ASP.NET MVC with Razor

Intro to Views in MVC Applications in C#

Goals for ASP.NET MVC with Razor

What is the MVC architecture pattern? (Review)

What is ASP.NET MVC and how does it follow the MVC architecture pattern? (Review)

Introduction to Razor syntax for making Views in ASP.NET MVC

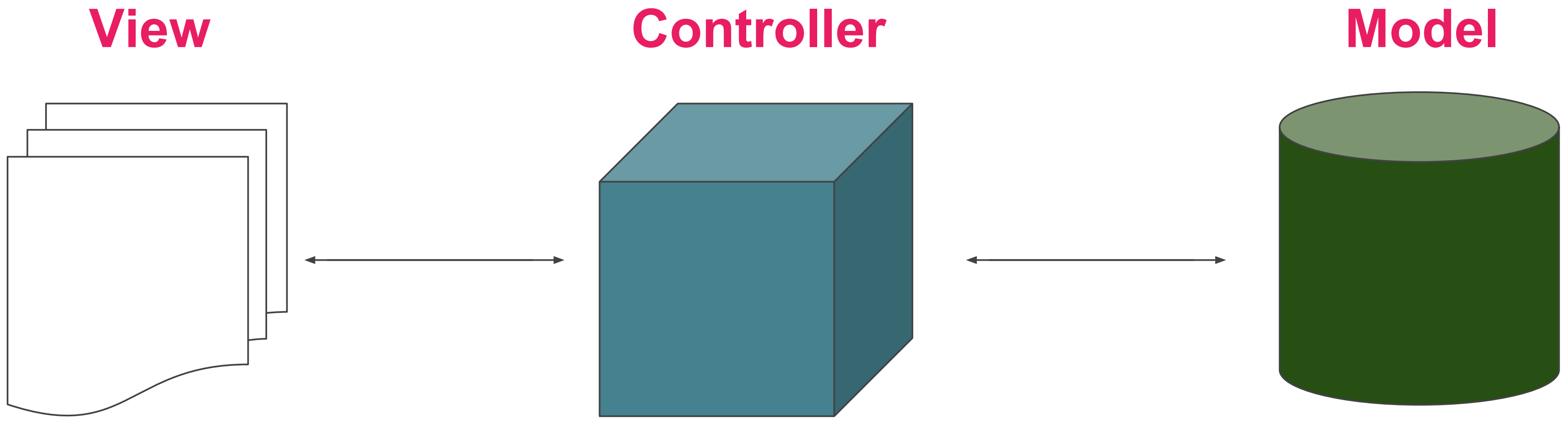
MVC - Model, View, Controller

Model - Data classes, talks to data storage

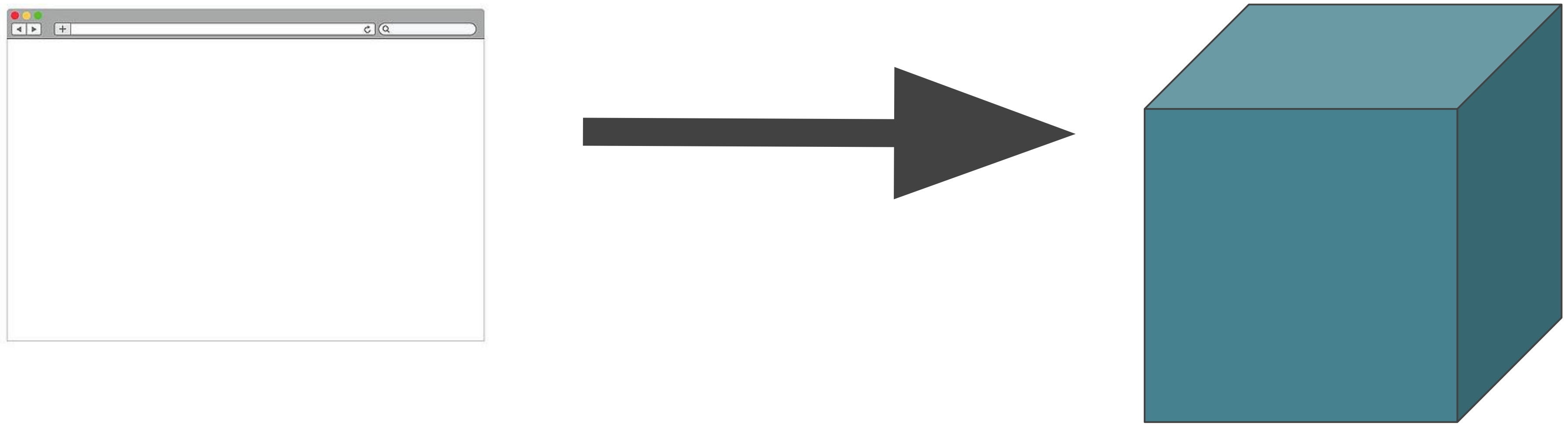
Controller - Application “Hub” that directs traffic and coordinates requests

View - User Interface

ASP.NET MVC Application Structure



Controller receives a request



http://www.mycompany.com/products

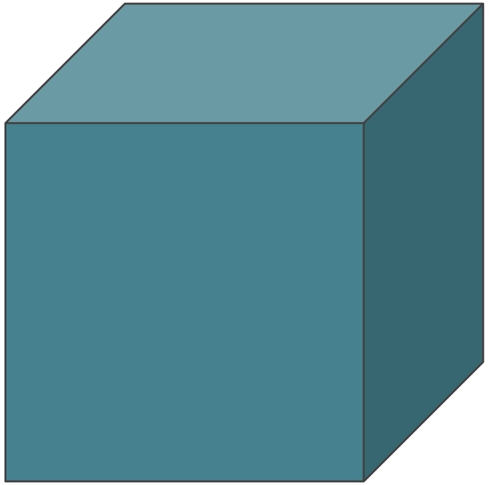
Controller

Controller processes request

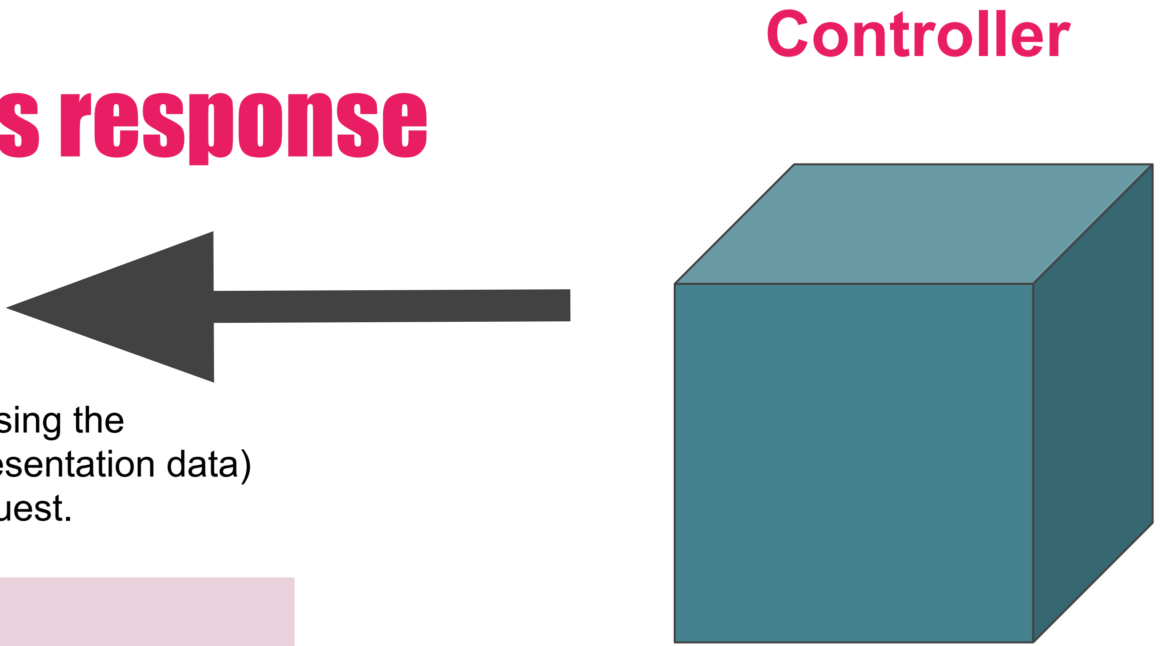
Controller gets data from Model that it needs to complete request

```
List<Product> myProducts = Model.GetProducts();

string userName = User.FirstName + “ “ + User.LastName;
```



Controller returns response

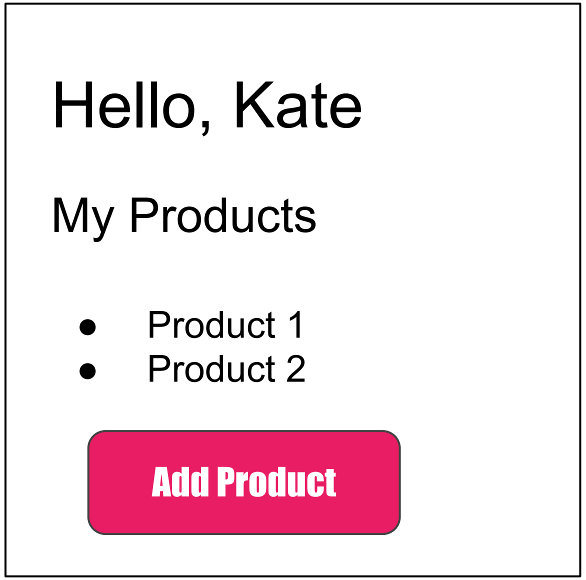


Controller formats Model data for view using the ViewBag or a ViewModel (a class for presentation data) and returns the appropriate View for request.

```
ViewBag.Products = myProducts;
ViewBag.UserName = userName;
return View();
```

```
ProductsViewModel myViewModel =
    new ProductsViewModel(myProducts,
userName);
return View(myViewModel);
```

View renders data in User Interface

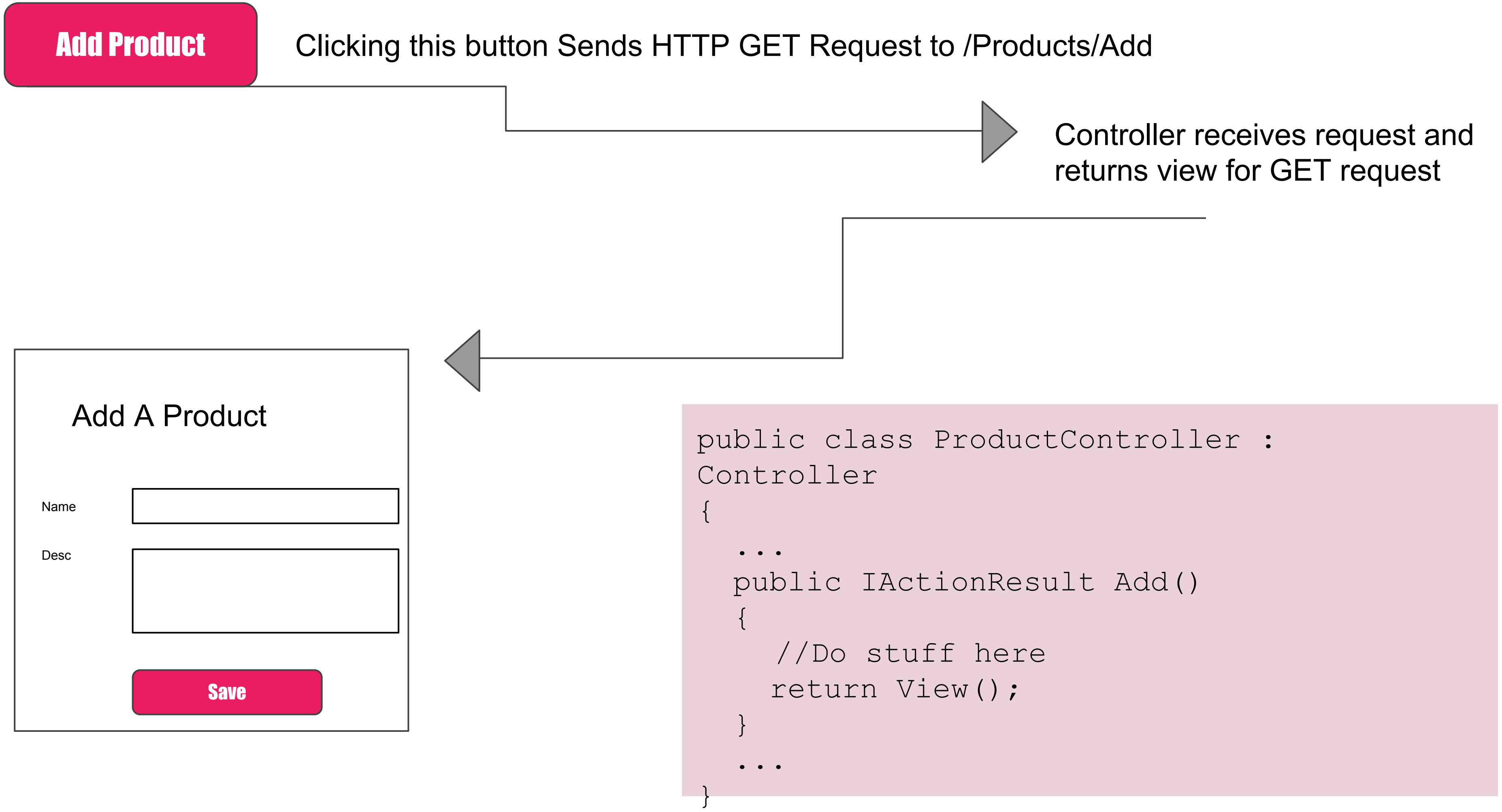


Hello, Kate

My Products

- Product 1
- Product 2

```
<h1>Hello, @ViewBag.Name</h1>
<br/>
<h3>My Products</h3>
<ul>
  @{
  for(int i = 0; i<ViewBag.Products.Count; i++)
     {
          <li>@ViewBag.Products[i].Name</li>
     }
  }
</ul>
<a href="/add" class="button">Add Product</a>
```

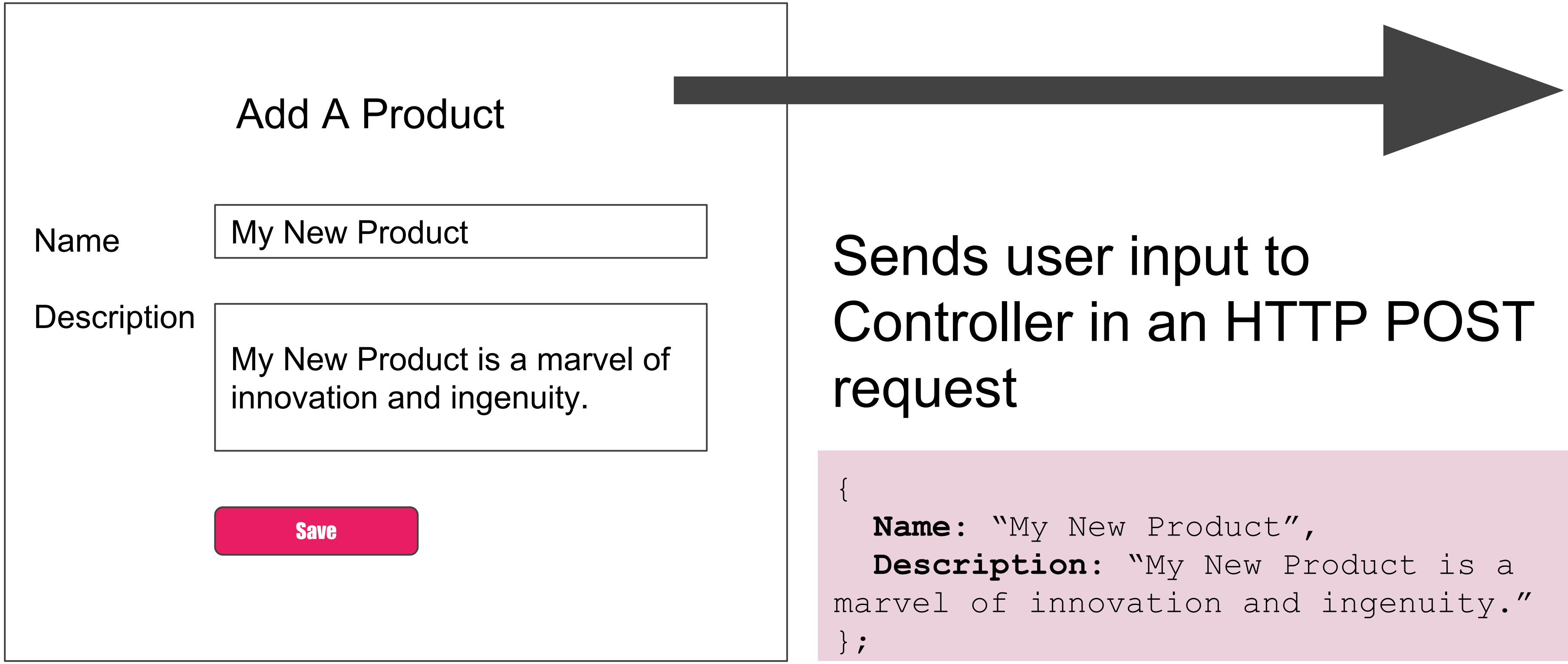


Clicking this button Sends HTTP GET Request to /Products/Add

Controller receives request and returns view for GET request

```
public class ProductController :
Controller
{
  ...
  public IActionResult Add()
  {
    //Do stuff here
    return View();
  }
  ...
}
```

View accepts user input and sends request



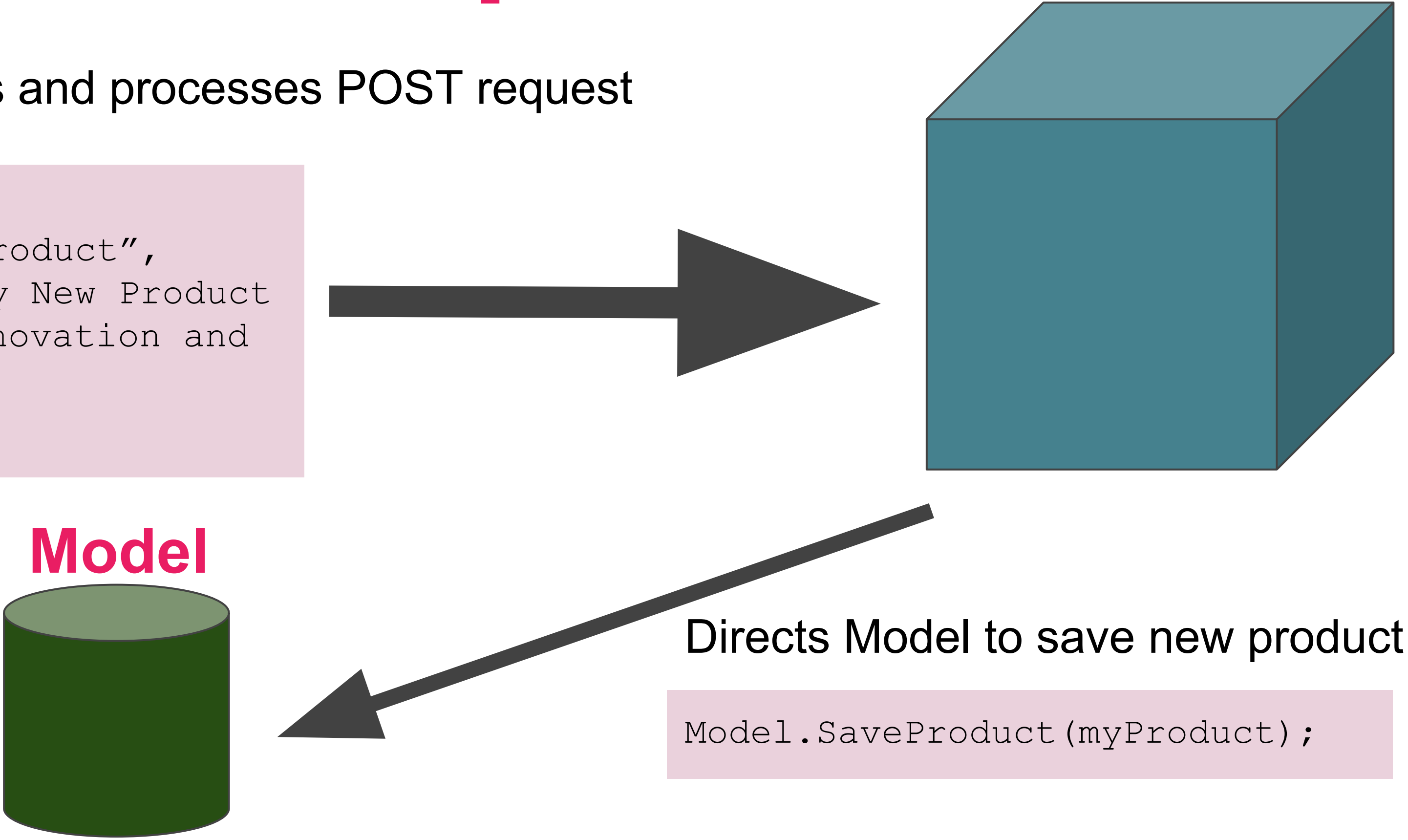
Sends user input to Controller in an HTTP POST request

```
{
  Name: "My New Product",
  Description: "My New Product is a
marvel of innovation and ingenuity."
};
```

Controller receives request

Controller receives and processes POST request

```
{
  Name: "My New Product",
  Description: "My New Product
is a marvel of innovation and
ingenuity."
}
```



Directs Model to save new product

```
Model.SaveProduct(myProduct);
```

Razor Syntax

Dynamically render HTML using C# code

```
@{

  //C# code goes here

}

@{string Message = “Hello”;}
```

Think Handlebars or Angular Templating, but using C#

The Razor engine translates html with embedded C# into pure html file - BEFORE the html reaches browser

```
<!-- Single statement blocks  -->
@{ var accountBalance = 78472; }
@{ var myMessage = "Hello, my name is Inigo Montoya."; }

<!-- Inline expressions -->
<p>The balance of your account is: @accountBalance </p>
<p>@myMessage You killed my father. Prepare to die.</p>

<!-- Multi-statement block -->
@{
  var greeting = "Hello, valued customer!";
  var weekDay = DateTime.Now.DayOfWeek;
  var greetingMessage = greeting + " Today is: " + weekDay;
}
<p>The greeting is: @greetingMessage</p>

<!-- Loops -->
@for(var i = 10; i < 21; i++)
{
  <p>Line #: @i</p>
}
```

Controllers can send data to the view with a ViewBag or ViewModel, and Razor can access it and present it in the User Interface.

ViewBag

dynamically share values from the controller to the view

ViewBag is a **dynamic** object and therefore has no IntelliSense support; errors will not be discovered during compile, **only at runtime.**

Works like a **dictionary**, not a class or property.

```
//Controller adds data to ViewBag
public ActionResult Index()
{
  List<string> students = new List<string>{"Susan", "Bob", "Phil"};
  ViewBag.Students = students;
  Return View();
}


//Razor accesses data for UI Presentation
<ul>
  @{foreach (var student in ViewBag.Students)
    {
      <li>@student</li>
    }
  }
</ul>
```

ViewModels

a one-stop-shop for all the View data in one class

Using ViewModels allows you to shape multiple objects from one or more data models or sources into a single class.

A ViewModel represents all the data that you want to display in your View, whether it be used for static text or for input values (like textboxes and dropdown lists).

```
//ViewModel holds all data needed for view
{
  public ProductsViewModel(List<Product> products, User currentUser)
  {
    this.UserProducts = products;
    this.UserName = ${{currentUser.FirstName} {currentUser.LastName}};
  }
  public List<Product> UserProducts { get; set; }
  public string UserName { get; set; }
}

//Controller creates new instance of ViewModel and passes it to view
public ActionResult Index()
{
  ProductsViewModel viewModel = new ProductsViewModel(Products, CurrentUser);
  Return View(viewModel);
}

//Razor gets access to the ViewModel in View
@model MyProject.Web.ViewModels.ProductsViewModel //at top of page
<h1>Hello, @model.UserName</h1>
```

Let’s Practice.