COMP642 Advanced Programming
Semester 2 2024

Project - Software Development

Worth:	40%
Due:	**Friday, 1 November 2024 5:00 p.m.**
Late Penalty:	Work not received by the due time attracts an immediate penalty of up to 25% of the marks available. No work will be accepted after **Sunday, 3 November 2024 5:00 p.m**.
Submission:	Zip your completed files and submit the .zip through the link on COMP642 Akoraka | Learn page.

This is an **individual** assessment. You must not collaborate or confer with others. You may help others by verbally explaining concepts and making suggestions in general terms, but without directly showing or sharing your own code. You must develop the logical structure, the detail of your code and the database on your own, even if you are working alongside others. Code that is copied or shares a similar logic to others will receive zero marks for both parties.

The use of Artificial Intelligence (AI) tools, such as ChatGPT, to complete this assessment is **prohibited**. Assessment answers will be analysed for evidence of the use of AI and penalties may be administered.

The University policy on Academic Integrity can be found here.

Introduction

Your task is to create a Python application which uses the model classes that you have designed in the first part of this project. Please refer to the feedback provided for the first part to improve your design. Alternatively, you may you use the class diagram shown in Appendix A. Note that this is a simplified class diagram and shows the attributes only (and not the methods). You may also add additional attributes to each class as you see fit.

You are required to code your design (possibly amended after feedback) and create a GUI (using tkinter or Flask and Python) that works with your design. You will also write test cases to verify the functionalities of your software application.

Customers should be able to place orders, view their current orders and past orders, cancel current order (if the order is not processed yet), make payments, and view their information.

Staff should be able to view current orders, fulfil the orders, and update the status of the orders. Staff can also use the application to generate several reports for the company.

Requirements

1. Adapt your design from Project Part 1 as necessary, so that the following functionalities can be implemented for customers and staff. You will need to use SQLAlchemy to store the objects to a MySQL database.

   Customers can:

   1. Log in and Log out.
   2. View available vegetables and premade boxes.
   3. Place order for vegetables and premade boxes. Premade boxes need to be assembled based on the size. At checkout, pay for the item using credit card or debit card or charge the amount to their account.
   4. View current order details.
   5. Cancel current order if the order has not been fulfilled.
   6. View previous orders details.
   7. View their own details.

   Staff can:

   1. Log in and Log out.
   2. View all vegetables and premade boxes.
   3. View all current orders and their details.
   4. View all previous orders and their details.
   5. Update an order status.
   6. View all customers and their details.
   7. Generate a list of all the customers.
   8. Generate the total sales for the week, month, and year.
   9. View the most popular items.

2. Implement an appropriately designed view. Your view should be an interface with appropriate controls, useful feedback, and exception handling. You may use tkinter or web application to implement the view.
3. Perform error handling and prevention mechanisms to ensure that the application is robust, reliable, and resilient.
4. Write and test your code for all the model classes and other components in your design using **pytest**.
5. Your code must be clear and easy to maintain, and appropriately commented.
6. Your application **does not** have to provide facilities for staff to add new customers/vegetables, delete existing customers/vegetables, or amend customer/vegetable details. It also **does not** have to provide facilities for a customer to amend their details.

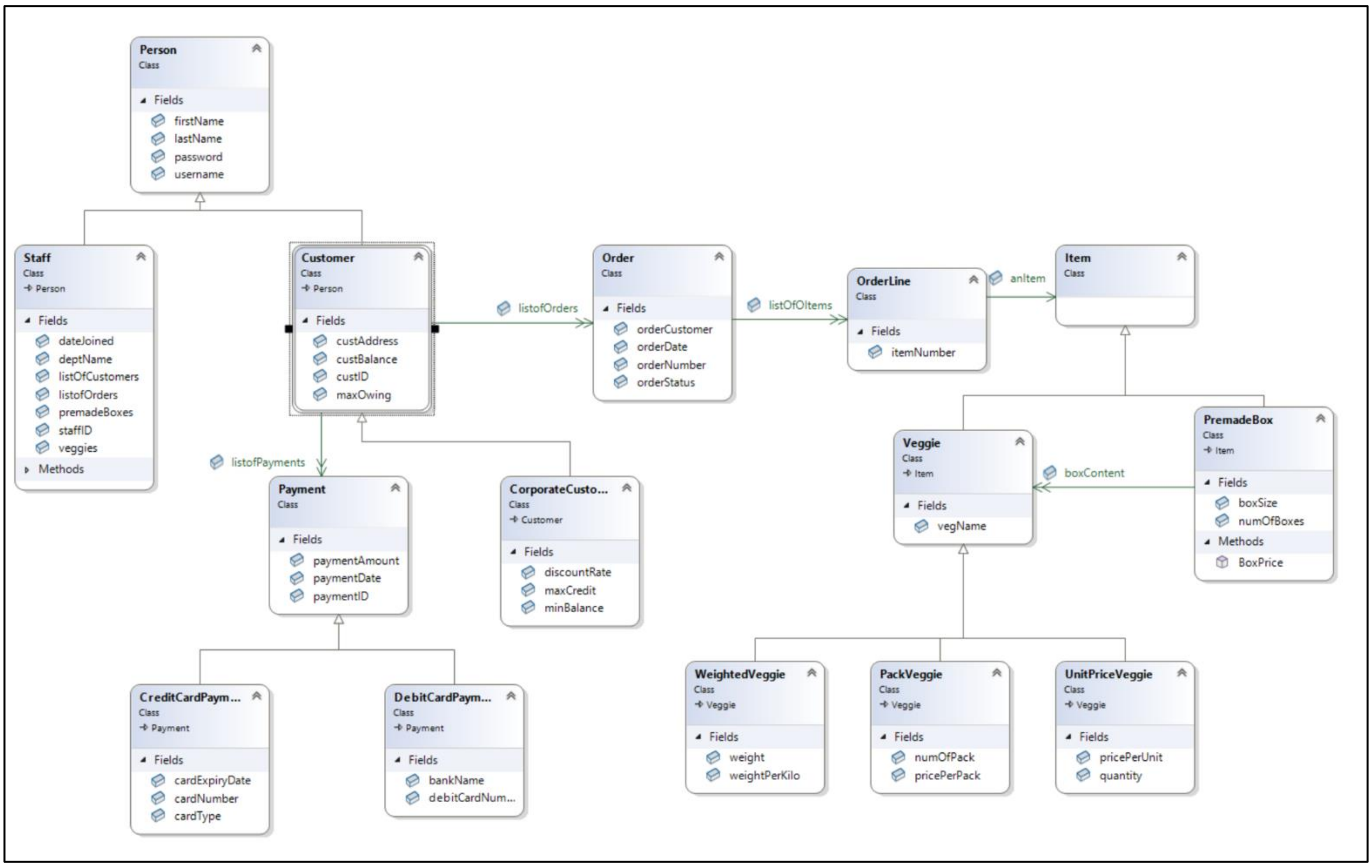
Marking Criteria

Criteria	**Marks (out of 140)**	**Mark Range**
Application functionalities: Customer (20) Staff (20)	**40**	Marks will be assigned using the following criteria: All requirements met (81% - 100%) Some requirements met (51% - 80%) Minimum requirements met (1% - 50%)
Application Interface	**10**	Excellent user interface, intuitive, and user friendly. Widgets are well-chosen, effectively implemented, and enhance the user experience (9 – 10). The user interface is intuitive and user-friendly. Widgets are mostly appropriate and functional but may have minor usability issues (7 – 8). The user interface provides basic functionality but lacks in intuitiveness and user-friendliness. Widgets may be poorly chosen or implemented, leading to confusion or difficulty in their use (5 – 6). The user interface is not intuitive or user-friendly. Widgets are poorly chosen or implemented (1 – 4).
Error Handling and Preventions	**10**	All relevant errors are detected and handled appropriately (9 – 10). Some errors are detected and handled appropriately but may miss some less common issues (5 – 8). Minimal or ineffective error detection; many errors are not identified (0 – 4).
Testing	**10**	Comprehensive test coverage with all relevant areas of the application thoroughly tested, including edge cases (9 – 10). Most key areas are tested, though some gaps may exist. The test suite covers essential functionality and interactions but might miss a few edge cases or less common scenarios (5 – 8). Significant gaps in test coverage, with critical areas or edge cases often untested. The test suite lacks depth and fails to cover many aspects of the application (1 – 4).
Code	**70**	See Code Evaluation Rubric.
Total	**140**	

Code Evaluation Rubric

Attribute	High (8 -10)	Moderate (4 – 7)	Low (0 – 3)
Code Quality Correctness (10)	The code is free of bugs and meets all requirements.	The code mostly functions correctly with some minor bugs or edge cases not fully addressed.	The code contains significant bugs or errors affecting functionality.
Code Quality Standards (10)	The code strictly follows best practices ensuring robust, secure, and reliable code.	The code generally follows standards but has minor deviations or inconsistencies.	The code does not adhere well to standards is inconsistent and prone to issues.
Readability and Maintainability Readability (10)	The code is highly readable with clear variable names, consistent formatting, and well-organised structure. Complex logic is well-documented.	The code is generally readable with mostly clear names and comments. There may be minor issues with formatting or organisation.	The code is difficult to read, with unclear names, inconsistent formatting, and insufficient comments. Understanding and maintaining the code is challenging.
Readability and Maintainability Maintainability (10)	The code is modular and easy to maintain, with clear separation of concerns and minimal risk of introducing new issues when updating.	The code is somewhat maintainable but may require effort to update or extend. Some areas could be improved for better modularity.	The code is hard to maintain or update due to poor structure and lack of modularity. Changes are likely to introduce new issues or require significant effort.
Comments Clarity (10)	Comments are clear, concise, and provide meaningful explanations of complex logic. They enhance understanding without stating the obvious.	Comments are mostly clear but may be slightly redundant or lack detail in some areas.	Comments are unclear, inconsistent, or insufficient, making it difficult to understand the code.
Comments Coverage (10)	Comments comprehensively cover key sections of the code, including purpose, functionality, and non-trivial implementations.	Comments cover most key sections but may have minor gaps or areas where additional detail would be useful.	Comments are sparse or missing, leaving key sections of the code poorly explained or undocumented.
Efficiency (10)	The code is highly efficient and well-optimised, balancing performance with readability and maintainability.	The code is reasonably efficient but may have some areas for improvement. It is somewhat optimised but could benefit from refinements.	The code is inefficient, either due to a brute-force approach or being overly complex and patched together. It lacks effective optimisation and resource management.

Appendix A - Class Diagram


Person
Class
Fields
firstName
lastName
password
username
Staff
Class
Person
Fields
dateJoined
deptName
listOfCustomers
listofOrders
premadeBoxes
staffID
veggies
Methods
Customer
Class
Person
Fields
custAddress
custBalance
custID
maxOwing
listofOrders
listofPayments
Order
Class
Fields
orderCustomer
orderDate
orderNumber
orderStatus
listOfOItems
OrderLine
Class
Fields
itemNumber
anItem
Item
Class
Payment
Class
Fields
paymentAmount
paymentDate
paymentID
CorporateCusto...
Class
Customer
Fields
discountRate
maxCredit
minBalance
Veggie
Class
Item
Fields
vegName
boxContent
PremadeBox
Class
Item
Fields
boxSize
numOfBoxes
Methods
BoxPrice
CreditCardPaym...
Class
Payment
Fields
cardExpiryDate
cardNumber
cardType
DebitCardPaym...
Class
Payment
Fields
bankName
debitCardNum...
WeightedVeggie
Class
Veggie
Fields
weight
weightPerKilo
PackVeggie
Class
Veggie
Fields
numOfPack
pricePerPack
UnitPriceVeggie
Class
Veggie
Fields
pricePerUnit
quantity