* 1. **Primary presentation:** 
     1. **MVC**



* + 1. **Three Layers Architecture**



* 1. **Element catalog:**
     1. **Elements and their properties**
        1. **MVC**

|  |  |  |
| --- | --- | --- |
| Elements | | Properties |
| Controller | **ProductController** | The ProductController class contains action methods that render view pages (AddProduct, EditProduct, ViewProduct) |
| **Category**  **Controller** | The CategoryController class contains action methods that render view pages (AddCategory, EditCategory, ViewCategory) |
| **StatisticsController** | The StatisticsController class contains action methods that render view pages (Statistics) |
| **SaleController** | The SaleController class contains action methods that render view pages (BillManagement, Checkout, PriceLog) |
| **StoreController** | The StoreController class contains action methods that render view pages (AddStore, ViewStore, EditStore) |
| **LoyalMember**  **Controller** | The LoyalMemberController class contains action methods that render view pages (AddLoyalMember, ViewLoyalMember, EditLoyalMember) |
| **StoreCategory**  **Controller** | The StoreCategoryController class contains action methods that render view pages (AddStoreCategory, ViewStoreCategory, EditStoreCategory) |
| **POSController** | The POSController class contains action methods that render view pages (AddPOS, ViewPOS, EditPOS) |
| **UserController** | The UserController class contains action methods that render view pages (AddUser, ViewUser, EditUser) |
| View | **AddStore** | This GUI helps the user to add a new Store. |
| **ViewStore** | This GUI helps the user to view the Store. |
| **EditStore** | This GUI helps the user to modify some information about the Store. |
| **AddCategory** | This GUI helps the user to add a new Product Category. |
| **ViewCategory** | This GUI helps the user to view the Product Category. |
| **EditCategory** | This GUI helps the user to modify some information about the Product Category. |
| **AddProduct** | This GUI helps the user to add a new Product. |
| **ViewProduct** | This GUI helps the user to view the Product. |
| **EditProduct** | This GUI helps the user to modify some information about the Product. |
| **BillManagement** | This GUI helps the user to view the list of Bills and the user can see some detail information of each Bill. |
| **PriceLog** | This GUI shows all Price history that was used for each product |
| **Checkout** | This GUI helps the user check bill and make a payment. |
| **AddUser** | This GUI helps the user to add a new User. |
| **ViewUser** | This GUI helps the user to view the User. |
| **EditUser** | This GUI helps user to modify some information about the User. |
| **AddLoyalMember** | This GUI helps the user to add a new loyal Customer. |
| **ViewLoyalMember** | This GUI helps the user to view the loyal Customer. |
| **EditLoyalMember** | This GUI helps user to modify some information about the loyal Customer. |
| **AddStoreCategory** | This GUI helps the user to add a new Retail Store Category. |
| **ViewStoreCategory** | This GUI helps the user to view the Retail Store Category. |
| **EditStoreCategory** | This GUI helps user to modify some information about the Retail Store Category. |
| **Statistics** | To make statistics about the total amount of product (or product category) was bought on month. |
| Model | **ProductModel** | Storing and retrieving the Product information and return a message back to view pages (AddProduct, EditProduct, ViewProduct) |
| **SaleModel** | Storing and retrieving the Sale information and return a message back to view pages (BillManagement, Checkout, PriceLog) |
| **StoreModel** | Storing and retrieving the Retail Store information and return a message back to view pages (AddStore, EditStore, ViewStore) |
| **LoyalMember**  **Model** | Storing and retrieving the Customer information and return a message back to view pages (AddLoyalMember, EditLoyalMember, ViewLoyalMember) |
| **UserModel** | Storing and retrieving the Userinformation and return a message back to view pages (AddUser, EditUser, ViewUser) |
| **CategoryModel** | Storing and retrieving the Product Categoryinformation and return a message back to view pages (AddProductCategory, EditProductCategory, ViewProductCategory) |
| **StoreCategory**  **Model** | Storing and retrieving the Store Category information and return a message back to view pages (AddStoreCategory, EditStoreCategory, ViewStoreCategory) |
| **POSModel** | Storing and retrieving the POSinformation and return a message back to view pages (AddPOS, EditPOS, ViewPOS) |
| **StatisticsModel** | Storing and retrieving the some information related Statistics and return a message back to view pages (Statistics) |

* + - 1. **Three layers architecture**

|  |  |  |
| --- | --- | --- |
| Elements | | Properties |
| Data Transfer Object Layer | **ProductDTO** | Consists of variables only belong to Product (Such as: ProductID, ProductName, BasicCost, CategoryID), not include functions. |
| **BillDTO** | Consists of variables only belong to Bill (Such as: BillID, RetailStoreID, CustomerID, UserID, Date, TotalPoint, PlusPoint, MinusPoint), not include functions. |
| **CustomerDTO** | Consists of variables only belong to Customer (Such as: CustomerID, CustomerName, CustomerAddress, CustomerPhone, Sumpoint), not include functions. |
| **UserDTO** | Consists of variables only belong to User (Such as: UserID, UserAddresss, UserName, UserPhone, Password, RetailStoreID), not include functions. |
| **BillDetailDTO** | Consists of variables only belong to BillDetail (Such as: BillID, ProductID, Quantity), not include functions. |
| **CategoryDTO** | Consists of variables only belong to Category (Such as: CategoryID, CategoryName, Quantity), not include functions. |
| **RetailStoreCategory**  **DTO** | Consists of variables only belong to StoreCategory (Such as: StoreCategoryID, StoreCategoryName), not include functions. |
| **POSDTO** | Consists of variables only belong to POS (Such as: POSID, POSName, RetailStoreID), not include functions. |
| **CostDTO** | Consists of variables only belong to Cost (Such as: RetailStoreID , ProductID, DateStart, DateEnd, Cost), not include functions. |
| **RetailStoreDTO** | Consists of variables only belong to RetailStore (Such as: RetailStoreID , RetailStoreName), not include functions. |
| Data Access Layer | **ProductDA** | Consists of function to retrieve data from database or store, update data - which related Product - to database. It is frequently called by class AddProduct, EditProduct, ViewProduct, Checkout, PriceLog. |
| **BillDA** | Consists of function to retrieve data from database or store, update data - which related Bill - to database. It is frequently called by class BillManagement, Statistics, Checkout. |
| **RetailStoreDA** | Consists of function to retrieve data from database or store, update data - which related RetailStore - to database. It is frequently called by class EditStore, AddStore, ViewStore. |
| **CustomerDA** | Consists of function to retrieve data from database or store, update data - which related Customer - to database. It is frequently called by class AddLoyalMember, EditLoyalMember, ViewLoyalMember, Checkout. |
| **UserDA** | Consists of function to retrieve data from database or store, update data - which related UserAccount - to database. It is frequently called by class AddUser, EditUser, ViewUser. |
| **BillDetailDA** | Consists of function to retrieve data from database or store, update data - which related BillDetail - to database. It is frequently called by class Statistics, BillManagement, Checkout. |
| **CategoryDA** | Consists of function to retrieve data from database or store, update data - which related Category - to database. It is frequently called by class AddCategory, ViewCategory, EditCategory, ViewProduct, EditProduct, AddProduct, Checkout, PriceLog. |
| **RetailStoreCategory**  **DA** | Consists of function to retrieve data from database or store, update data - which related RetailStoreCategory - to database. It is frequently called by class AddStoreCategory, ViewStoreCategory, EditStoreCategory, AddStore, ViewStore, EditStore. |
| **POSDA** | Consists of function to retrieve data from database or store, update data - which related POS - to database. It is frequently called by class AddPOS, ViewPOS, EditPOS. |
| **CostDA** | Consists of function to retrieve data from database or store, update data - which related Cost - to database. It is frequently called by class PriceLog. |

* + 1. **Relations and their properties**

|  |  |
| --- | --- |
| **Connector** | **Properties** |
| **Allowed to use** | The layers are related to each other by the strictly ordered relation allowed to use. |

* 1. **Context diagram:**
  2. **Architecture background:**

System was separate into three layers include: Presentation layer, Transfer Data Object Layer, Data Access Layer.

The reason why we use three layers instead of four layers (such as: Presentation layer, Business Logic layer, Data Object Transfer Layer, Data Access Layer) is:

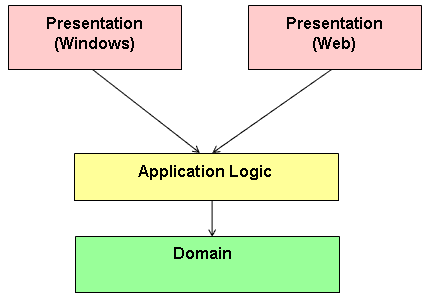
* The controller in MVC undertake tasks that is implemented on Business Logic Layer.
* On Web environment, increasing performance is important. Therefore, eliminating a layer is needed.

Let’s get to the bottom of three layers:

* Presentation Layer (Including MVC framework): is responsible for communication with end user to collect data and show a result of data through components in user interface. We combine MVC framework with the Presentation Layer. For the benefit of MVC framework, we will talk about this later, after show all features of three layers.
  + The view is responsible for providing the user interface (UI) to the user. It is given a reference to the model, and it transforms that model into a format ready to be presented to the user.
  + The controller is responsible for responding to user input, often making changes to the model in response to user input. In this way, controllers in the MVC pattern are concerned with the flow of the application, working with data coming in, and providing data going out to the relevant view.
  + The model that is used to send information to the Data Access Layer, perform business calculations, and even render in a view. Otherwise, these objects represent the domain of the application focuses on, and the models are the objects you want to save, create, update, and delete.
* Data Transfer Object: this layer can creates objects to support Presentation Layer to perform and used is as a transfer object and pass-by-reference to Data Access Layer.
* Data Access Layer manages the physical storage and retrieval of data from database.

Besides using 3 layers, we also use MVC framework. There is the list of MVC# framework features that is the reason why we use this:

* *Views and controllers get connected automatically*. The MVC framework automatically establishes links between views and corresponding controllers. Therefore, the developers do not care about the associating views and controllers that linked to their views.
* *Multiple GUI platforms supported.* MVC allows targeting different GUI platforms such as: Window, Web, Silverlight, etc,… Therefore, the same application can be used with quite different presentation layers - one for Windows, the other for Silverlight or Web environment, etc.:



### *Platform-independent navigation to views*. To make application logic fully independent of the presentation layer, MVC provides a platform-independent way of navigating to views. Instead of  activating a Windows form or redirecting to a Web page a developer just simply call a uniform Navigator.Navigate(...) method. For example:

public class OrderDetailsController

...

public void ProcessOrder()

{

// No Response.Redirect(...) or Form.Show() calls

Task.Navigator.Navigate(OrderSupportTask.ProcessOrder);

}

### *Tasks concept.* Sometime, we have to unites several views with their controllers to do some job, this is called a task. For example a checkout task may consists of two views, one to choose a product (such as: Milk, Drink, Cake,… we can order in supermarket), the other – to do the payment. In MVC# all controllers within a task are given a link to the task object. Generally a task can be expressed as a workflow or a state machine.

### 