we will not be able to run unit tests on the UI code. So the only way to test the application would be to manually test the GUI. The page controller based design does not support unit testing, and we would not be able to use automated unit testing tools such as NUnit, MBUnit and so on (which we can easily use to test the other layers such as BL and DAL).

There are ways to perform automated testing in GUI using testing tools that use Javascript to actually perform the button click events through the code, although they are clumsy and difficult to use. Even if we write scripts today, they will need to change if the GUI changes in the future, which is very much possible as the GUI can be changed many times during a project's lifetime and also after it is finished. This makes unit testing more difficult as one would need to rewrite the automated testing scripts on every GUI change. So most people tend to use brute force testing, which involves clicking all possible UI controls (such as buttons and so on) and verifying whether the code works as expected. This is a very time-consuming task, and if the GUI changes, the testing needs to be carried out again.

We will now see how MVC design helps us implement a clean separation between the UI and the controller, and also make our UI unit-testable.

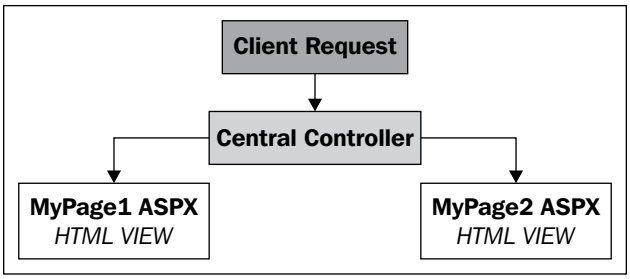
MVC Design: A Front Controller based Approach

MVC, which stands for Model View Controller, is a design pattern that helps us achieve the decoupling of data access and business logic from the presentation code , and also gives us the opportunity to unit test the GUI effectively and neatly, without worrying about GUI changes at all. In this section, we will first study the basic MVC pattern and then move on to understanding the ASP.NET MVC framework.

A framework is a set of tools that includes libraries or methods developed according to a certain architecture, so that applications do not need to re-invent the wheel. Instead of re-writing the basic implementation each time, they can use the framework and abstract themselves from the internal framework implementation details.

Front Controller Design

MVC is based on a front controller design, where we have a centralized controller instead of multiple controllers, as was the case in the page controller based design that we saw earlier. By default, ASP.NET is page controller based. So making a front controller based project would require a lot of work (using HttpHandlers to route the requests manually). Basically, in a front controller design, we trap all of the client requests and direct them to a central controller, and the controller then decides which view to render (or which ASPX page to process). Here is how a basic model of a front controller design works:



As you can see, the front controller sits at the "front" of all of the pages and renders a view based on logic in the central controller file. In the next section we will study and analyze exactly what goes on inside a controller, a view, and a model.

Basics of MVC

Let's first get into the theoretical aspects of MVC. MVC design has three major parts:

- **Model:** This refers to the data that is shown in the UI. This data can come from different sources, for example, a database.
- **View:** This refers to the user interface (UI) components that will show the model data.
- **Controller:** This controls when to change the view, based on user actions, such as button clicks.