UNIVERSITY OF TECHNOLOGY, JAMAICA

School of Computing and Information Technology

Project Report

Group Members: De-Andrea Malcolm (1608204)

Jordane Plummer (1403184)

Tasi-Ann Mighty (1704280)

Date: May 2, 2020

Module: Advanced Programming (CIT3009)

Lecturer: Mr. Gilroy Gordon

Table of Contents

Introduction

Cups is a local coffee shop that provides a relaxing getaway in the middle of the city for the disabled community. They are also a wonderful example of a Social Enterprise Boost Initiative similar to DeafCan coffee. Kat, the manager, has been encouraged by her mentor to establish another store at 95 Moolean Avenue in the heart of Montego Bay. Kat would like to encourage an empowering environment through self-service.

An Artificial Intelligent system through Computer Vision and Speech Processing was created to accomplish this. The touch-screen self-service kiosk allows customers to order their favourite treats and verify using their Digital Id. This is a web-based system developed with the ASP.NET Framework 4.7.2 using the C# programming language. The Bootstrap Framework was also used to enhance the User Interface and User Experience. This framework uses the languages HTML, CSS and JavaScript. The Entity Data Model Designer component of Microsoft Visual Studio 2019 was used to develop data model files which stores the schema for SQL the Server database. The ASP.NET Web API Framework is used to build RESTFUL API which is used to create a client server architecture.

Visual Studio 2019 was the IDE used to develop the web application and no major configurations were done.

Research Concepts

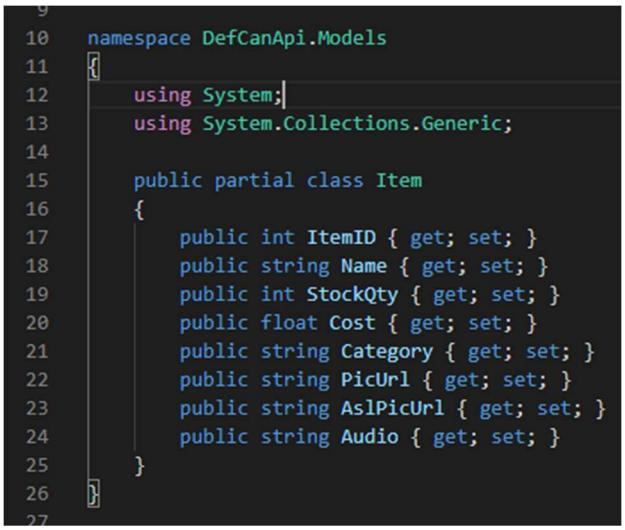
S.O.L.I.D. Principles

The acronym S.O.L.I.D. stands for five (5) design principles for Object Oriented Programming. These include Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation and Dependency Inversion principle. These 5 principles were introduced by Robert C. Martin (Uncle Bob), in his 2000 paper Design Principles and Design Patterns (LH, 2019).

- *S — Single Responsibility Principle*

This principle states that a class or module should have one, and only one, responsibility. According to Janssen (2020), a benefit of this principle is that the code is easier to understand which makes implementing of changes or expansion of the program in the future easier.

Here is a snippet of the Models directory of the program which shows a class "Item" which only has one responsibility; it identifies attributes of an Item.



- *O — Open/Closed Principle*

The open/closed principle states that objects or entities (classes, modules, functions) should be open for extensions, but closed for modification. In other words, code should be written in such a way that extending functionality can be done without modifying the existing code. This can be done through inheritance; inheriting the existing class and creating a new class to carry out new function. A benefit of implementing this principle is

that prevents situations in which a change to one of your classes also requires you to adapt all depending classes (Janssen, 2020).

- *L — Liskov's Substitution Principle*

This principle is often described mathematically as, "Let q(x) be a property provable about objects of x of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T." (Oloruntoba, 2015). In simpler terms, where ever a parent class (superclass) object is requested, a child class (subclass) can be a substitute. This can also describe polymorphism. This is beneficial because it helps to maintain the open/close principle when expanding your code in the future.

```
4   namespace DefCanApi
5   {
6       public class FilterConfig
7       {
8           public static void RegisterGlobalFilters(GlobalFilterCollection filters)
9           {
10              filters.Add(new HandleErrorAttribute());
11          }
12      }
13  }
14
```

In line ten (10) of this snippet is an example of the Liskov's Substitution Principle. Even though a subclass object was not created, the constructor of the subclass was called as a parameter in the function of a superclass. This method of writing the code performs the same task with less lines of code.

- *I — Interface Segregation Principle*

According to Janssen (2020), this principle, in the exact words of Robert C. Martin, states "Clients should not be forced to depend upon interfaces that they do not use". So, separate and inherit interfaces so that what is needed is the only thing that is implemented. Classes which implement interfaces are forced to override all the methods of that interface. Therefore, if a class implements an interface and is forced to implement methods that are not needed or will not be used in that class, that makes the code redundant. Interface segregation says, create new

interfaces to separate the methods of the interface according to what is required and if needed use inheritance to join them. This principle is beneficial because it helps to reduce redundancy by not making a class inherit unrelated methods, as well as it makes expanding of the program easier. It also helps the single responsibility principle which makes the code easier to understand.

- *D - Dependency Inversion Principle*

This principle states that high-level modules should not depend on low-level modules; both should depend on abstractions. Also, abstractions should not depend on details; details should depend on abstractions. High-level modules should be easily reusable and unaffected by changes in low-level modules. For this to be possible, abstraction is introduced which prevents coupling the high-level and low-level modules. Benefits of this principle are that it helps to enforce the Liskov's substitution principle and encourages code to be arranged in such a way that expansion is possible and easier in the future.

Design Patterns

Design patterns are solutions to commonly occurring problems in software design. It is a description to solve a recurring design problem in your code.

Repository Pattern

This pattern implements a collection that can be accessed through an abstraction. It also supports the objective of achieving a clean separation and one-way dependency between the domain and data mapping layers (Kudchikar, 2019).

Model- View-Controller

This design pattern separates common application concerns. Model has to do with how we access our resources or how an object carries data. It also has logic to update the controller if its data changes. View has to do with what the user interacts with or the visualization of data. Finally, the controller facilitates interaction between the model and view. It controls the data flow into the model and updates the view whenever data changes (Tutorials Point, 2020).

Singleton Pattern

This design pattern allows us to maintain only one instance of a resource to be utilized by multiple objects. It helps to ensure that a class has only one instance, while providing a global access point to this instance. This pattern falls under the category of a creational pattern. Benefits of this pattern are that you are sure that a class has only a one instance, you can gain a global access point to that instance and the object is initialized only when it is requested for the first time (Shvets, 2020).

Factory Pattern

This design pattern manages the creation of objects and delegates the creation of objects to another class. Also, it falls under the category of a creational pattern. A benefit is that this pattern helps to enforce the open/closed principle.

Tools

Code Documentation Generation Tool

This tool automates the process of generating documentation from code. It analyzes the code based on the rules of the language and the developer documentation to generate diagrams, descriptions and other documentation artefacts. (Doxygen)

Code Generation/Scaffolding Tool

Code generation is that creation of a basic template of components of the program such as classes, functions etc, that can be used to create the application. Scaffolding is a technique supported by some model–view–controller frameworks, in which the programmer can specify how the application database may be used. Code generation was done throughout this project as the ASP.NET Framework and the Visual Studio 2019 supports. No specific tool was used.

Source Control Management Tool

Source control (or version control) is the practice of tracking and managing changes to code. Source control management (SCM) systems provide a running history of code development and help to resolve conflicts when merging contributions from multiple sources (Amazon Web Services Inc., 2020). Git, which is an open-source distributed source code management system, was used for this project.

Package Management Tool

A package manager/package management tool is responsible for downloading, installing, updating, and configuring of software in your system. NuGet is the package manager used in this project.

Unit Testing and Test Automation

Unit testing is the process of software testing which comprises of individual units or components being tested (Guru99, 2020). In visual studio, individual Views can be run within the code and utilized the model and controllers which they are attached to. All the features were separated as such within different views. Each view was ran and test data entered to ensure it was carrying out its required function. If it placed data into the database, the database was checked via Microsoft SQL Server to see if it was logged successfully and the same was done if data was drawn from the database to ensure the functionality of the individual parts of the code were doing their required functions. This process is very beneficial as it helps with ensure each individual components is working and it helps to identify which is not.

Continuous Integration

Continuous Integration is a software development practice where members of a team integrate their work. The continuous integration server used is GitHub. The benefit of this practice was so that everyone was able to work and make necessary changes on the project at once without having to worry who has the latest version of the code.

References

Amazon Web Services Inc. (2020). *What is Source Control?* Retrieved from Amazon Web Services: https://aws.amazon.com/devops/source-control/

Guru99. (2020). *Unit Testing Tutorial: What is, Types, Tools, EXAMPLE*. Retrieved from guru99.com: https://www.guru99.com/unit-testing-guide.html

Janssen, T. (2020, April 1). *SOLID Design Principles Explained: The Single Responsibility Principle*. Retrieved from Stackify: https://stackify.com/solid-design-principles/

Kudchikar, S. (2019, June 30). *Repository Pattern C#*. Retrieved from codewithshadman.com: https://codewithshadman.com/repository-pattern-csharp/

LH, S. (2019, January 1). *SOLID Principles: Explanation and examples*. Retrieved from ITNEXT: https://itnext.io/solid-principles-explanation-and-examples-715b975dcad4

Oloruntoba, S. (2015, March 18). *S.O.L.I.D: The First 5 Principles of Object Oriented Design*. Retrieved from scotch: https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

Shvets, A. (2020). *What's a design pattern?* Retrieved from refactoring.guru: https://refactoring.guru/design-patterns/what-is-pattern

Tutorials Point. (2020). *ASP.NET MVC - NuGet Package Management*. Retrieved from tutorialspoint.com: https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_nuget_package_management.htm

Tutorials Point. (2020). *Design Patterns - MVC Pattern*. Retrieved from tutorialspoint.com: https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm