# Group name: October17

**u20735929 - Mr. Tshegofatso Mapheto**
**u21598012 - Mr. Dhinaz Rangasamy**
**u18108378 - Mr. Paul Nhlapo**
**u17281904 - Mr. Yeshlen Moodley**
**u19345993 - Mr. Lekgotla Motaung**
**u22552121 - Mr. Yudi Govender**

From the given scenario we have seen the following patterns to be suitable for us to use in order to construct the restaurant  system.

**Chain of Responsibility** - We can use this pattern during the meal preparation stage. The order will first be passed to the head chef, who will then pass it to the appropriate chef (if the order includes a main course, it will be passed to the main course chef and so on). Once the dish has been prepared by the appropriate chef, it will then pass back to the head chef to plate and finish. This concludes the chain of responsibility.

**Observer** - We can use this pattern to check on the current state of an order - a waiting state, a cooking state, a plating state and a ready state. The observer pattern can also be used by the head chef to observe the state of secondary chefs in the kitchen and whether they are busy preparing a meal or not.

**State** - We can use this pattern to model the different states an order may find itself in. These states have been mentioned above in the Observer pattern. The state pattern is also useful in modeling the current state the restaurant may find itself - empty, full, closing. State can also be used to express the Customer’s satisfaction, the state would then influence the value of the tip provided.

**Strategy** - We can use this pattern to change the strategy used to prepare each meal. Different strategies must be used depending on the way a customer would like their meal - grilled, fried, braised, baked etc. It can also be used when preparing meals that may be required to be served hot or cold. Finally, the strategy pattern can also be used when seating customers. If there is a large group, tables may be needed to be grouped together, couples may be seated at smaller tables etc.

**Memento** - We can use the memento pattern to save a customer’s order after they have decided what they want from the menu and placed their order. This can be especially useful for popular online and takeaway orders that will not require much variation. Thus the order can be saved as state and be used to recommend it again to the customer.

**Prototype** - We can use the prototype pattern to create the same meal more than once whenever we have a group of people that order the same meal or when we have an individual order more than one serving of the same meal.

**Singleton** - The singleton pattern can be used to create a single instance of a single object. This will be extremely useful for the kitchen class. Since so many different classes interact with the kitchen it is important that there is one instance of it. This will help maintain order and synchronize all activities within the kitchen.

**Builder** - We can use this to create complex meals such as specials or when a customer requests specific instructions when preparing their meal such as degree of cooking (rare, medium-rare, well done, etc.). It is also useful for the plating stage. By using the builder pattern the head chef can assemble the dish according to the order that it corresponds to.

**Proxy** - We can use this for an online system so that whenever a client orders online they can interact with the system online before sending through their orders. This may be used to imitate delivery services.

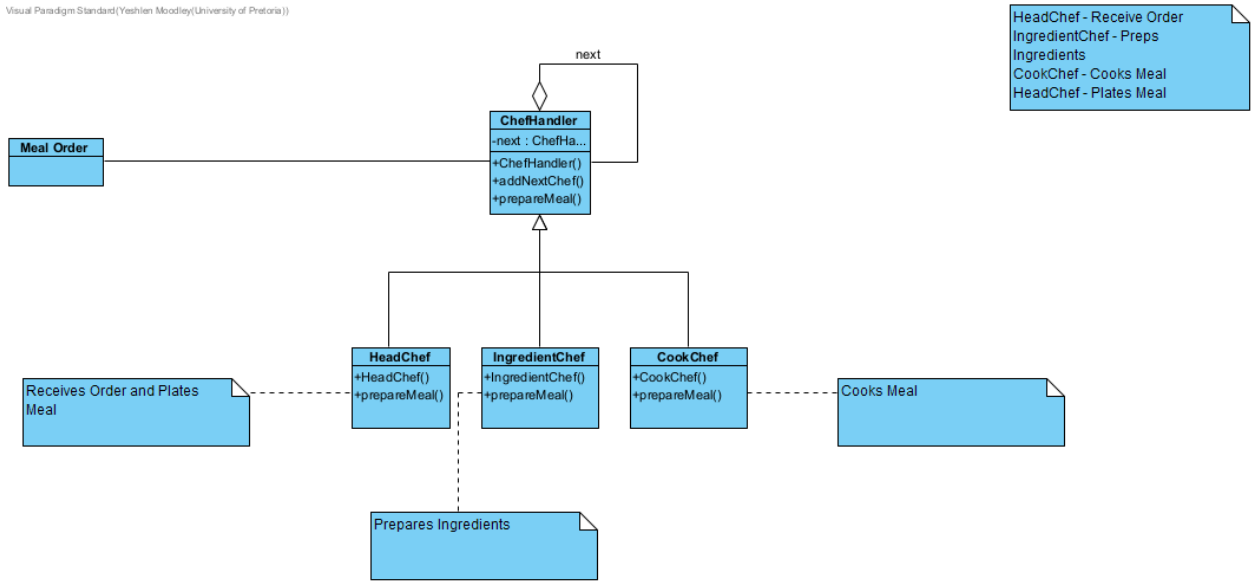
**Iterator** - We can use this to run through our order to see if all orders have been done and to run through tables to check if all tables have been served. This can be the manager, the waiter or the chef since they need to check in on the customers now and then. The use of the Iterator pattern with regard to the Floor; the monomers of the aggregate structure would be the tables, Floor would then be the aggregate, and the derived classes of Staff would be the concreteIterators.

**Command** - We can use command for making orders if ever a customer decides that the order they made is not what they want anymore they can change it to what they want by undoing what they had initially ordered before the order gets confirmed. The command pattern will also be very useful in passing the order to the kitchen. The command will be passed by a waiter to the head chef who will then push the different parts of the meal to the relevant queue (starter, main and dessert). This helps maintain order and keep track of an order.

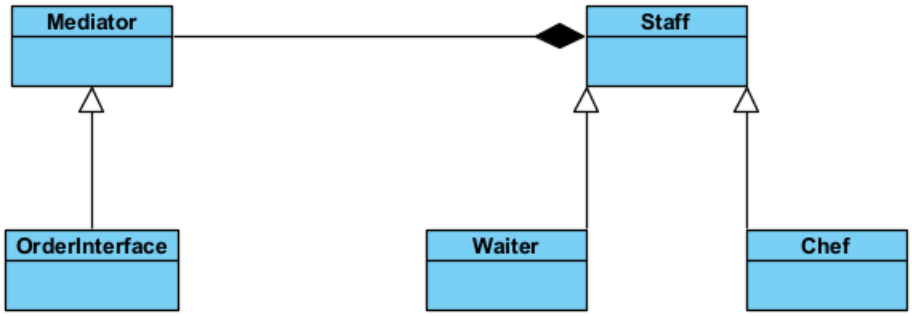
**Visitor -** In this case, the Table class would be the element of the Aggregate/ObjectStructure, and the Waiter class would represent the Visitor. The use of the Visitor design pattern will allow the Customer object, which would be seated at the Table, to notify the Waiter when they are ready to place their order. The Visitor would then be able to retrieve the Customer's order and forward it to the Kitchen. The accept function of the Visitor design pattern will also allow for the Waiter to "Subscribe" to a particular table.

**Decorator -** The decorator pattern can be used for the construction of the Customer's order, as this pattern would essentially create a linked list, which would allow for the Customer to append items onto their order, as well as instructions/preferences for the Chef.

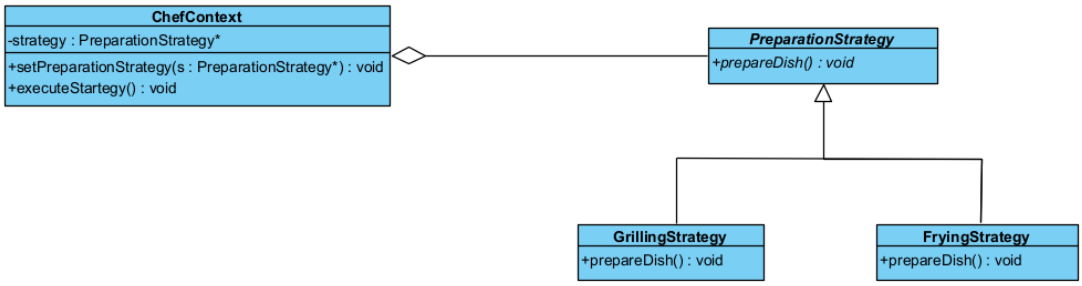
## Chain of Responsibility Pattern used during meal preparation:



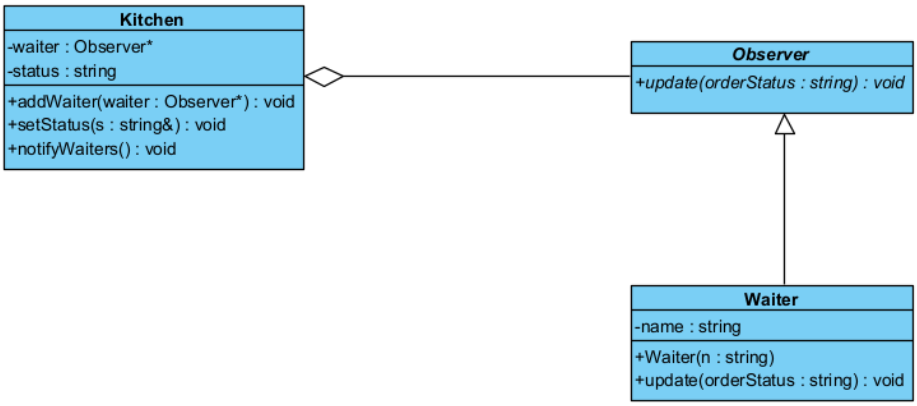

## Mediator Pattern as the waiter:



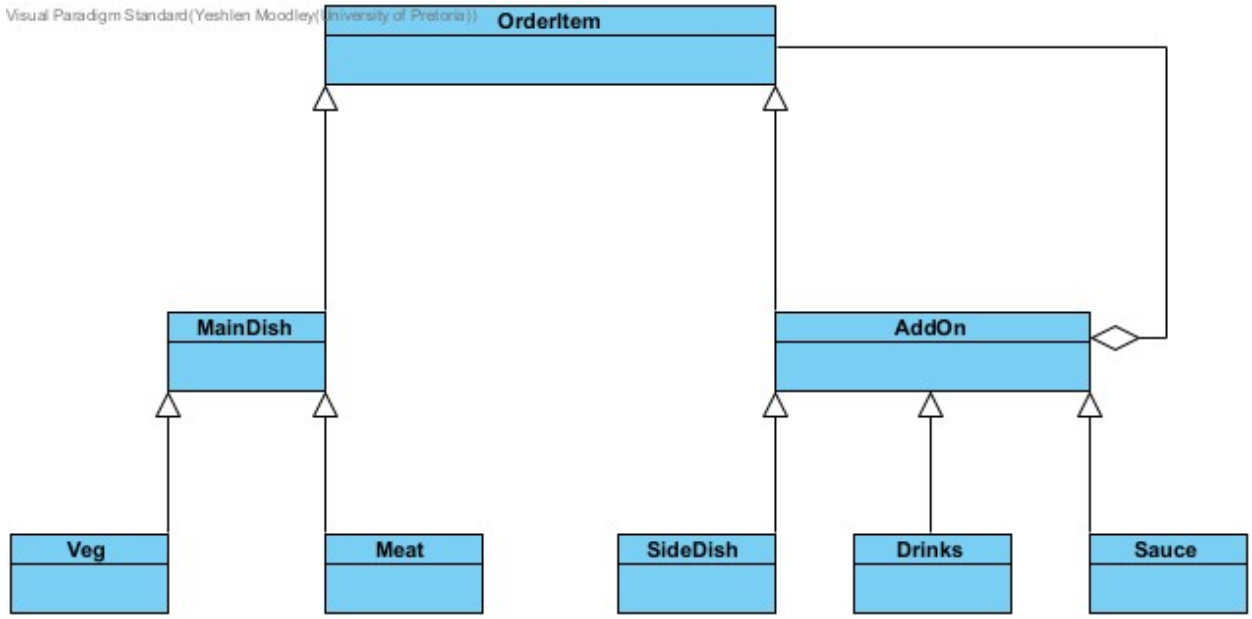
**Strategy Pattern that will determine how the meal will be prepared.**



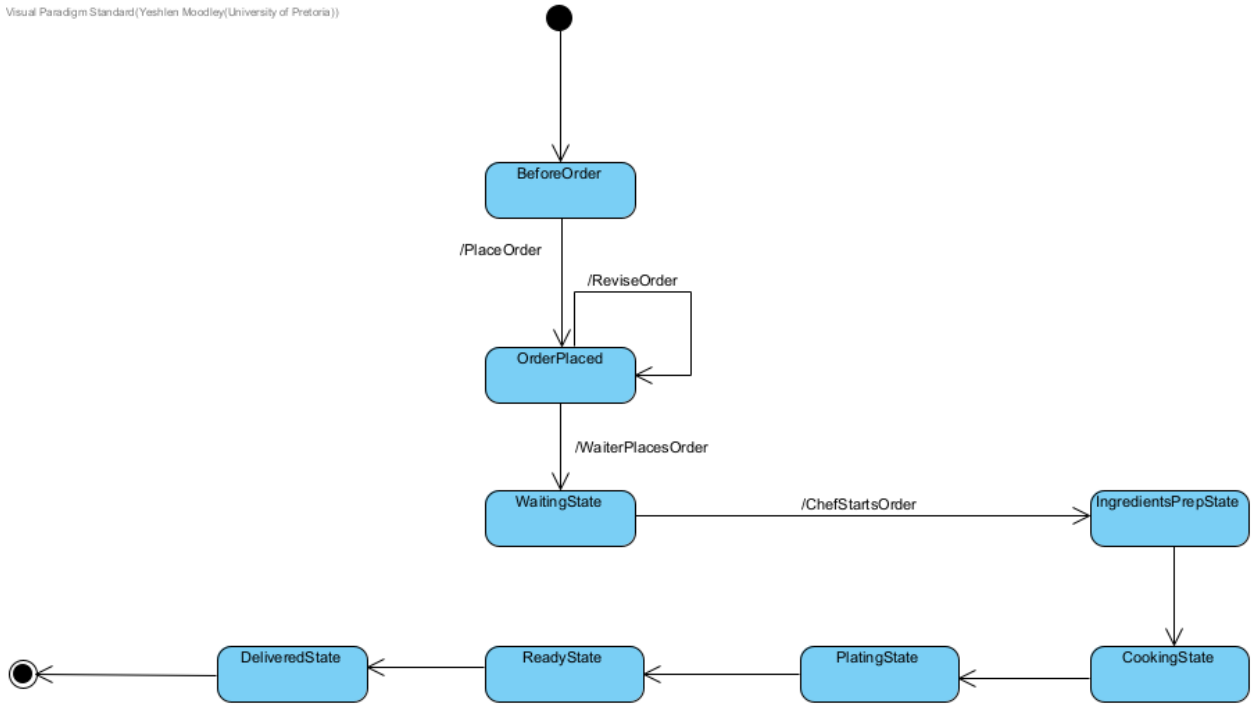

**Observer Pattern that illustrates relation between kitchen and waiter:**



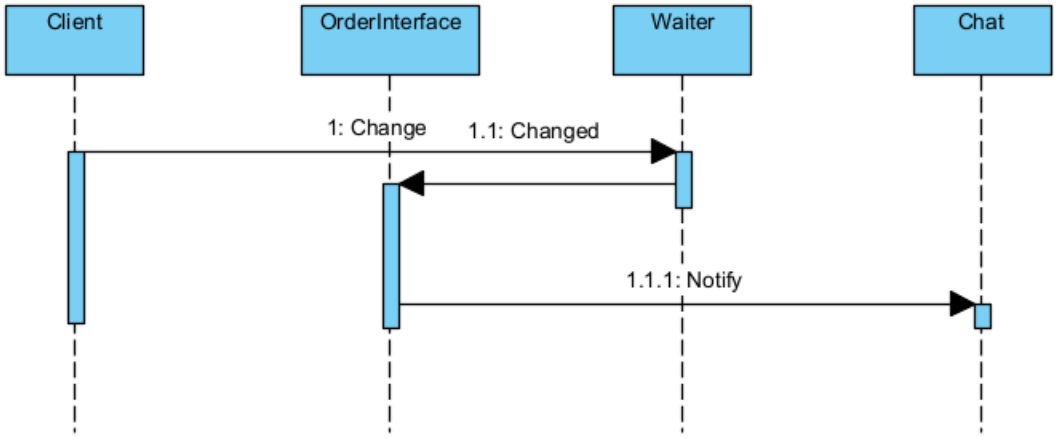
**Decorator Pattern showing how to add extras to order:**



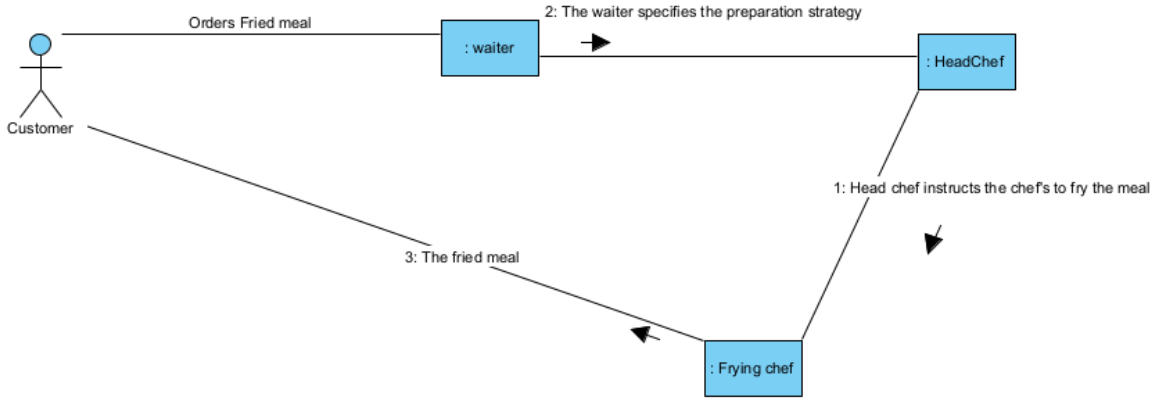

**State Diagram thats shows the states an order may find itself in:**



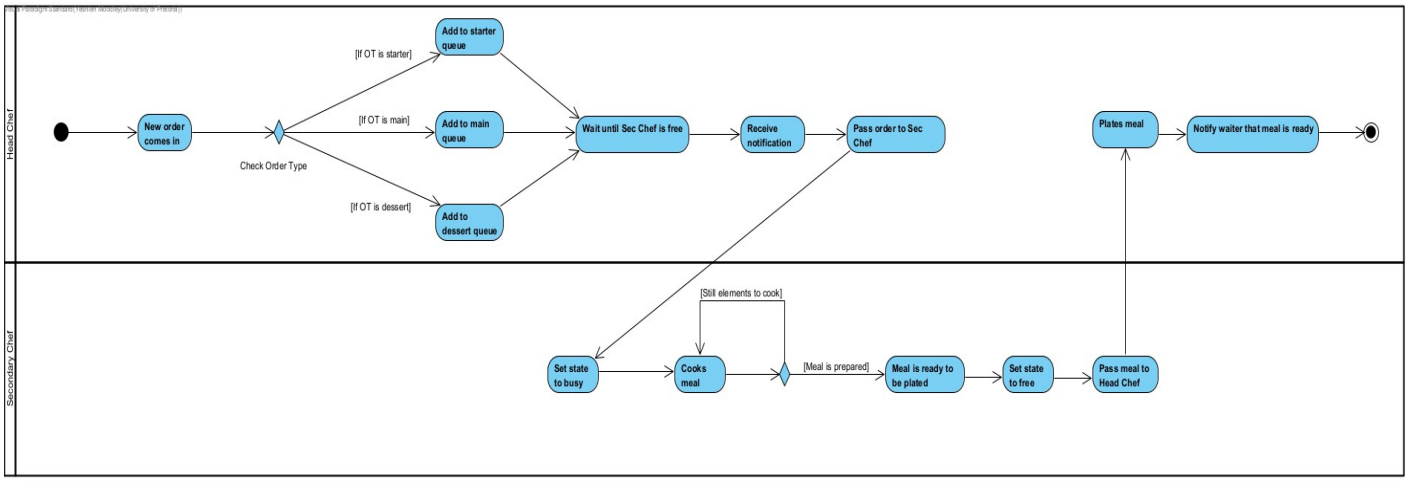
**Sequence Diagram of the Mediator as the Waiter:**



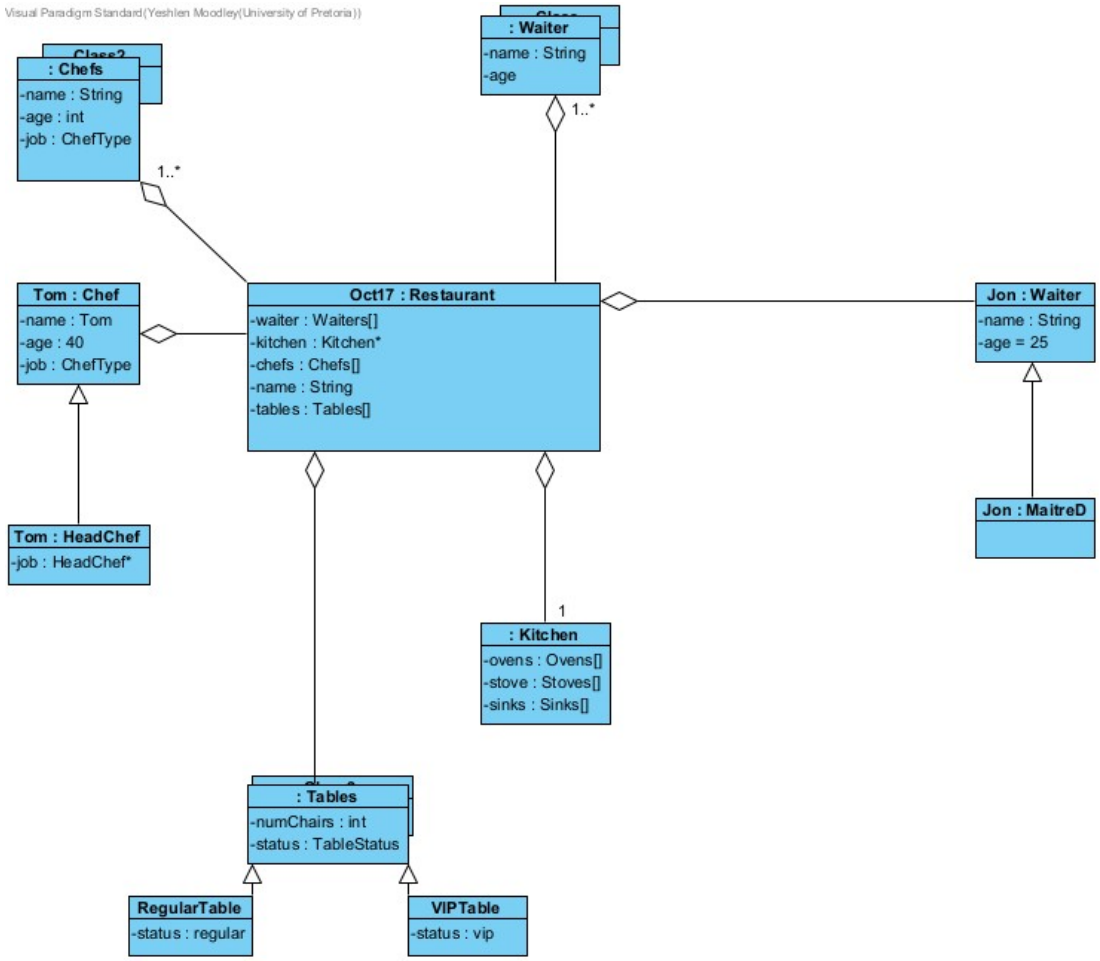

**Communication Diagram of how the Strategy is going to be used:**



## Activity Diagram of meal preparation process:



## Object Diagram of the Restaurant:



**Coding standards**:

The main goal of the coding phase is to code from a design document prepared after the design phase through a high-level language and then to unit test this code. It is good practice for programmers to maintain a well-defined and standard style of coding, called coding standards. Every programming language has its own defined standards that each programmer must follow when using it. It serves as a guideline for all members working on the project.

Before you even begin to write code you must create the design document. This document will contain various UML diagrams and other useful standards to enforce uniformity. UML is the general-purpose modeling language used to visualize the system that you need to develop. It is a graphical language that is standard to the software industry for specifying, visualizing, constructing , and documenting the artifacts of the software systems, as well as business modeling.
It benefits us as a team in the following ways:
- Simplifies complex software design
- Reduces thousands of words of explanation in a few graphical diagrams that may reduce time consumption to understand.
- It helps to view the entire system both in its entirety and individual parts.
- It becomes very much easy for the software programmer to implement the actual system once they have a clear picture of the problem.

**Naming standards :**

Code is written once but read multiple times by others in the project team or even from other teams. Readability is therefore important. Readability is nothing more than figuring out what the code does in as little time as possible. While there are well-established naming conventions, there's no single one that fits all scenarios. Each programming language recommends its own convention. Each project team or organization may define its own convention.
In programming, naming conventions are a set of rules for choosing the character sequence to be used for identifiers which denote variables, types, functions, and other entities in source code and documentation.
As the years went by programmers there were many common naming conventions amongst them but, there are four that have proved to be the most popular ones.

**Camel Case:**
In camel case, you start a name in lowercase. If the name has multiple words, the first letter of the second word and onwards will start with a capital letter.
Let's say you have the following functions: switchOn() and switchOff().

**Snake Case:**
Similar to camel case, you start the name in lowercase. If the name has multiple words, all words will be in lowercase and an underscore "_" is used to separate the words.
Let's say you have the following variables of snake case: first_name and last_name.

**Pascal Case:**
Unlike the previous examples, names in Pascal case start with a capital letter. In case of the names with multiple words, all words will begin with capital letters.
Let's say you have the following variables in Pascal case: FirstName and LastName.

**Kebab Case:**
Kebab case is similar to snake case, but you use a hyphen (-) instead of an underscore "_" to separate the words.
Let's say you have the following variables in kebab case: first-name and last-name.

All the naming conventions are based on what you are naming and the type of language you are working on.
Naming conventions should have meaningful and understandable variable names that clearly indicate their purpose.
We use the above mentioned naming conventions for local and global variables and functions but for constant variables we use all capital letters (COUNTER).
We avoid using digits in variable names.
The preferred naming convention for this project will be **camel case.**
Similar to variable names, function names should be simple and clearly indicate their purpose/use.

**Git standards :**

Git standards are a set of best practices and conventions that teams follow when using Git, a distributed version control system.
Whenever you initialize a project you need to add a README file, that will explain what the project that you are building is used for and also how one can use it, if they want to.

It is important to use meaningful commit messages and meaningful comments whenever you make changes to your code so that the whole team can see and understand why the changes had to be made.

It is also important to use meaning names for the branches that you create to work on a feature you are developing.

Git is used to store and track changes to the code you write as an individual or as a team. You first create a repository where you will be working at. Using git makes it easy to work in teams, having an easy-to-follow workflow that works for your organization or business is crucial to fast development.

There are several types of Git workflows that teams can use but just like naming conventions there are five commonly used git workflows by most programmers.

**Basic WorkFlow:** This workflow is easy. There is a single repository which will contain the master branch and each developer will clone the repo, work on it locally, commit the changes, and push it to the master branch for other developers to grab and use.

**Feature Branches Workflow:** This workflow uses Git's primary feature, branches. Each developer will create a new branch from the master branch to build and test it locally. When the feature is complete, merge the branch with the master branch and push it to the production server (stable build).

**Gitflow Workflow**: This workflow uses two parallel long-running branches, master and developer. The master branch is used for releases, while developer is where all completed and stable features ready for the next release reside. When you start working on a new feature, create a new feature branch from 'developer'. Once the feature has been built, completed and tested, return the code to the 'developer' branch. When it's ready to release, create a separate Release branch.

**Feature Branches and Merge Requests Workflow:** This workflow considers that all team members have different levels of expertise and authority. In this instance, merge requests and push permissions can be used to limit pushing to specific branches in the repository while maintaining complete control over the code.

**Forking Workflow**: In this workflow, when a developer needs to make changes to an open source project, they don't work on the repository directly. Instead, they fork it, thus making a duplicate of the repository. The developer is then free to work on the new functionality in whatever way they like.

**References**:


Naming Conventions (devopedia.org)
How to Choose the Git Workflow & Branching Model That's Right for Your Team (howtogeek.com)
Unified Modeling Language (UML) | Class Diagrams - GeeksforGeeks
Programming Naming Conventions – Camel, Snake, Kebab, and Pascal Case Explained (freecodecamp.org)
https://www.cs.up.ac.za/cs/lmarshall/TDP/TDP.html