Restaurant Application

**The Application**

For this assignment I have chosen a restaurant application to demonstrate the use of design patterns and the functionalities and features that the application provides using these design patterns.

The Restaurant application allows the author to order food and beverages through the app. The app is linked to a database that stores information of the individual who is using the application. It then uses this info retrieve details of the user during purchases orders, reviews etc. The app will also show the status of the customer’s order once it has been placed including the delivery time and location of the deliverer.

The following design patterns have been used to implement the aforementioned functionalities and more.

**Design Pattern and Functionality**

1. **Singleton Pattern**

The Singleton design pattern is a software design pattern that restricts the instantiation of a class to one "single" instance. This is useful when exactly one object is needed to coordinate actions across the system.

In this application, the Singleton pattern is used to assign a single instance, of a database connection of the app to the database, to a global variable. Everytime the app is required to send a request from the database, all processes take place using this one instance of the variable.

**2. Factory Pattern**

The factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.

In our scenario, the factory method is used to create an instance of a beverage. The kinds of beverages available on the app are, Coffee, Tea, Juice, Shake, Soda and Water. There is one abstract class which has been derived in to the aforementioned drink classes. The factory method allows the instantiating of these drinks to be the responsibility of one class which uses ENUMS to determine what child class to derive the object from. Interestingly enough, the factory method also uses the Singleton pattern as all objects are derived from a single instance of the DrinksFactory object.

**3. Builder Pattern**

The Builder is a design pattern designed to provide a flexible solution to various object creation problems in object-oriented programming. The intent of the Builder design pattern is to separate the construction of a complex object from its representation.

There is one functionality of the app which is accessible by selected users. This functionality ‘Build a shop’ allows the high-level user, for example: the owner, to build shops in the application which then get sent through the app to the respective department who bring the shop idea into fruition. Since the shop can be complex but has the same basic building block as every shop, we use the builder class. This allows the user to add **only** the functionalities that he wants to his shop. For example: ‘Restaurant shopLipscani = ShopBuilder().addWoodenFlooring().addLights().addFoodServce().addDrinkService().addParking().build();

**4. Decorator Pattern**

In object-oriented programming, the decorator pattern is a design pattern that allows behavior to be added to an individual object, dynamically, without affecting the behavior of other objects from the same class.

One of the basic functionalities of the app is for customer’s to be able to order a dish and then add variations to it, like extra toppings in case of food or adding items like lemon, whipped cream, sweeteners in beverages to enhance the taste. These toppings cost extra on top of the basic price of the product and so add to the total cost of the product. As the user chooses toppings, the extra cost gets added to the final price of the food or beverage object.

**5. Adaptor Pattern**

The adapter pattern is a software design pattern that allows the interface of an existing class to be used as another interface. It is often used to make existing classes work with others without modifying their source code.

In our current application, the app uses different social media APIs to update the homepage of the app with relevant information regarding popularity of the brand, likes on posts, latest social updates etc. The adaptor class is used to allow the app to interact with the different APIs by defining a common interface which contains the different methods performable. Each API class is then individually tweaked to show the relevant information. The app uses the adaptor class interact with the Social APIs.

**6. Facade Pattern**

The facade pattern (also spelled façade) is a software-design pattern commonly used in object-oriented programming. Analogous to a facade in architecture, a facade is an object that serves as a front-facing interface masking more complex underlying or structural code. A facade can

improve the readability and usability of a software library by masking interaction with more complex components behind a single (and often simplified) API.

In our application, when the user presses the ‘Confirm Order’ button, it executes the Order.confirm() function which in turn executes a series of methods: Order.setDeliveryLocation();Order.setOrderItems();Order.setPaymentMethod()Order.generateReceipt();Order.deliver();

The methods mentioned above are executed in order and once all methods are executed, the Order.confirm() is successfully completed.

**7. Observer Pattern**

The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.

In our application, the observer pattern is used to implement the Order status component. Everytime the order reaches the next stage, the user is notified of the update on his device. As the order moves from being registered to being delivered, the different stages it enters and exits trigger handlers that then send a notification to the client.

**8. State Pattern**

The state pattern is a behavioral software design pattern that allows an object to alter its behavior when its internal state changes. This pattern is close to the concept of finite-state machines. The state pattern can be interpreted as a strategy pattern, which is able to switch a strategy through invocations of methods defined in the pattern's interface.

In our application, the state pattern is used to decide the contents of the menu of the restaurant during the different times of the day. The restaurant has a seperate menu for breakfast, lunch, dinner and late-nights. The behaviour of the Restaurant changes due to a change in the internal state of the application.

**9. Command Pattern**

In object-oriented programming, the command pattern is a behavioral design pattern in which an object is used to encapsulate all information needed to perform an action or trigger an event at a later time. This information includes the method name, the object that owns the method and values for the method parameters.

In our application the benefit of the Command pattern to be able to undo commands is the one we focus on. Using the command pattern, the application allows the user to undo an order.