1. **Static Perspective**

**Module View**

**(Layered Style and Uses Style)**

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



* + 1. **Layereds Style**



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

|  |  |  |
| --- | --- | --- |
| Elements | | Properties |
| Controller | **StudentController** | The StudentController class contains action methods that render view pages. |
| **UserController** | The UserController class contains action methods that render view pages (AddUser, EditUser, ViewUser) |
| **StatisticsController** | The StatisticsController class contains action methods that render view pages (Statistics) |
| View | **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 the user to modify some information about the User. |
| **ViewStudent** | This GUI helps the user to view the Student. |
| **Statistics** | To make statistics about the total amount of Student following |
| Model | **StudentModel** | Storing and retrieving the Student information and return a message back to view pages. |
| **UserModel** | Storing and retrieving the User information and return a message back to view pages (AddUser, EditUser, ViewUser) |
| **StatisticsModel** | Storing and retrieving the some information related Statistics and return a message back to view pages (Statistics) |

* + - 1. **Layered Style**

|  |  |  |
| --- | --- | --- |
| Elements | | Properties |
| Presentation layer | **Logon\_UI** | User enter username and password at Logon screen. |
| **Student\_Mana**  **gement\_UI** | This GUI help user manage student record such as: import student record, edit student, view student list. |
| **Statistics\_UI** | User can view student record following criteria. The student information are displayed on table. |
| Business Logic Layer | **Printing\_BL** | This class have functions support printing receipt |
| **Authority\_BL** | This class contains functions that perform business logic for user management such as: add new user, edit user, view user. |
| **Data\_Handle\_BL** | Contains function that perform business logic for student record management such as: import student, view student, edit student |
| **Statistics\_BL** | Contains fucntion that help user to review list of student following criteria. |
| **Logon\_BL** | This class have functions that call functions from class User\_EF of Data Access Layer to check username and password of user. |
| **Common\_BL** | Supports all class on Business Logic Layer by using common functions. |
| Data Access Layer | **User\_EF** | Consists of function to retrieve data from database or store, update data - which related User - to database. It is frequently called by class Authority**,** Logon. |
| **Student\_EF** | Consists of function to retrieve data from database or store, update data - which related Student - to database. It is frequently called by class Data Handle, Statistics. |
| **Department\_EF** | Consists of function to retrieve data from database or store, update data - which related Department - to database. It is frequently called by class Statistics, Data Handle. |
| **Context** | Context class is the primary class for interacting with data as objects that are instances of entity types that are defined in an Entity Data Model (EDM). |

* + 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. |
| **Uses** | The uses style shows how modules depend on each other; it is helpful for planning because it helps define subsets and increments of the system being developed. |

* 1. **Architecture background:**

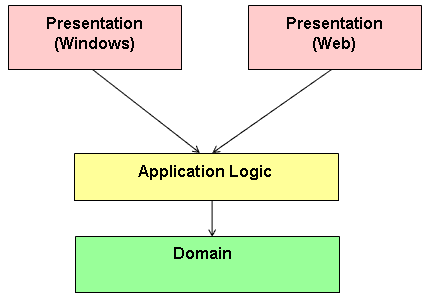
System was separate into three layers include: Presentation layer, Business Logic Layer, Data Access Layer. W

Let’s get to the bottom of three layers:

* Web aplication 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 Business Logic Layer if this request to model need more resourse to perform business calculations.Basically, model perform business calculations on simple level and 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.
* Business Logic Layer contains components that are used to perform business logic and defines business entities that are used by the business component.
* Data Access Layer manages the physical storage and retrieval of data from database.We use the entity framework to Data Access Layer.
  + With Entity Framework, the developers issue queries using LINQ, then retrieve and manipulate data as strongly typed objects. The Entity Framework’s ORM implementation provides services like change tracking, identity resolution, lazy loading, and query translation so that developers can focus on their application-specific business logic rather than the data access fundamentals.

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.

### 