MVC

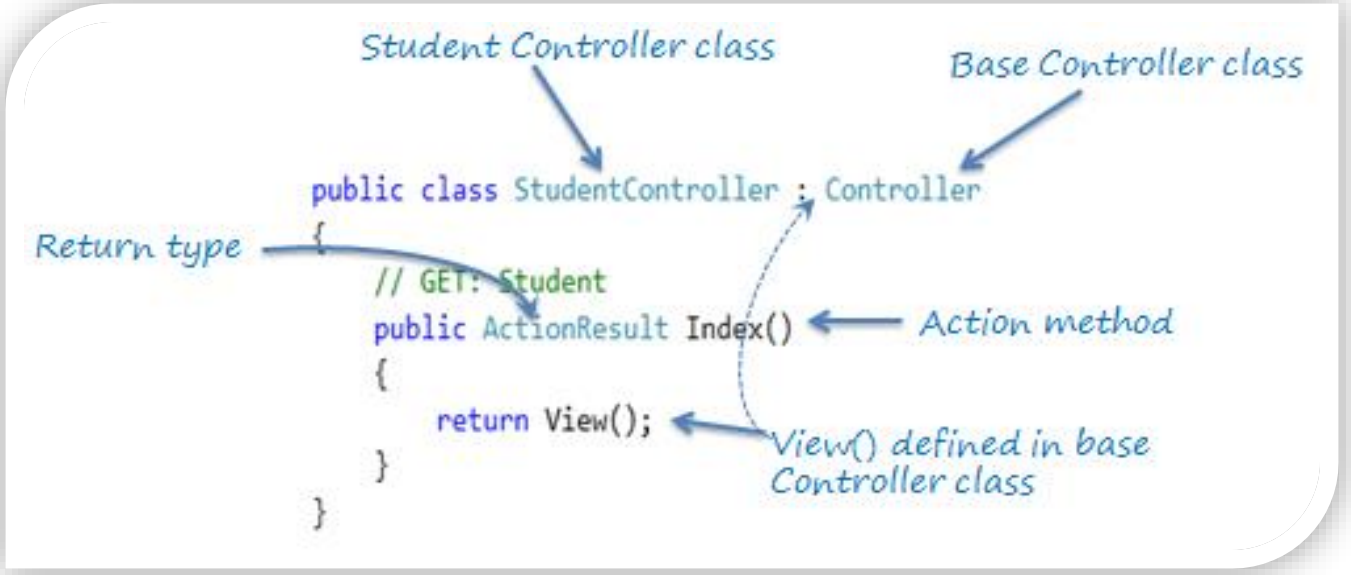
Christen Zarif Foad

OutLine

- Action Selector
- Model Binding

Action method

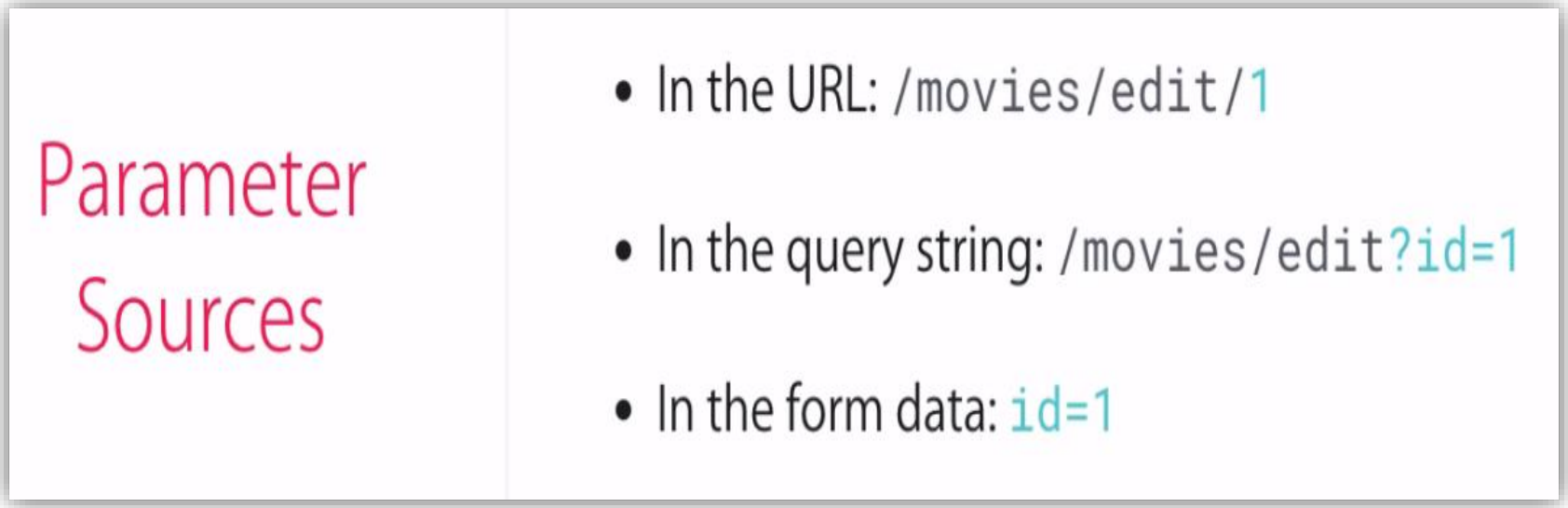
- Public methods of a Controller class
- Action method must be public. It cannot be private or protected
- Action method cannot be overloaded
- Action method cannot be a static method.



Action method Parameters:

- Every action methods can have input parameters as normal methods. It can be primitive data type or complex type parameters

- Action method can include Nullable type parameters.



Action method Parameters:

- The values for action method parameters are **retrieved from the request's data collection**.
  - The data collection includes name/values pairs for form data or query string values or cookie values.

- **Model binding** in ASP.NET MVC automatically maps the URL query string or form data collection to the action method parameters if both names are matching.

Model Binding

Model Binding

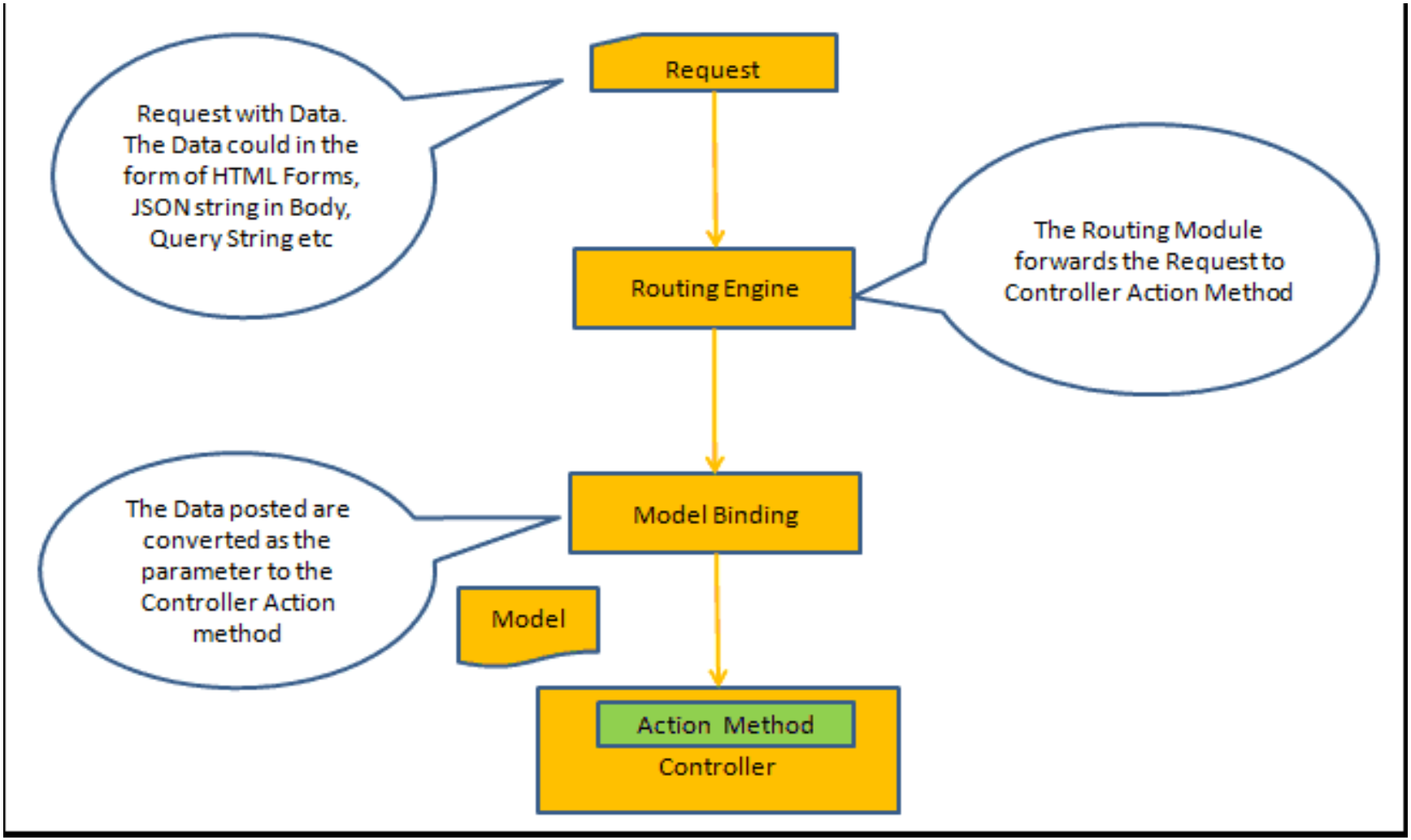
- The **Model binding** is the process of mapping the data posted over an HTTP request to the parameters of the action method in the Controller.

- The HTTP Request can contain data in various formats. The data can contain in the HTML form fields. It could be part of the route values. It could be part of the query string or it may contain in the body of the request.

Model Binding

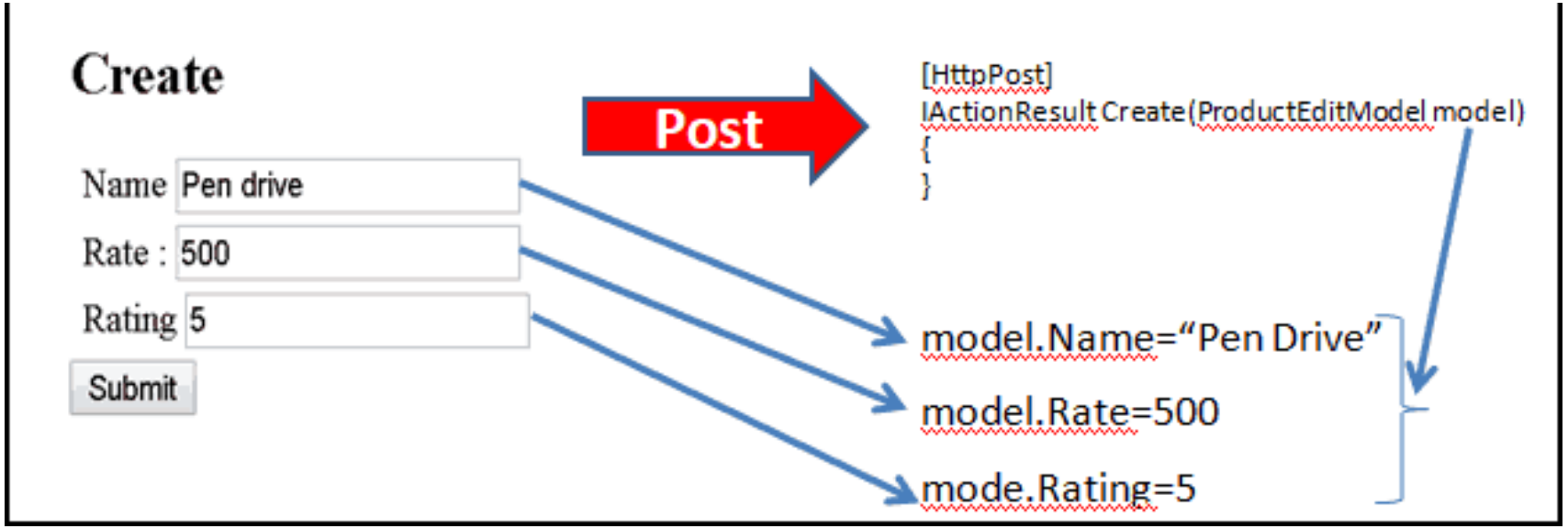
- Controllers are classes
- Actions are methods
- Methods take parameters
- MVC will convert forms to parameters

Model Binding



Model Binding

- Getting Data From forms to controller

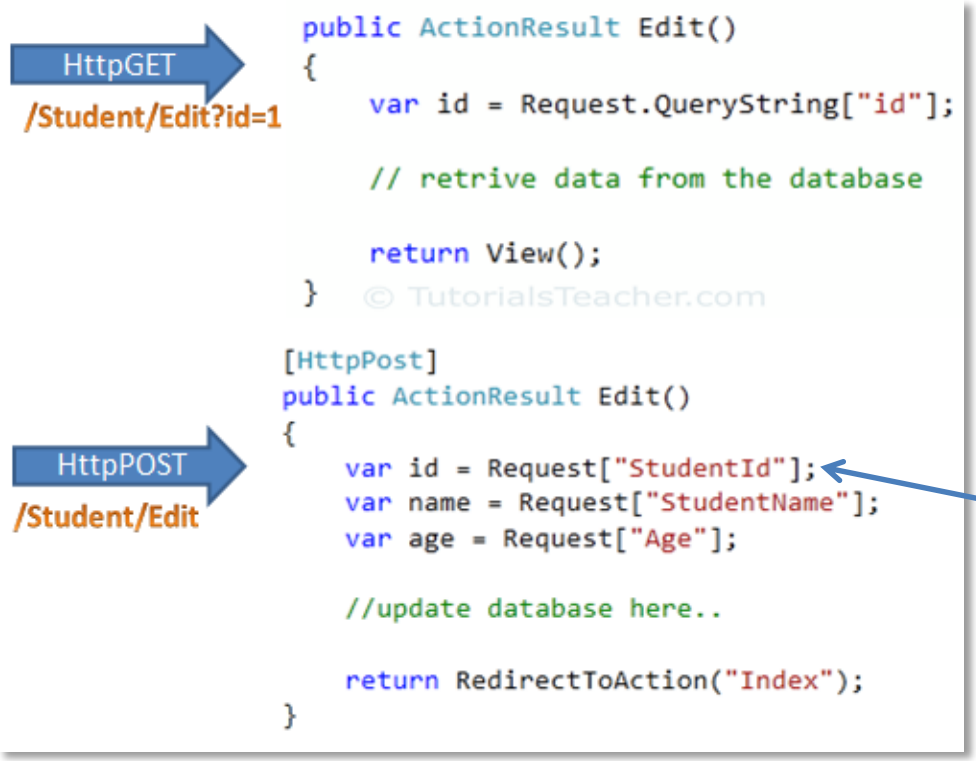


- **For the Binding to work correctly**
  - The Property Name must match with the Request data
  - Properties must be defined as public settable
- Model Binding Demo

BEFORE MODEL BINDING

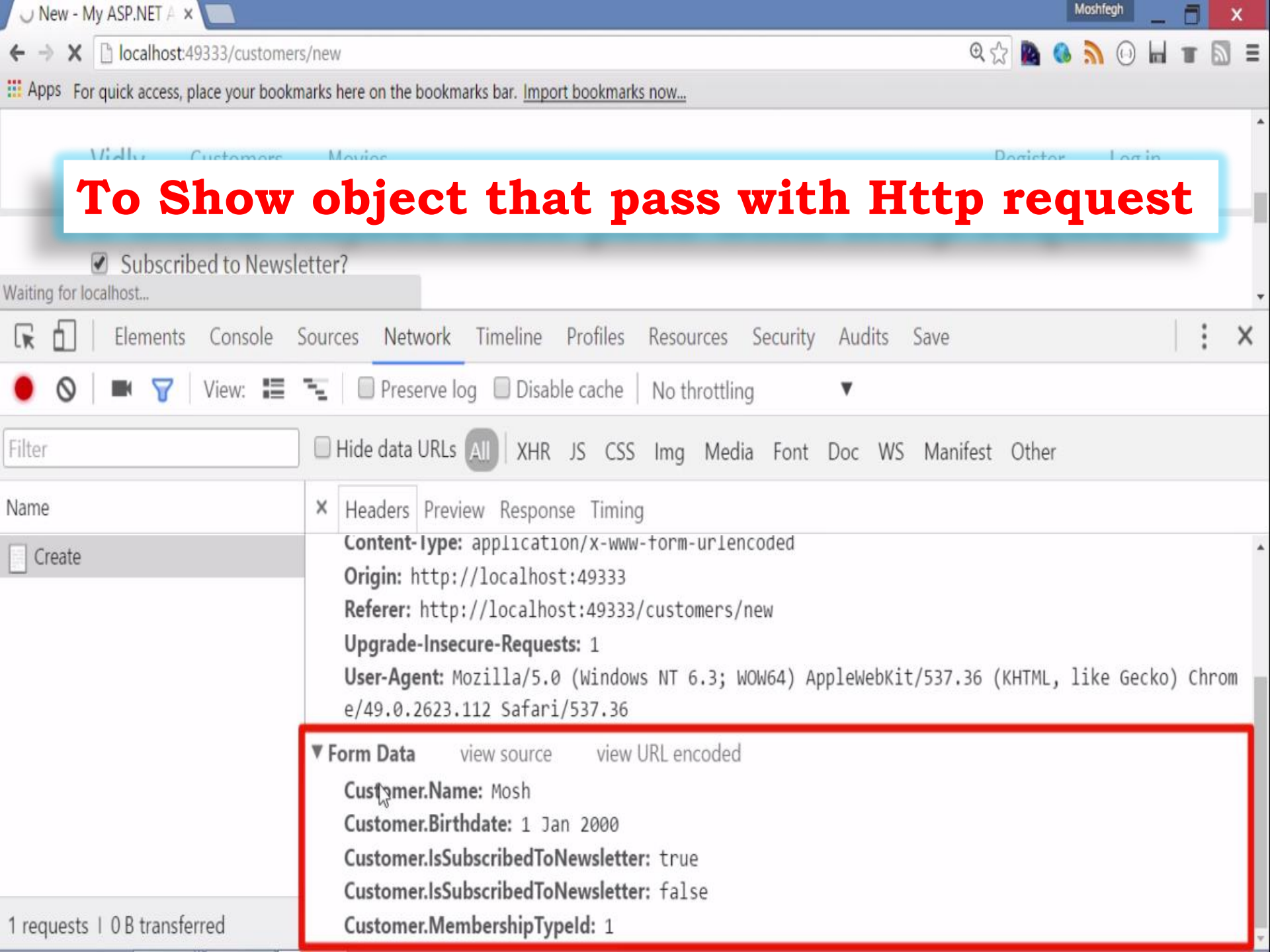
Request Object

- The traditional way to get data from **http Request**



Request.Form[“StudentId”]

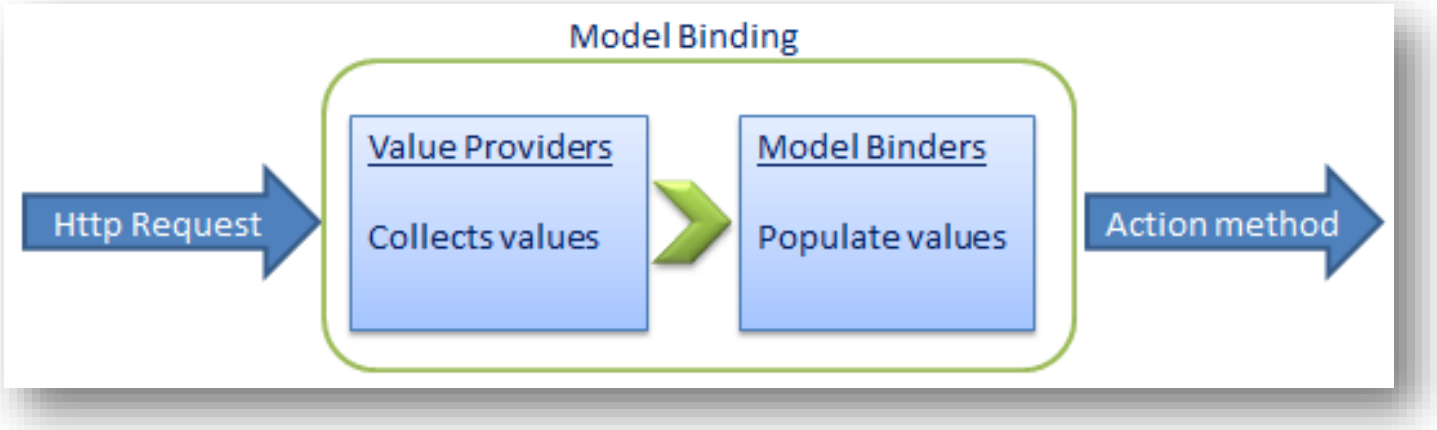
- Accessing request values using the Request object is a cumbersome and time wasting activity.


New - My ASP.NET
Moshfegh
localhost:49333/customers/new
Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
To Show object that pass with Http request
Subscribed to Newsletter?
Waiting for localhost...
Elements Console Sources Network Timeline Profiles Resources Security Audits Save
View: Preserve log Disable cache No throttling
Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other
Name
Create
Headers Preview Response Timing
Content-Type: application/x-www-form-urlencoded
Origin: http://localhost:49333
Referer: http://localhost:49333/customers/new
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
e/49.0.2623.112 Safari/537.36
Form Data view source view URL encoded
Customer.Name: Mosh
Customer.Birthdate: 1 Jan 2000
Customer.IsSubscribedToNewsletter: true
Customer.IsSubscribedToNewsletter: false
Customer.MembershipTypeId: 1
1 requests | 0 B transferred

DEMO TEST REQUEST PROPERTY

Model Binder

- Model Binding is Two step Process:
  - Collecting values from request using Value Providers
  - Populating models with those values using Model Binders



ValueProvider

- A value provider is a class whose primary responsibility is to extract a value from an incoming request

- ASP.NET provides several built in value providers and also allows you to create your own.

- To Create Custom ValueProvider

Built in ValueProvider

- Available Built in Provider, Sorted according priority
- Based on priority , **Model Binders** looks into Value Providers to find specific value on model property

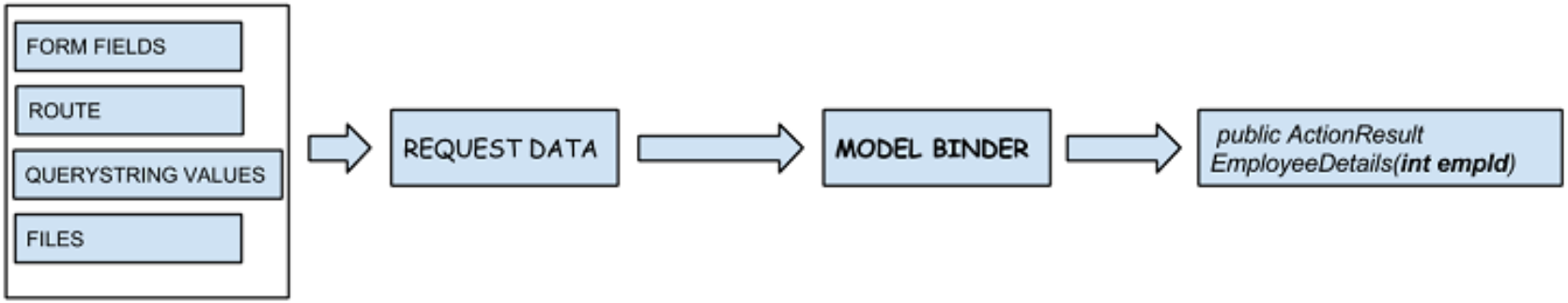
Source	Provider	Fetch Data From
Form Fields	FormValueProvider	Request.Form
Route Data	RouteDataValueProvider	RouteData.Values
Query String	QueryStringValueProvider	Request.QuryString
Posted Files "FileUpload"	FormFileValueProvider	Request.Files

Default Model Binding

- When we define action method parameters , the values are auto populated in those parameters by the default model binder.
- There are two main points to consider in the Model Binding process
  - The name of the *field* defined in the request should be the same as the name of the action method *parameter*
  - The request data should be convertible to the action method parameter

Default Model Binding

- The Model binder relies on another component to provide it the request values,
- The ***Value provider***.
  - The value providers searches the different request sources for the data and then provides the data to the Model Binder which then binds the data to the action method parameter .



Default Model Binder (Con.)

- **DefaultModelBinder** class Maps a browser request to a data types of objects
  - **Primitive Type Like:**
    - String ,double,or DataTime objecs
  - **Collection Like**
    - Icollection<T> ,Ilist<T>,Idictionary<TKEy,Tvalue>
  - **Model Class Like:**
    - Person, Department …

Model Binding

- With model binding, MVC framework converts the **http request values** (from query string or form collection) to **action method parameters using valueProvider.**
- This binding is ***case insensitive***.
  - So "id" parameter can be "ID" or "Id“
- **These parameters can be of primitive type or complex type**.
  1. Binding to Primitive type
  2. Binding to Complex type
  3. FormCollection
  4. Bind Attribute

1-Binding to Primitive type

- View

```
<form action="/Account/Login" method="post">
    <input type="text" name="Email">
    <input type="text" name="Password">
    <button type="submit">Login</button>
</form>
```

- Controller

```
public class AccountController : Controller
{
    public ActionResult Login(string Email,string Password)
    {
        return View();
    }
}
```

2-Binding to Complex type

- Model

```
public partial class Account
{
    public int Id { get; set; }
    public string UserName { get; set; }
    public string Password { get; set; }
}
```

- View

```
<form action="/Account/Login" method="post">
    <input type="text" name="UserName">
    <input type="text" name="Password">
    <button type="submit">Login</button>
</form>
```

- Controller

```
[HttpPost]
public ActionResult Login(Account account )
{
    return View();
}
```

4-Bind Attribute

- ASP.NET MVC framework also enables you to **specify which properties** of a model class you want to bind.
- The [Bind] attribute will let you specify the exact properties a model binder should include or exclude in binding.
- The Bind attribute will improve the performance by only bind properties which you needed.

```
[HttpPost]
public ActionResult Edit([Bind(Include = "StudentId, StudentName")] Student std)
{
```

```
[HttpPost]
public ActionResult Edit([Bind(Exclude = "Age")] Student std)
```

Advantage of Model Binding

- It **simplifies** accessing the request information in the action method.
- It makes the action methods easier to **Unit Test** as the action method is not relying on any framework component such as Request or Response.

HttpPostedFileBase Model Binder

- In this case, there's a default value provider called the HttpFileCollectionValueProvider which supplies the uploaded files to the model binder.
- View:

```
<form method="post"
      enctype="multipart/form-data"
      action="@Url.Action("UploadData")">

    <input name="upload" type="file" />
    <input type="submit" value="upload" />
</form>
```

- Controller

```
public ActionResult UploadData(HttpPostedFileBase upload)
{
    var f = System.IO.Path.GetFileName(upload.FileName);
    upload.SaveAs(Server.MapPath("~/Uploads/" + f));
    return RedirectToAction("Index");
}
```

Action Selectors

Action Selectors

- Action selector is the **attribute** that can be applied to the action methods**. It helps routing engine to select the correct action method** to handle a particular request.
- MVC framework routing engine uses Action Selectors attributes to determine which action method to invoke
- MVC 5 includes the following action selector attributes:
  - ActionName
  - NonAction
  - ActionVerb

Action Selectors “ActionName”

- ActionName attribute allows us to specify a **different action name** than the method name.
- It allows you to start your action with a number or include any character that .net does **not** allow **“-”** in an identifier.

```
public class StudentController : Controller
{


    [ActionName("find")]
    public ActionResult GetById(int id)
    {
        // get student from the database
        return View();
    }
}
```

- *“http://localhost/student/find/1”* 
- *“http://localhost/student/getbyid/1”.* 

Action Selectors “NonAction”

- NonAction selector attribute indicates that a public method of a Controller is not an action method.

```
public class StudentController : Controller
{
    public StudentController()
    {

    }

    [NonAction]
    public Student GetStudnet(int id)
    {
        return studentList.Where(s => s.StudentId == id).FirstOrDefault();
    }
}
```

Action Selectors “ActionVerbs”

- The ActionVerbs selector is used when you want to control the selection of an action method based on a **Http Request** method (Get – Post – Put – …..)

- MVC framework supports different ActionVerbs:
  - HttpGet, HttpPost, HttpPut, HttpDelete, HttpOptions & HttpPatch.

- If you do not apply any attribute then it considers it a **GET** request by default.

Http methods

Http method	Usage
GET	• To **retrieve** the information from the server. • Parameters will be appended in the query string.
POST	✓ used to **send** data to the server, ✓ To create a new resource.
PUT	• To **update** an existing resource “Full Update”.
HEAD	✓ Identical to GET except that server do not return message body. ✓ Same as GET, but transfers the status line and header section only
OPTIONS	• OPTIONS method represents a request for information about the communication options supported by web server.
DELETE	✓ To **delete** an existing resource.
PATCH	• To full or partial update the resource.

ActionVerbs (con.)

http://localhost/Student/Edit/1

HttpGET

```
public ActionResult Edit(int Id)
{
    var std = students.Where(s => s.StudentId == Id).FirstOrDefault();

    return View(std);
}
```


http://localhost/Student/Edit

HttpPOST

```
[HttpPost]
public ActionResult Edit(Student std)
{
    //update database here..

    return RedirectToAction("Index");
}
```

ActionVerbs (con.)

- Multiple action verbs can be applied to a single action method using AcceptVerbs attribute

```
[AcceptVerbs(HttpVerbs.Post | HttpVerbs.Get)]
public ActionResult GetAndPostAction()
{
    return RedirectToAction("Index");
}
```

- To Upload Image:
  - https://rachelappel.com/category/asp-net-mvc/