MVC Architecture

[BOROUGH] OF STRUCTURES

By: Luke Breyer

Terminology

- MVC - Model, View, Controller. Refers to an application design model used to separate the application's elements into the three aforementioned interconnected components.
- Model – The central component of the application. It handles the data and internal methods and is composed of three main sections: Domain, Infrastructure and Data. It receives updates via the Controller.
    - Domain – The section of the Model that houses the application's object classes and their related Service classes. It acts as the primary access point for the Controller.
    - Infrastructure – The section of the Model that houses behavior pertaining to the overall structure of the application's internals or to the handling of the Model objects.
    - Data – The section of the Model that houses behavior that handles the application's external elements along with any constants or unities required by any component.
- View – The front-end component of the application. It displays to the user the user interface, and through this takes in any input, actions, or requests from and passes them to the Controller.
- Controller – The connecting component of the application. It takes in user behavior from the View and then updates the Model accordingly. It then returns from the Model in order to update the View.

MVC Architecture Overview

- MVC Framework is an organizational structure within programming focused on compartmentalizing aspects of an application into three main components:
    - M – Model
    - V – View
    - C – Controller
- Each component is built to handle specific development aspects of an application with the goal of the framework being to improve its overall organization and maintainability.
- Applications using the MVC framework are commonly subdivided into two main projects: the Navigator/Application and the Core. The former houses the View and Controller while the latter houses the elements that make up the Model and acts as a class library for the application.

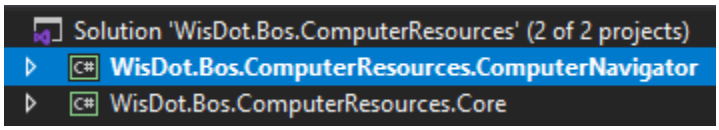

Figure 1: Navigator and Core

- Separating an application's internal elements using this architecture affords the development process and number of benefits including compartmentalization of modifications, parallel development support and improvements to both maintainability and scalability.

Model (M)

- The Model component handles the data and data structures of an application. This primarily includes the objects that the code will act upon along with their related logic. It represents the primary central internal component of the application, and it exists independently of the View or external UI component taking in updates, requests and commands from the Controller.
- The Model component for a given application is subdivided into three primary groupings, Domain, Data and Infrastructure, that work interconnectedly to make up an application's internals.

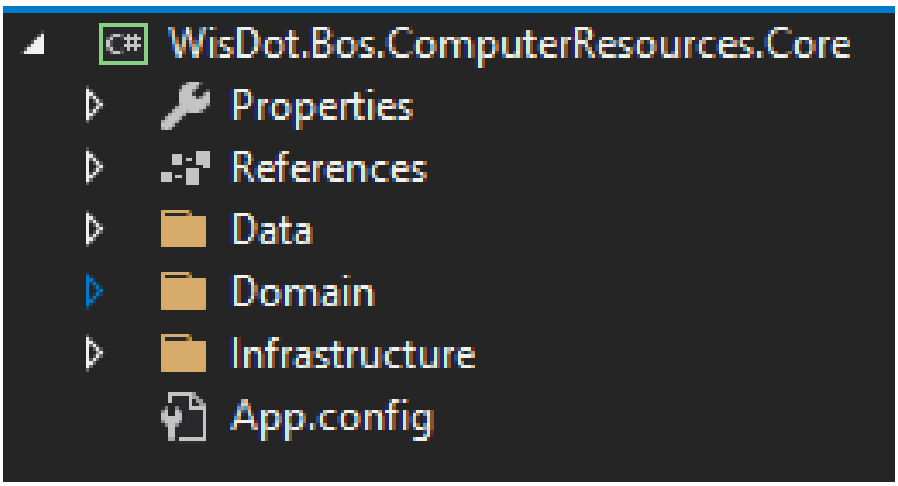

Figure 2: Data, Domain, and Infrastructure

Domain

- The Domain component of a Model primarily houses the actual Models of the application along with their respective Service components given that the Models have related behavior or methodology in the program's context.

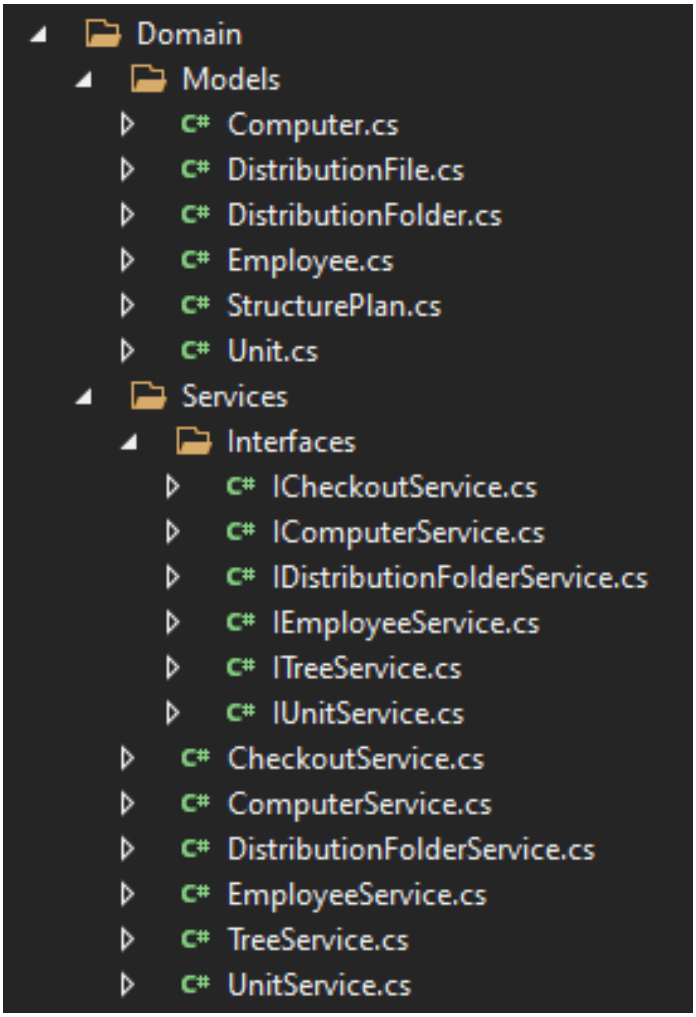

Figure 3: Domain

Models

- An application's Models represent the object classes that will form its internal structures. Each Model object class is acted upon by the Service classes that house the Models' related methodology.
    - The Models should each house their internal fields at appropriate access levels within context along with any constructors it should need.
    - Should the model have any fields that initialize to 'null' (such as strings), in good practice, the Model object class should also contain a private 'Initialize()' method that initializes these fields to appropriate values. This is to avoid potential nullPointerExceptions and should be called at the beginning of each constructor.

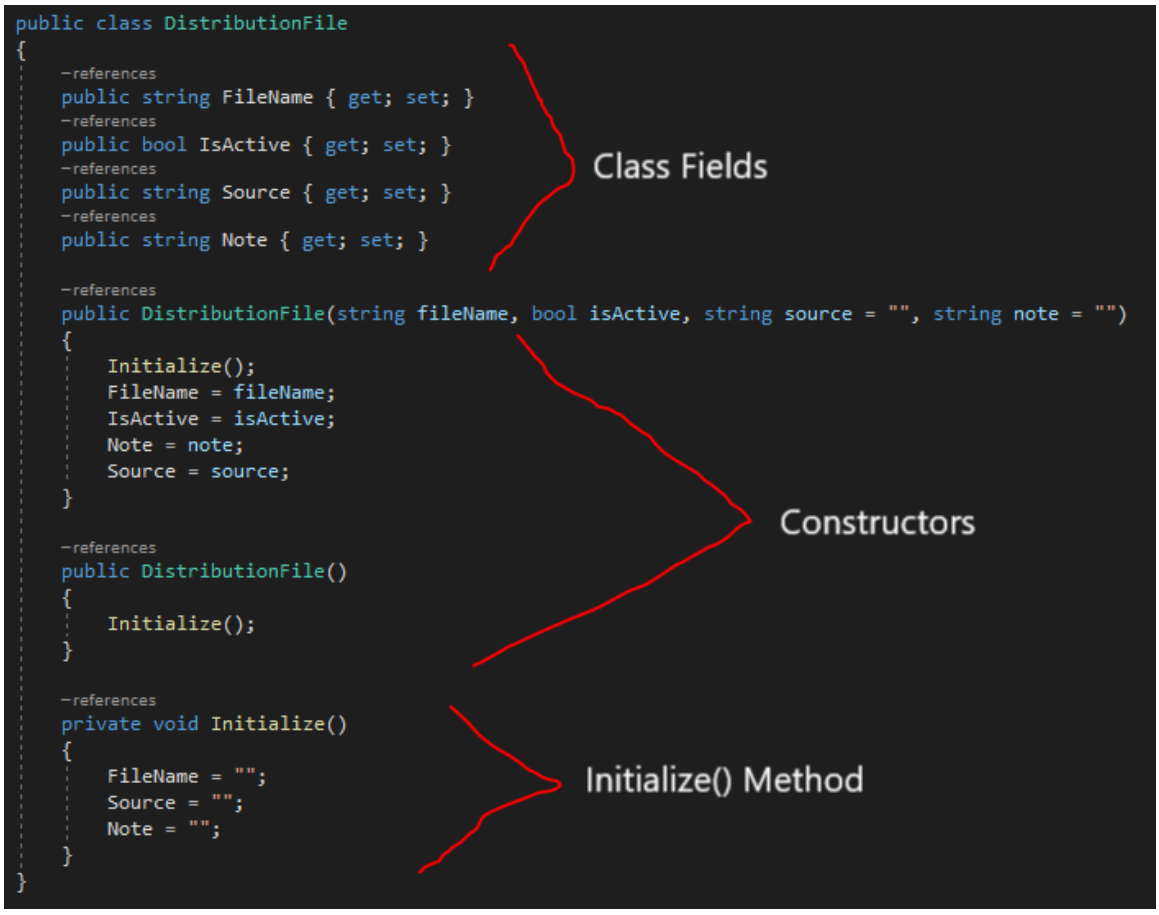

Figure 4: Model Class

Services

- These Model objects that contain related methodology should have an attached Service class by the naming convention: *ModelName*Service. This class houses any methods pertaining to a given Model object and should act as the primary access point for the Controller into the Model component.
    - Methodology within a Service class should be delineated into one of three categories in order to be handled appropriately by one of the object's Query or Repository classes or by the Service itself.
    - If the behavior refers to the application's external resources through File I/O, importing, exporting or any other manner of access to the application's data, the method belongs in the Model's Query class.
    - If the behavior refers to the infrastructure of Model object, its handling or its overall usage within the application's context, the method belongs in the Model's Repository class.
    - If the method refers exclusively to the Model and does not fit under either class, the method can be handled within the service itself.
    - The Service class should be prototyped by an Interface and contain private references to its Repository and Query in order to call upon them during method handling.

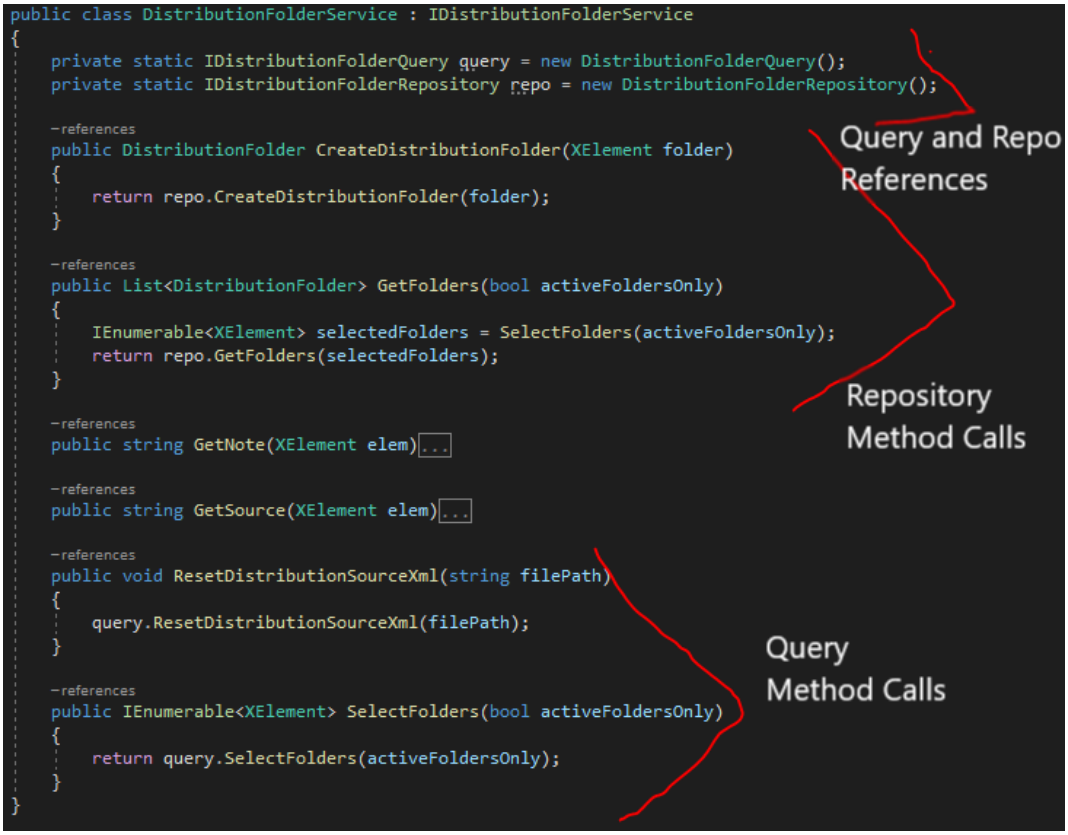

Figure 5: Service Class

Infrastructure

- The Infrastructure component of a Model houses the classes and methodology pertaining to the overall structure of the application's internals. This will primarily take the form of the Repositories for the appropriate Model class objects, but it can also often contain general or utility classes that refer to the application's internals but are not associated with a particular Model class.

Repositories

- The repository classes contain the behavior and methodology relating to the Model object classes that interact with the overall structure of the application's internals or with the handling of the Model objects. It should follow the naming convention *ModelName*Repository.
    - The Repository should be referenced solely by the related object's service and should not be accessed by the View or the Controller. It serves to support the Service class and should consist primarily of such methodology.
    - Repositories should also be prototyped by interfaces located within the Infrastructure's respective Interfaces folder.

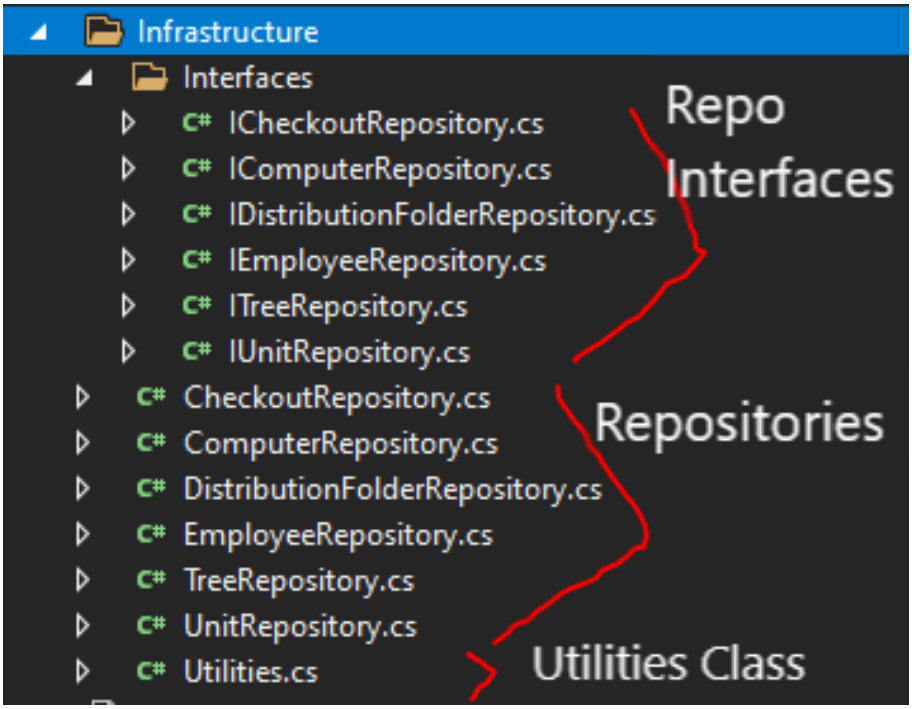

Figure 6: Infrastructure

Data

- As the name suggests, the Data component of the Model houses the data and data management for the internals of the application. Classes and methodology related to the application's externals, File I/O, data management or any constants or data abstracted from another component belong in this grouping. Commonly, the Data component will primarily contain the Query classes pertaining to a given Model object.
- The data component will also often contain constants for the application or from other components along with defined datatypes such as enumerated types or constantly defined class objects.

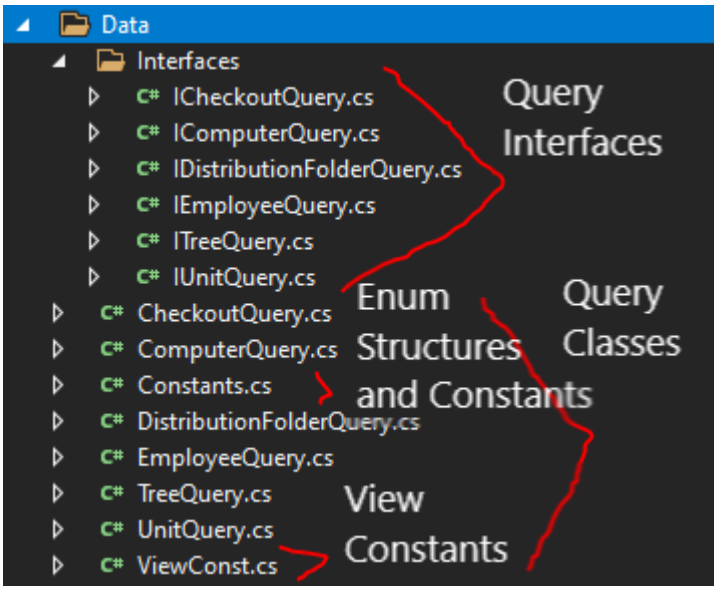

Figure 7: Data

Queries

- The Query classes contain the behavior and methodology for a given Model object class relating to the interaction between that object's Service and any necessary externals. It should follow the naming convention *ModelName*Query.
    - Query classes provide the link between the data and externals of an application and any Service or acting class that should need to interact with them. They serve to organize this access point coordinate any required I/O.
    - The Model object class related Queries should also be prototyped within the Data component's Interfaces folder.
    - When accessing externals for an application, any explicit file paths or access paths should be stored within the config file for the active/startup project. From there, they can be read from this file into the Data component to be used by the Query.

```
<add key="BOSCUBEMAP" value="\\mad00fph\n4public\Bos\struct_devel\BOSNavigator\CubeMap\Plan.pdf" />
<add key="CHECKOUTLOG" value="\\mad00fph\n4public\Bos\struct_devel\BOSNavigator\checkout.csv" />
<add key="SIFileName" value="C:\Program Files (x86)\Adobe\Acrobat DC\Acrobat\Acrobat.exe" />
<add key="ComputersXmlFilePath" value="C:\Users\DOTLDB\source\repos\WisDot.Bos.ComputerResources\BOSComputers.xml" />
<!--<add key="ComputersXmlFilePath" value="\\mad00fph\n4public\bos\struct_devel\BosNavigator\BOSComputers.xml" />-->
<add key="DistributionSourceXmlFilePath" value="C:\Users\DOTLDB\source\repos\WisDot.Bos.ComputerResources\distribution-sources.xml" />
<!--<add key="DistributionSourceXmlFilePath" value="\\mad00fp1\w4bhs\struct_devel\distribution-sources.xml" />-->
<add key="EmployeesXmlFilePath" value="C:\Users\DOTLDB\source\repos\WisDot.Bos.ComputerResources\bosowners.xml" />
<!--<add key="EmployeesXmlFilePath" value="\\mad00fph\n4public\bos\struct_devel\BosNavigator\bosowners.xml" />-->
<add key="UnitsXmlFilePath" value="C:\Users\DOTLDB\source\repos\WisDot.Bos.ComputerResources\boslocations.xml" />
<!--<add key="UnitsXmlFilePath" value="\\mad00fph\n4public\bos\struct_devel\BosNavigator\boslocations.xml" />-->
<add key="BridgePlansDirectory" value="\\dotstrc\04bridge\" />
<add key="AdminDoc" value="\\mad00fph\n4public\Bos\struct_devel\BOSNavigator\UserRights.xml" />
```

Figure 8: Application Config File

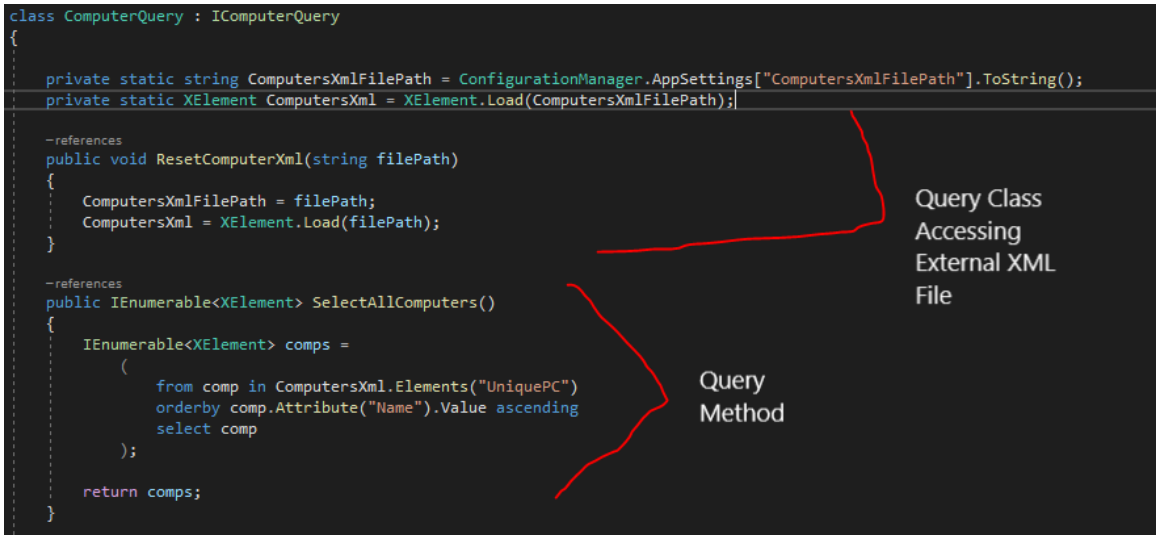

Figure 9: Query Accessing External File

View (V)

- The view represents the front-facing end of the application. It displays to the user the interfaceable elements and responds to their actions, requests and behavior. The view takes in input from the user and passes that information off to the Controller in order to handle the task. The View itself does not handle tasks itself but rather passes the requests off to the Controller which in turn interprets and then calls upon the appropriate Service within the Model component.

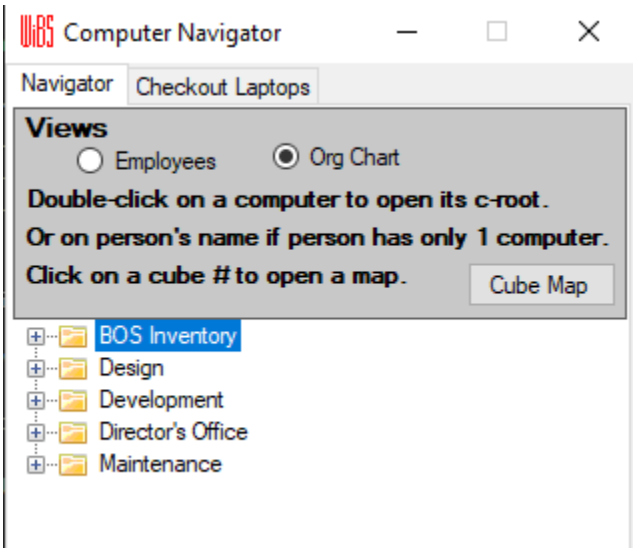

Figure 10: Application View

- Behavior and methodology within the View class should be abstracted out to the Model component through Service classes that do not necessarily pertain to a specific Model object class. The only methodology that should be defined explicitly within the View class itself is behavior that affects only elements of the view with no external logic. An example would be clearing a display element or adding items to a grid.
    - Statics or constants used by the View class should also be abstracted out to the Data section of the Model component.
- The View should also be defined by an interface class within its project. Within this interface, all methods that are not defined as user-element interactions (such as on X button pressed or on check) should be prototyped within this interface class.

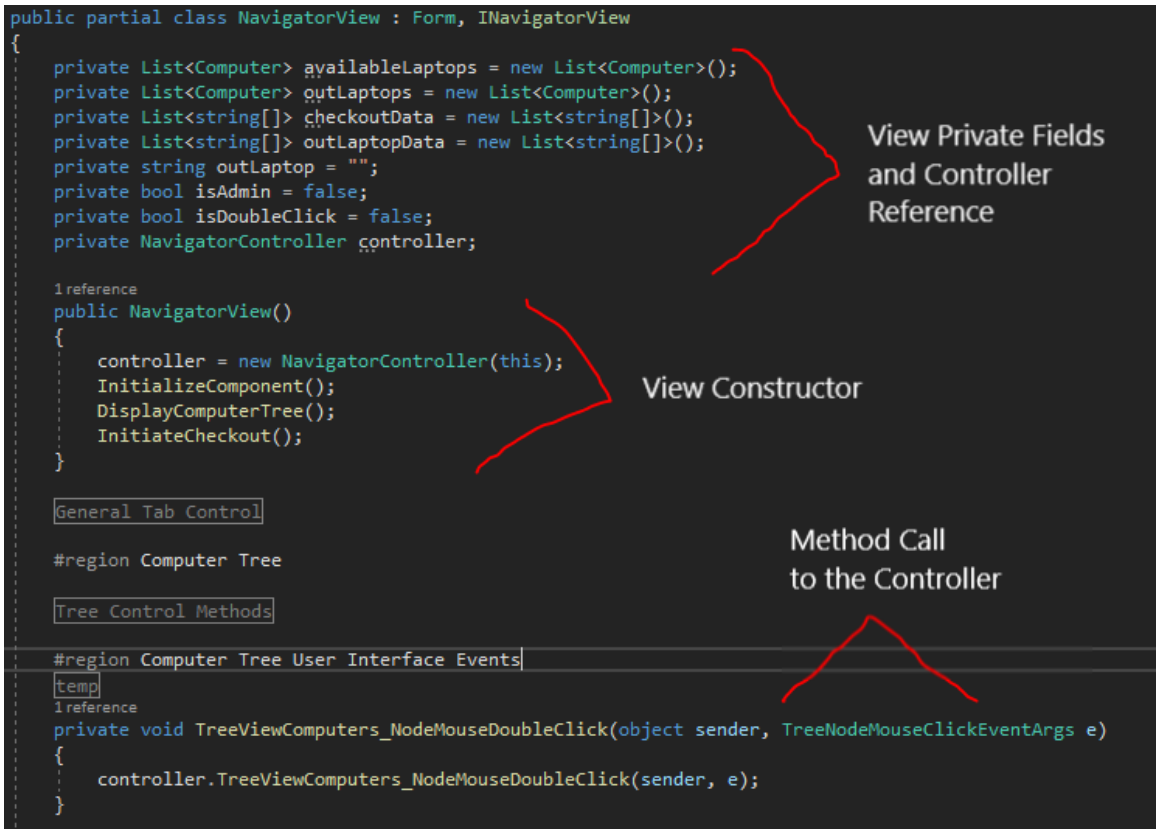

Figure 11: View Class

Controller (C)

- The Controller exists as the connection between the Model and the View. As users interact with the View, the View passes this information on to the Controller which, in turn, calls upon the appropriate Service within the Model component. The Controller updates the Model and accesses from it the necessary data. It then returns this information to the View who then presents that information to the User.
- The Controller should only be able to interact with the Service classes of the Model. The Services should act as the primary access point for the application barring a few exceptions such as accessing constants, structures or utilities from the Data or Infrastructure or the Model component.

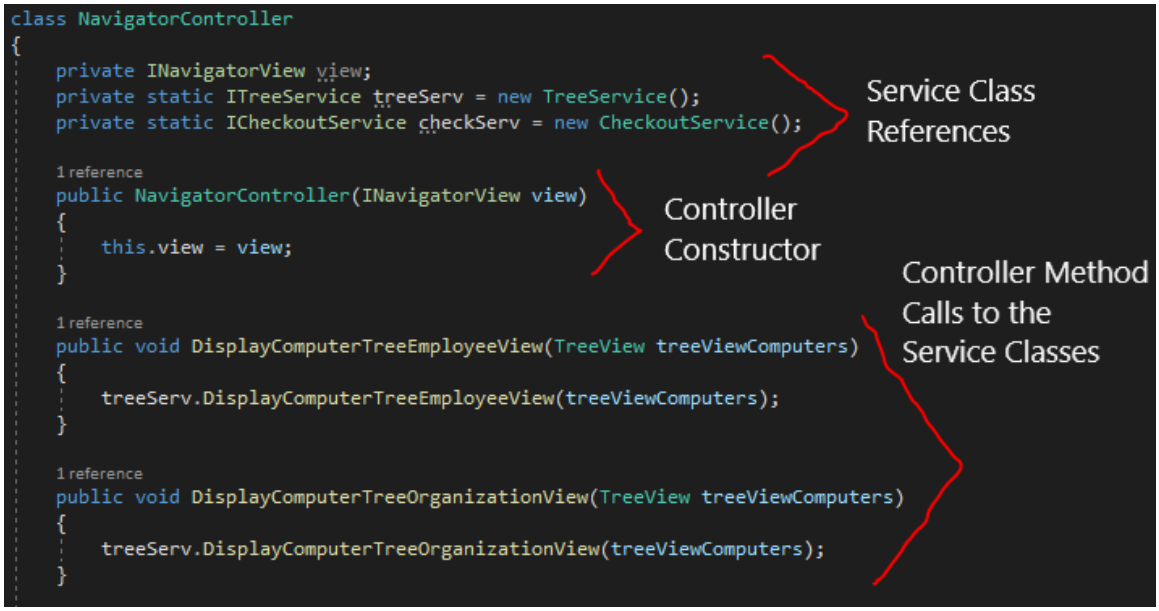

Figure 12: Controller Class

Real World Example

- To help conceptualize the MVC framework, one can think of a restaurant. The customers exist as the users of this application, and on the front end the menus represent the View. The menus present the information of the application to the user, and in turn they interact by ordering food through the waiter who acts as the Controller. The Controller is the bridge between the View and the Model, and they take the order as information and pass it on to the kitchen. The kitchen, acting as the model, receives the user request from the controller and returns their order to the waiter or the controller. The waiter then takes the food back to the customers and finally presents it as requested.

Summary

- The MVC Framework represents an efficient organizational structure centered around the abstraction and compartmentalization of application behavior into three main components: Model, View and Controller.
    - The Model represents the data section of the application housing the internal structure in three groupings.
        - The Domain houses the actual Model objects along with their respective services. This acts as the access point to the Model component.
        - The Data houses the Queries that interface with external elements along with any internal data or constants.
        - Finally, the Infrastructure houses the Repositories that govern the overall structure and internal behavior.
    - The View is the front-facing component of the application. It displays the elements and data and then interacts with the user to take in their actions and requests.
    - The Controller links together the Model and the View. It takes in requests from the View and updates the Model accordingly. In turn, it passes back returned information from the Model in order to update the View.

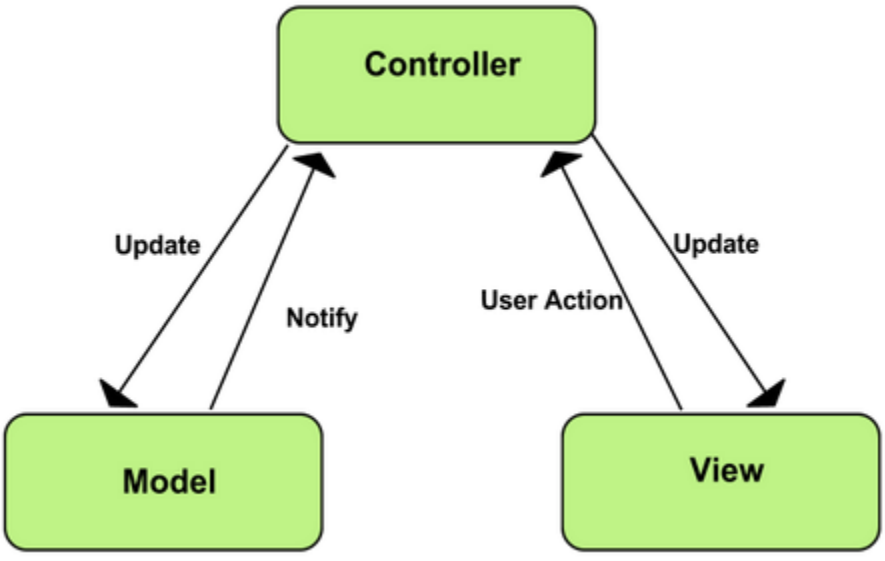

Figure 13: MVC Connections

Other Resources

- Short Readings
  - MVC Architecture – Chrome Developers
  - MVC Tutorial for Beginners: What is, Architecture & Example – Guru99
  - MVC Framework – Introduction – tutorialspoint
- Short Videos
  - MVC Explained in 4 Minutes – Web Dev Simplified
  - What is MVC architecture? - Abhay Redkar
  - What Is MVC? Simple Explanation – Traversy Media
- Long Form Tutorial Video
  - Step-by-step ASP.NET MVC Tutorial for Beginners | Mosh