Programming III – Fall 2023

Course Project: All-you-can-eat ordering app.

Team Information

Team VM:

Member A: Veronika Vilenski - 0961951

Member B: Madison Cassley - 2231694

Project Description

The Project we are building is an application that allows the user to order food from a provided menu. Each menu item has a name, description, and picture (ex: Name: “California Roll”, Description: “Yummy”, ImageFileName: “sushi2.jpg).

From the menu you can scroll or click on tabs with categories to find the item you wish to order. You can add as many items as you wish to your order, if the item already exists in your order, it will add more to it, if it doesn’t exist, it will create a new place for that item. You can also remove any items from your order. When you are satisfied with your order you can send it and voila! Order Made.

There is also an Admin window for people to see every order made from a certain date to another certain date. On this window you can see the order number, date, time, item(s) ordered and the quantity.

Development Approach

Explain how you prepared for the project. You can use the 5 steps of algorithmic thinking to help build this section (you will need to elaborate on each step).

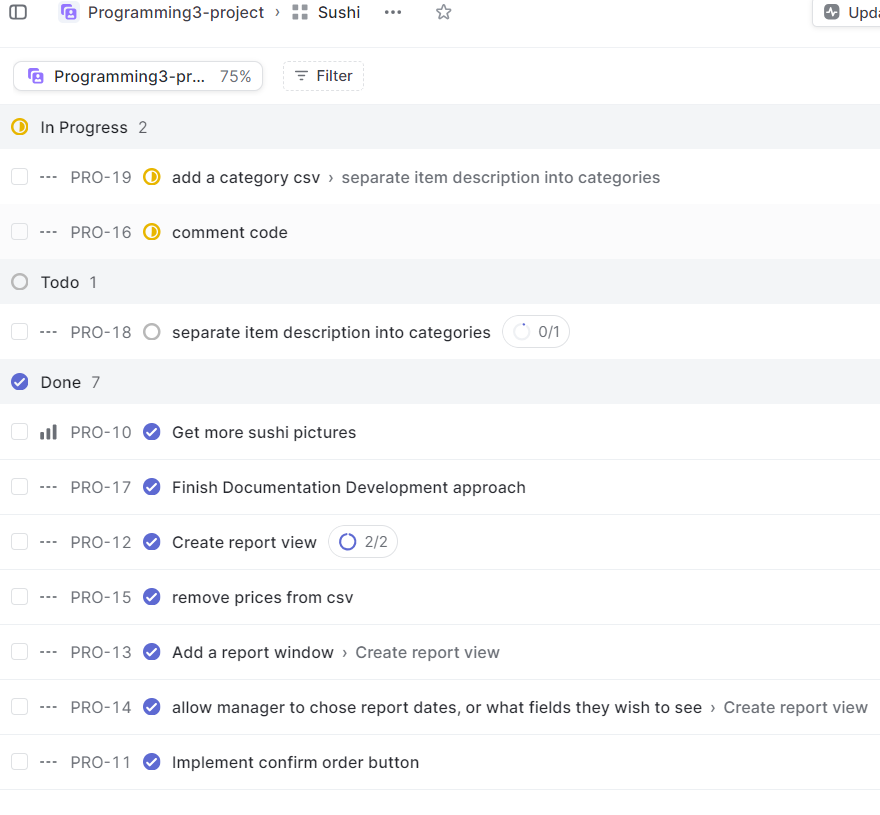
1. Understanding the problem.

We looked at the requirements given to us for this project by Aref and Veronika came up with the amazing idea of making our topic an ordering app with some of our favorite foods. The data that will be given, or our input, is information on the food items we want to make available in our “restaurant” for our ordering app. We will be receiving the name, description, and image of each item we decide to have available to order. There will also be different categories based on the available food items (drinks/food/plates/etc.). We expect as an output an application window that takes the data and presents it on screen so that users can interact with it and create orders using the items and categories.

1. Formulating the problem.

The input of items and categories can be represented using csv files with written data so that our app can process it. We can write the name, description, and a path to an image for each item. We can also have a csv file for each category containing the name of the category. The output will be represented on an app designed with OOP and WPF programming on C#. We expect the program to read the data from the csv files and produce a visual output for users to interact with.

1. Developing the application \ algorithm.
2. Create csv files with all info about our items (name, description, path to image)
3. Use a method to read from csv files
4. Implement classes to handle data reading and different app aspects (back-end)
5. Create visuals to display data and be interactive (front-end)

An example of Developing the application is our use of Linear, we created easy-to-do tasks and problems and worked on them one by one to build the project. You can view an example screenshot below in the ‘Contributions’ section. Here is another screenshot of some of the many tasks we created:  


1. Implementing the application \algorithm.

Chosen Language: C#-WPF

Chosen Program: Microsoft Visual Studio 2022

Using C# was better for us because it is what we are both most comfortable with and use the most often. It is also very useful for its WPF which allows us to make an interactive app. It's known for its strong support of object-oriented programming (OOP) principles, which help in organizing and structuring code. For instance, classes and objects in C# facilitate the creation of reusable code components, making it easier to manage complex projects. Additionally, C# boasts a robust ecosystem with tools like Visual Studio, making development more efficient. Overall, C# empowers developers with a powerful language for building scalable and maintainable applications.

1. Testing.

All testing of this project can be viewed in the App snapshot sections where you can view the progress and output of the app.

 OOP Design

Talk about the classes you need to create for the application and what is the purpose of each class. Include the UML class diagram in this section. The UML class diagram should include the relations between the created classes. Do not mention the WPF classes (Window, etc.)

There are 6 non WPF classes that we created for this project:

1. CsvReporter
   * The CsvReporter class plays a crucial role in managing CSV reports related to order processing. It centralizes various operations such as writing new order details to a CSV file, reading and interpreting existing data, and extracting key information like the next order number. Additionally, it provides functionalities to convert order data into a list of **OrderDisplayModel** objects and filter this data into a structured **DataTable**. These capabilities are essential for handling and analyzing order data efficiently within the application. By managing CSV files for order reports, this class enables the application to store and retrieve order-related information in a structured, accessible format, making it an integral component for data management and reporting in our system. IS ASSOCIATED TO OrderDisplayModel BECAUSE IT USES IT IN ITS METHODS
2. MenuCategory
   * The MenuCategory class serves as a fundamental building block for organizing menu items in our application. Each instance of this class represents a specific category of menu items, encapsulated by its **Name**. The class also maintains a list of **MenuItem** objects, which are the individual items belonging to that category. By structuring the menu this way, our application can easily group related items, enhancing both the organization of the menu and the user experience. For example, in a restaurant application, **MenuCategory** could represent groups like "Appetizers", "Main Courses", "Desserts", etc., each holding their respective menu items. This class simplifies the management and display of menu items, making it easier for users to navigate and for the backend system to manage the menu's structure and content. HAS AN AGGREGATION RELATIONSHIP TO MenuItem (MenuCategory contains MenuItem Objects but does not strictly own them)
3. MenuItem
   * The MenuItem class is another key component to our project, encapsulating the details of individual menu items such as their name, image, description, and category. It features both static elements, like the **LoadedMenuItems** list which stores all menu items loaded into the application, and dynamic aspects, such as the ability to load menu items and their categories from CSV files. This class not only provides a structured representation of each menu item but also plays a crucial role in organizing and managing the menu items efficiently, enhancing the user's navigation and interaction with the menu in our application. Top of Form
4. Order
   * The Order class in our project is a central element that represents a customer's order, encapsulating details like order date, time, and items. It operates primarily with static methods and properties, allowing for a unified management of order details across the application. The class maintains a static list of **OrderItem** objects, representing individual items in an order, and a static **OrderNumber** for tracking the unique identifier of each order. Key functionalities include adding, updating, or removing items from the order, calculating the total item count, and managing order numbers. It also provides methods to set or retrieve the order date and time, and to convert order details into a CSV format for reporting or record-keeping. This class is crucial for handling the lifecycle of an order in our application, from creation and modification to final processing and data export, making it a fundamental component for order management and processing.
5. OrderDisplayModel
   * The OrderDisplayModel class serves as a specialized model for presenting and handling order details. This class is designed to encapsulate various attributes of an order, such as order number, date, time, total, and item specifics like name and quantity. It features both default and parameterized constructors, allowing for flexibility in creating instances with varying levels of detail. A significant aspect of this class is its attention to data formatting and validation, particularly in the way it manages the order date to ensure consistent formatting across the application. The **OrderDisplayModel** functions as a structured and readable format for displaying order details, making it crucial for user interfaces or reporting functionalities where clear and accurate representation of order data is essential. This class effectively bridges the gap between raw order data and its user-friendly presentation.
6. OrderItem
   * The OrderItem class in our project, an extension of the **MenuItem** class, represents an individual item within an order. Its primary function is to encapsulate not only the properties of a **MenuItem** but also the specific quantity of that item as ordered. The class ensures the integrity of data through its **Quantity** property, which includes validation to prevent negative values, thereby maintaining the logical consistency of an order. Key to its functionality is the ability to construct an **OrderItem** instance from a **MenuItem** and a specified quantity, linking the general menu item details with specific order-related information. Additionally, it provides a method **ToCsv()**, which formats the **OrderItem** details into a CSV-friendly string, a useful feature for data export and reporting. This class plays a crucial role in bridging the gap between the menu items available for order and the specific details of a customer's order, making it a fundamental component in the management and processing of orders in your application. HAS AN INHERITANCE RELATIONSHIP TO MenuItem

UML DESIGN:  
A diagram with text and images

Description automatically generated with medium confidence

Contributions TODO: MAKE TABLE FORMAT

Each team member worked hard on the project. We bounced ideas back and forth and ensured the proper functionality of the code. Veronika worked mainly on code and algorithm while Madison worked on documentation, UML, and the code wherever possible. Overall, it was a project that played into the strengths and weaknesses of both team members. We used the website “Linear” To work together and improve efficiency. This website allowed us to keep track of who is doing what and when so we don’t overlap and can see the progress of each person. Here is a sample snapshot of our Linear page when we were working on some specific tasks:

A screenshot of a chat

Description automatically generated

App Snapshots

This section includes snapshots of the final application showing different features. It could be a guideline to using the application. You may include snapshots of the app while being developed. Remember to add explanatory captions to the snapshots.

A screenshot of a computer

Description automatically generated

This is one of the first implementations of our main ordering screen. December 12

A screenshot of a computer

Description automatically generated

This is an unfinished window where it would show the details of an item and give you the option to add it to your order

December 12th

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedGif Made by Maddie showing small app tests. App is not complete but moving along December15th

New base white design for app background (Pictures and orders still exist)

New Admin window to track orders

Completed App design with Category tabs for easier scrolling

A screenshot of a computer

Description automatically generated

A screenshot of a food

Description automatically generated

Future Work

For future enhancements to our application, we could consider integrating advanced data analytics and personalized recommendations based on user preferences and order history. This would involve implementing machine learning algorithms to analyze order patterns, which could significantly enhance the user experience by suggesting items they are likely to enjoy. Additionally, introducing a user-friendly interface with real-time order tracking and a feedback system could improve customer engagement. On the technical side, expanding the application to support cloud-based data storage would ensure scalability and accessibility. Finally, integrating with third-party APIs for payment processing and delivery logistics could streamline the end-to-end ordering process, making it more efficient for both users and administrators. These improvements not only add value to the user experience but also pave the way for our application's growth and adaptability in a competitive market.

Used Packages

**WPF Animated GIF**

<https://github.com/XamlAnimatedGif/WpfAnimatedGif>

**Material Design In XAML**

<https://github.com/MaterialDesignInXAML>

Appendix A: Team Contract

Team Contract is available in the Documentation folder within the project

Appendix B: UML Class Diagram

* DO NOT PLACE A  LINK TO THE DIAGRAM.
* Do not include WPF-created classes in the class diagram.
* The diagram should be placed in the document.

IT IS SHOWN ABOVE IN THE OOP DESIGN SECTION

## **REFERENCES WE USED:**

IsUserVisible Class:

<https://stackoverflow.com/questions/1517743/in-wpf-how-can-i-determine-whether-a-control-is-visible-to-the-user>

VisualTreeHelper:

<https://learn.microsoft.com/en-us/dotnet/api/system.windows.media.visualtreehelper?view=windowsdesktop-8.0>

CollectionView:

* <https://learn.microsoft.com/en-us/dotnet/api/system.windows.data.propertygroupdescription?view=windowsdesktop-8.0>
* <https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/collectionview/?view=net-maui-8.0>

Rect struct:

<https://learn.microsoft.com/en-us/dotnet/api/system.windows.rect?view=windowsdesktop-8.0>

DependencyObject Class:

<https://learn.microsoft.com/en-us/dotnet/api/system.windows.dependencyobject?view=windowsdesktop-8.0>

Project Management:

<https://linear.app>