Design Methodology

Tools/Techniques being used

Version control system (VCS)

A VCS is a repository that is used to keep track of file/folder changes that are made over time. Git/GitHub, a decentralised VCS, will be used for this project. This simple application will allow me to keep track of every line of code I write and see my progress throughout the project. I will also not have to worry about losing any of my work, as I this will be saved on the repository. Another benefit is that I can keep different versions of development efforts separate. This independence will greatly benefit me as I can work on multiple parts of the project without worrying about the effect it may have on other aspects.

My repository structure will be based on a branch and merge process, with all functioning code preserved in the repository's main folder (main branch), and ongoing work in a copy of this folder (working branch). All completed work will then be merged into the main folder (branch). The goal of employing this structure/process is to avoid breaking the functioning code in the main folder with incomplete code which will still be in development. The consequences of merging incomplete code could lead to conflicts and fatal errors. This is a common industry practice.

Programming approach

For this project, I will be employing an iterative methodology, specifically following an agile approach. There are several reasons behind this choice. Agile methodology facilitates incremental planning and implementation, allowing for easier adaptation of processes and designs without the need to restart the entire project. This flexibility suits me well as I have uncertainties about the final product's specifications. In case my initial plans prove too ambitious for the project's timeframe, an agile environment enables prompt adjustments. Moreover, incremental development practices like agile enable rapid software delivery. Given the time constraints of this project, I aim to deliver a functional increment of the final product, even if I can't complete the entire project within the given timeframe.

In contrast, the rigid waterfall model requires completing each phase in the development life cycle before progressing to the next. This means the entire product must be developed before any testing or improvements thereafter can be made. The main drawback of the waterfall process is the difficulty of accommodating change as it is not an iterative design methodology. Rather it is a plan-driven model with distinct phases that cannot interleave. Therefore, this model is only appropriate when the requirements are well understood/planned out and changes will be limited during the development process. But as mentioned, during this project I will be deciding to add/remove functionality based on several factors, the main being time constraints.

Platform/Technology Selection

During this project, I will encounter and use many different technologies. For example, HTML, CSS and JavaScript are essential when it comes to creating web applications to provide page structure/organisation, design and interactivity.

I will also be using a database (SQL Server) provided by ASP.NET core on Visual Studios to store customer and order-related information to facilitate CRUD operations by the kitchen staff as it is more efficient (memory and time) than storing copious amounts of information within, for example, a HashMap, List or other types of objects as not only would this use up a lot of memory and degrade the performance of the application, but also wouldn't scale well and would result in a lot of coupling within the system to retrieve order related information and/or to make updates. This could potentially pose challenges down the line if I opt to expand the application, integrating new elements like menu options, user logins, or webpage features such as a branch finder. I've also decided to use this as it seamlessly integrates into the development environment.

C#, ASP.NET and Visual Studios

I will also be making use of C# and ASP.NET Core within this project for serval reasons. For starters, ASP.NET Core is part of the larger .NET ecosystem, which includes libraries, frameworks, and tools for various tasks such as authentication, data access, and deployment. During this project, I intend to utilise these built-in features to carry out the aforementioned tasks to build my web application where kitchen staff users can log in to see authorised pages, efficiently retrieve/modify/delete order-related information from a database using the server-side rendering capabilities and, eventually, deploy my work using integrated tools such as Azure.

Not only this, but I will also be using the Visual Studios IDE as it provides an abundance of additional tooling support for ASP.NET Core development, including code refactoring, debugging, and further enhances on unit testing capabilities. All of which facilitate good coding practices from the beginning of the development process rather than being seen as an afterthought. This also helps in reducing the development time of this system.

Last but not least, through the use of ASP.NET, I will gain the opportunity to gain hands-on experience in utilising the MVC architecture to create applications. This design pattern is used to promote a clear separation of concerns, reduce coupling between components and increase cohesion. It's also a common industry practice, so gaining experience in this domain could prove to be fruitful.

Efficiency and cost considerations UML Diagram

Within the class diagram for this system, we see that an MVC architecture is employed. This choice is deliberate, aiming to foster separation of concerns by dividing the system into distinct components, where each task has specific responsibilities and are interconnected through interfaces.

For instance, the Model (M) handles the core business logic, managing the manipulation of order-related data and ensuring accurate communication to the kitchen staff. The Views (V) encompass the graphical user interfaces (GUIs) through which customers browse menu options, add items to their orders, and complete payments. In contrast, the views for the kitchen staff are tailored for order management tasks like reading, updating, and deleting orders. The Controller (C) acts as an intermediary bridging the gap between the View and Model. It translates user inputs from the View—such as submitting payment details or clicking buttons—into actionable commands executed by the Model.

This approach significantly reduces the coupling between the Model and View, facilitating potential adaptability. Should there be a need to integrate the Model with different Views or Controllers, the system can be easily modified and extended. An additional benefit of incorporating the MVC pattern is that it promotes code organisation and reduces development costs as less code will need to be implemented to allow fluent communication between classes.

For the MVC to work effectively, I made use of inheritance and polymorphism, which can be seen through the use of interface class methods found within the view and controller. These methods will be implemented by the respective child/implementation classes and are used to promote code reuse and maintainability.

Furthermore, the class diagram reveals an aggregation of smaller components to build larger ones. This design decision is made to minimise coupling and promote scalability. By structuring menu items in this manner, the system is primed for accommodating additional information and items in the future without necessitating extensive modifications as aggregated components can be built and destroyed separately. This can be seen with the Menu Items and the Menu class, where a Menu Item is built using serval components such as a meal type, nutritional info and ingredients.

With that being said, we also have some classes which are heavily coupled with others. These are necessary as certain objects cannot and should not exist without the existence of previous objects in the diagram as it would make no logical sense. For example, an Order should not exist without having menu items and customer information for that order. Thus, although coupling is usually seen negatively, some coupling is deemed necessary so that we can create whole systems which are logically sound.