SQL injections and the like (Nieles et. al, 2017).

2.2) Programming

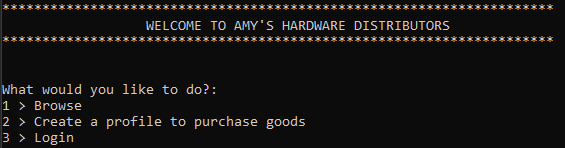
Initially the Python modules relating to MySQL database interactions were downloaded and installed via the Python Installs Packages (PIP) program, accessible via command prompt in the Windows environment (MySQL, N.D; Python, N.Da). Please note that for Codio the same modules were installed using specific Linux commands (MySQL, N.D). To setup the program with the functionality pertaining to a MySQL database, relevant modules were imported, and functions to perform write and read respectively, were coded (MySQL, N.D; Python, N.Db). The various designed classes were then fully programmed, with the exception of the “Employee” and “ProductType” classes displayed in the original design, as these were functionally required by a business management software application, rather than one used for customer interaction. Thereafter, the required objects were instantiated in the codes main body, and the objects interfaced in a small portion of code (Phillips, 2018). The source code is contained in the “Store\_connected\_to\_database.py” file.

Please note that although the system design took into account expansion to accommodate multiple stores and warehouses, the code was adapted to facilitate the interaction of only one of each (Sommerville, 2016). Furthermore, a separate implementation (“Store\_run\_in\_Local\_Memory.py”) has been submitted reflecting the code should it not have had a database to move object data to. In this version of the implementation there are various Object-Oriented Programming techniques visible which were not explicitly required in the version using the database for persistence. Please refer to the actual code in the source files and pertaining comments to understand all technicalities.

2.3) Execution Instructions and Testing

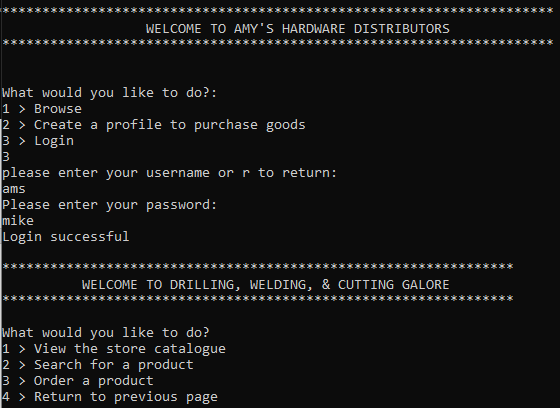
Testing was performed at various levels of functionality (Brookshear & Brylow, 2018). Each method within a particular class was tested once implemented (Brookshear & Brylow, 2018). Furthermore, each object pertaining to a class was tested including the interfacing of objects after connection (Brookshear & Brylow, 2018). Data inputs relevant to various attributes are tested within the code to ensure that correct values were inputted by a user (Brookshear & Brylow, 2018). Error handling was put in place to manage various exceptions that might be raised (Brookshear & Brylow, 2018). It must be noted that extensive testing of each input is not performed as well as might be required in a real-life implementation (Brookshear & Brylow, 2018). For instance, a credit card number may be fixed at 16 digits, and less or more entered should be prevented by the code. Another example may be an email address not in the correct format, or a name that is too long. These sort of tests require extensive time and extra code, and therefore were not added. There are also some further escape and return options required in the code to allow more usability. Testing often incorporated either hardcoding test data and interacting with the program, or using print statements to ensure the correct data values were being moved around during different steps within the program (Brookshear & Brylow, 2018). Additionally, debugging was performed using Microsoft Visual Studio where variable values were noted as the program was stepped through each line. It was necessary that all lines of code were executed to ensure that no errors were present when executed (Brookshear & Brylow, 2018). When writing to the database relevant tables were then displayed directly via the database interface to confirm that data was correctly presented (MySQL, N.D). Performance issues pertaining to functionality like large-scale searching was not tested, neither was any specific efficiency considered by representing algorithms by Big O Notation (Brookshear & Brylow, 2018).

When interacting with the program the first menu that will appear is shown in Figure 1. A user may then either browse items available in the store without logging in, create a user profile, or login if a profile already exists. A user cannot make a purchase unless they have logged in.



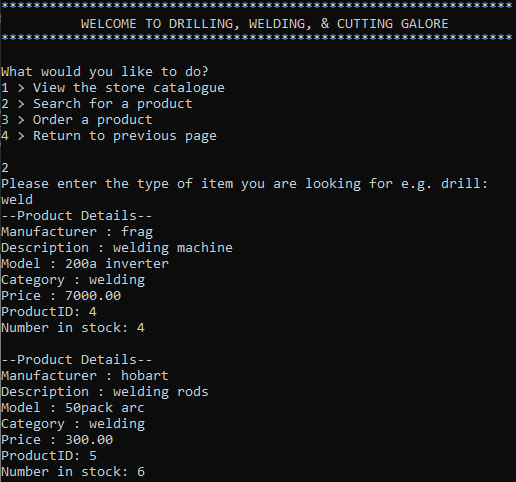
**Figure 1 – Initial Menu Options**

Selecting option 2 will initiate a process to guide a user through entering specific personal information and a username and password. Thereafter, a user can login. Once logged in or selecting to browse, the menu shown in Figure2 appears. Selecting option 1 will display all products available in the store. Choosing option 2 will require a user to insert a portion of the type of item they are searching for.



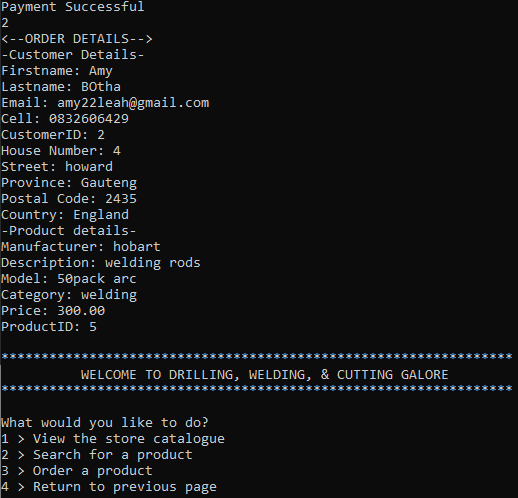
**Figure 2 – Second Menu Options**

An item is displayed as in Figure3. The “ProductID” must be taken note of, as it will be used to select a product when placing an order. Once deciding to place an order the user guided through the process of entering the required information.



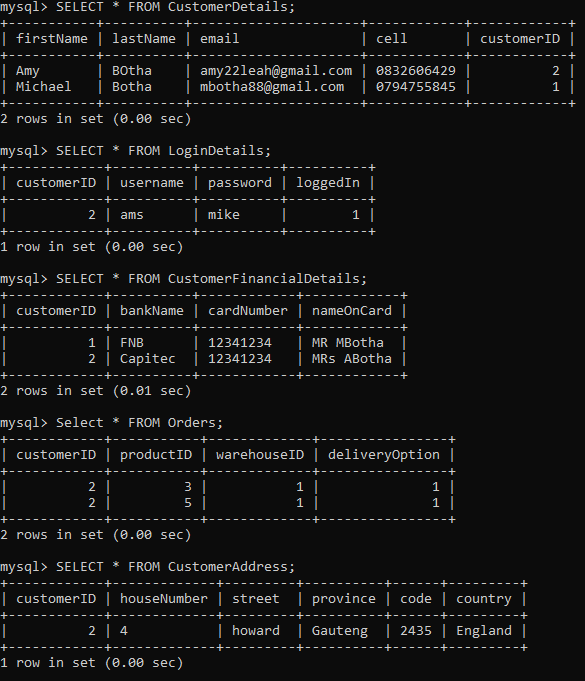
**Figure 3 – Viewing Product Details**

Once an order has been successfully paid for the program prints confirmation to the customer along with all the product details as per Figure4.



**Figure 4 – Successful Order Submission**

Once a full session had been performed with the system, tables were checked directly in the database to ensure the presence and validity of information (MySQL, N.D).



**Figure 5 – Database Confirmations**

3.) References

Brookshear, J., Brylow, D. (2018) *Computer Science: An Overview*. 13th ed. London: Pearson.

Connolly, T., Begg, C. (2014) *Database Systems: A Practical Approach to Design, Implementation, and Management*. Pearson Education available from: https://ebookcentral.proquest.com/lib/universityofessex- ebooks/detail.action?docID=5174902 [accessed 27 September 2021].

MySQL. (N.D) MySQL 8.0 Reference Manual. Available from: <https://dev.mysql.com/doc/refman/8.0/en/> [Accessed 11 October 2021]

Nieles, M., Dempsey, K., Pillitteri, V. (2017) *An Introduction to Information Security*. Revision 1. United States of America: National Institute of Standards and Technology. Available from: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-12r1.pdf>  [Accessed 18 September 2021].

Phillips, D. (2018) Python 3 Object-Oriented Programming. 3rd Edition. Birmingham: Packt Publishing.

Python. (N.Da) Installing Packages. Available from: <https://packaging.python.org/tutorials/installing-packages/> [Accessed 11 October 2021]

Python. (N.Db) Importing Modules. Available from: <https://docs.python.org/3/library/modules.html> [Accessed 11 October 2021]

Sommerville, I. (2016) *Software Engineering*. 10th ed. Essex: Pearson Education Limited.

University of Essex Online. (2021a) *Database Design* [Lecturecast]. OOIS\_PCOM7E AUGUST 2021 Object-oriented Information Systems. University of Essex Online.

University of Essex Online. (2021b) *Introduction to MySQL* [Codio Lessons]. OOIS\_PCOM7E AUGUST 2021 Object-oriented Information Systems. University of Essex Online.

University of Essex Online. (2021c) *Encapsulation* [Codio Lessons]. OOIS\_PCOM7E AUGUST 2021 Object-oriented Information Systems. University of Essex Online.