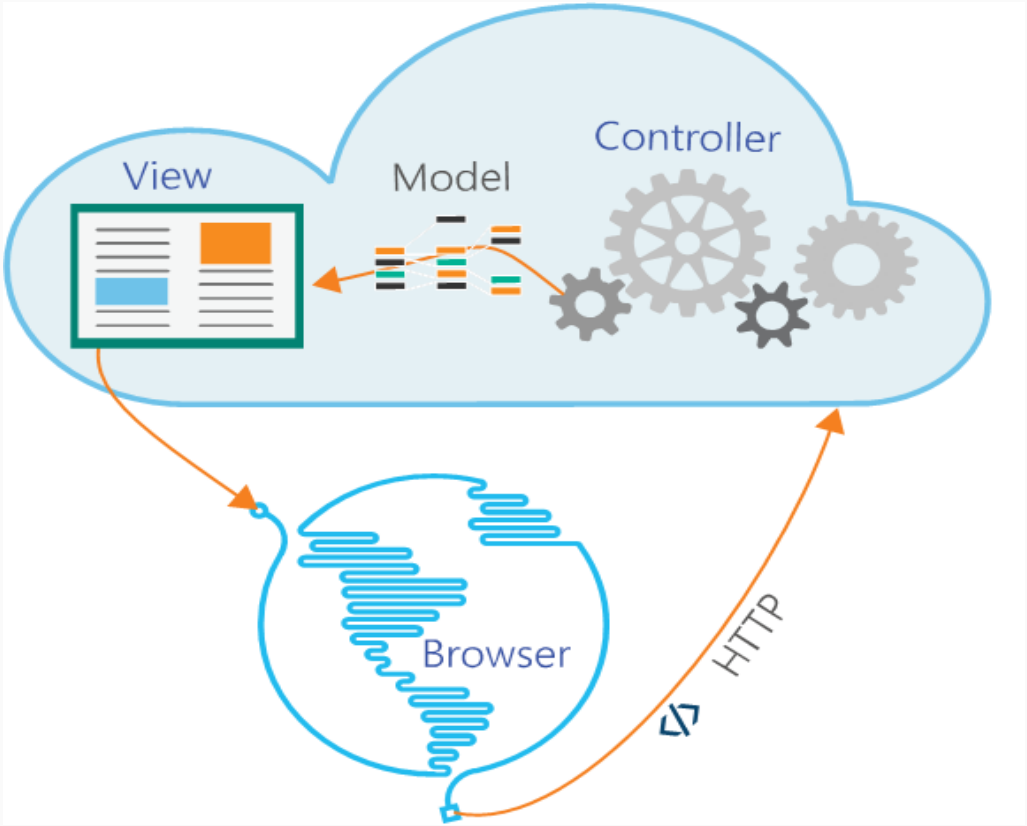
Școala
informală
de IT

Controllers & Actions


View
Model
Controller
Browser
HTTP

Controllers and Actions

What is Controller

- It is a class
- Derives from the base System.Web.Mvc.Controller class
- Generates the response to the browser request

```
public class HomeController : Controller
{
        public IActionResult Index()
        {
            ViewBag.Message = "Welcome to ASP.NET MVC!";

            return View();
        }

        public IActionResult About()
        {
            return View();
        }
}
```

Controller

- The core MVC component
- A set of classes that handles
  - Communication from the user
  - Overall application flow
  - Application-specific logic
- Every controller has one or more "Actions“ mapped to Verbs

Controllers

- All the controllers should be available in a folder by name Controllers
- Controller naming standard should be "nameController" (convention)
- Routers instantiate controllers in every request
  - All requests are mapped to a specific action
- Every controller should inherit Controller class
  - Access to Request (context) and HttpContext

Actions

- Actions are the ultimate request destination
  - Public controller methods
  - Non-static
  - No return value restrictions
- Actions typically return an IActionResult

```
public IActionResult Contact()
{
        ViewBag.Message = “Your contect page.”;
        return View();
}
```

Actions

- You can pass a model
- Or you can pass a model and a view name.
- The same model type needs to be declared in View using @model

```
public IActionResult Create()
{
  Dog model=new Dog();
  return View(model);
}
```

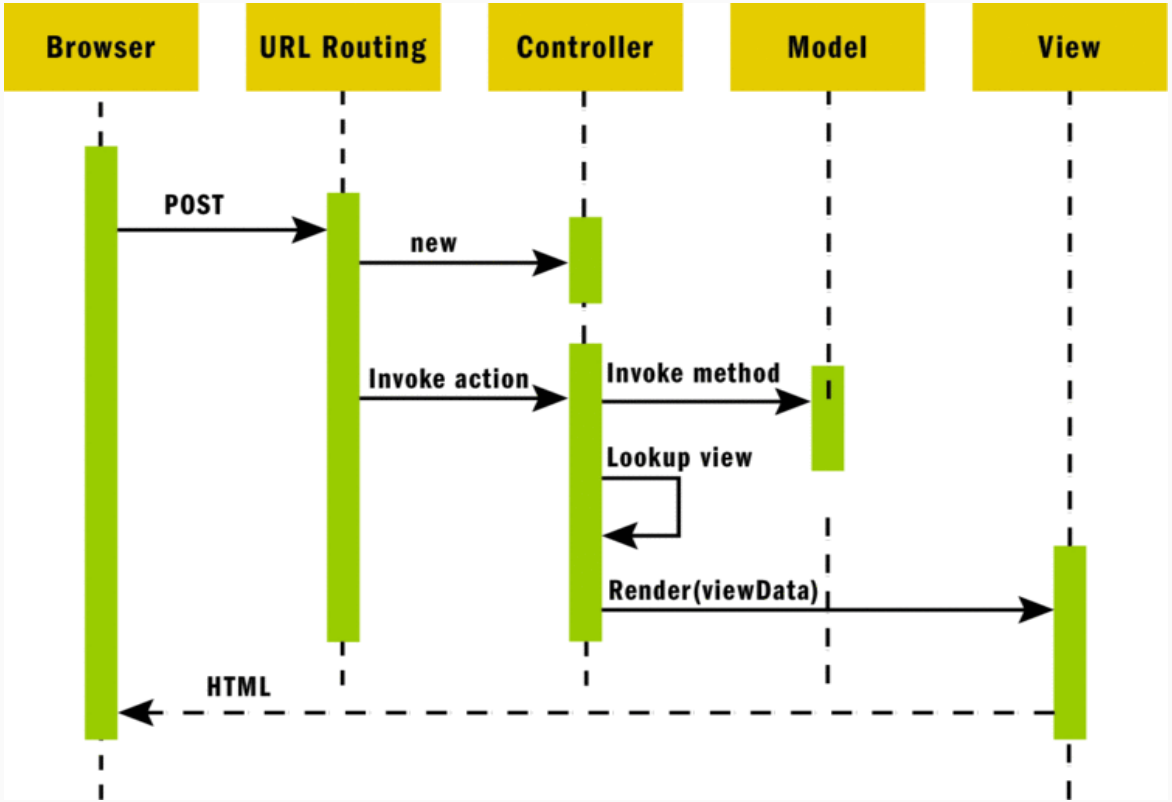
Will search a view called Create.cshtml
under Views/{ControllerName}
or
under Views/Shared

Action Selectors

- ActionName(string name)
- AcceptVerbs
  - HttpPost
  - HttpGet
  - HttpDelete
  - HttpOptions
  - …
- NonAction
- RequireHttps

```
public class UsersController : Controller
{
        [ActionName("UserLogin")]
        [HttpPost]
        [RequireHttps]
        public ActionResult Login(string
pass)
        {
                return Content(pass);
        }
}
```

ASP.NET MVC Request



Action Result Types

- Specific type (e.g. int, string, Product)
- IActionResult
- ActionResult<T>

```
public Product GetById(int id)
{
    return services.GetProductById(id);
}
```

```
public IActionResult GetById(int id)
{
    var product =
services.GetProductById(id);
    if (product == null)
    {
        return NotFound();
    }

    return Ok(product);
}
```

```
public ActionResult<Product> GetById(int id)
{
    var product = services.GetProductById(id);
    if (product == null)
    {
        return NotFound();
    }

    return Ok(product);
}
```

Action Results Sets

- Producing the HTML
- Redirecting the Users
- Returning Files
- Content Results
- Returning Errors and HTTP Codes
- Security Related Results

Action Results

- The `OkResult` (short method: `Ok()`) return the 200 OK status code.
- The `CreatedResult` (short method: `Created()`) returns 201 Created with a URL to the created resource.
- The `NoContentResult` (short method: `NoContent()`) returns a 204 No Content status code, indicating that the server successfully processed the request, but that there is nothing to return.

```
public IActionResult OkResult()
{
  return Ok();
}
```

```
public IActionResult CreatedResult()
{
  return Created("http://net.org/item",
    new { name = "testitem" });
}
```

```
public IActionResult NoContentResult()
{
  return NoContent();
}
```

Action Results

- The `BadRequestResult` (short method: `BadRequest()`) return 400 Bad Request, which indicates that the server cannot process the request due to an error in said request.
- The `NotFoundResult` (short method: `NotFound()`) returns the 404 Not Found status code, indicating that the requested resource, for whatever reason, was not found on the server.

```
public IActionResult BadRequestResult()
{
  return BadRequest();
}
```

```
public IActionResult NotFoundResult()
{
  return NotFound();
}
```

Action Results

- The above status code results do not cover all the possible HTTP status codes (of which there are many). For situations in which you need to return a status code which isn't given a dedicated action result, we can use the generic `StatusCodeResult` (short method: `StatusCode()`).
- The basic `FileResult` class (short method: `File()`) returns a file at a given path.

```
public IActionResult StatusCodeResult(int statusCode)
{
  return StatusCode(statusCode);
}
```

```
public IActionResult FileResult()
{
  return File("~/downloads/pdfsample.pdf", "application/pdf");
}
```

Action Results

- Possibly the most basic Result class in all of ASP.NET Core MVC is the `ViewResult` class (short method: `View()`), which returns a view.
- You can easily return JSON content from your application by using the `JsonResult` class (short method: `Json()`).
- … or you can use the general `ContentResult` object (short method: `Content()`) to return your content.

```
public IActionResult Index()
{
  return View();
}
```

```
public IActionResult JsonResult()
{
  return Json(new { message = "This is a JSON result", n = 42 });
}
```

Predefined Action Result Types

Type	Description
ContentResult	Sends raw text content (not necessarily HTML) to the browser
EmptyResult	Sends no content to the browser
FileContentResult	Sends the content of a file to the browser. The content of the file is expressed as a byte array
FileStreamResult	Sends the content of a file to the browser. The content of the file is represented through a Stream object
LocalRedirectResult	Sends an HTTP 302 response code to the browser to redirect the browser to the specified URL local to the current site. It only accepts a relative URL
JsonResult	Sends a JSON string to the browser. The ExecuteResult method of this class sets the content type to JSON and invokes the JavaScript serializer to serialize any provided managed object to JSON
NotFoundResult	Returns a 404 status code
PartialViewResult	Sends HTML content to the browser that represents a fragment of the whole page view
PhysicalFileResult	Sends the content of a file to the browser. The file is identified via its path and content type
RedirectResult	Sends an HTTP 302 response code to the browser to redirect the browser to the specified URL
RedirectToActionResult	Like RedirectResult, it sends an HTTP 302 code to the browser and the new URL to navigate to. The URL is built based on action/controller pair
RedirectToRouteResult	Like RedirectResult, it sends an HTTP 302 code to the browser and the new URL to navigate to. The URL is built based on a route name
StatusCodeResult	Returns the specified status code
ViewComponentResult	Sends HTML content to the browser taken from a view component
ViewResult	Sends HTML content to the browser that represents a full page view
VirtualFileResult	Sends the content of a file to the browser. The file is identified via its virtual path

Security-Related Action Result Types

Type	Description
ChallengeResult	Returns a 401 status code (unauthorized) and redirects to the configured access denied path. Returning an instance of this type or explicitly calling challenge methods of the framework has the same effect
ForbidResult	Returns a 403 status code (forbidden) and redirects to the configured access denied path. Returning an instance of this type or explicitly calling forbid methods of the framework has the same effect
SignInResult	Signs the user in. Returning an instance of this type or explicitly calling sign-in methods of the framework has the same effect
SignOutResult	Signs the user out. Returning an instance of this type or explicitly calling sign-out methods of the framework has the same effect
UnauthorizedResult	Just returns a 401 status code (unauthorized) without taking any further action

Action Parameters

- ASP.NET MVC maps the data from the HTTP request to action parameters in few ways:
  - Routing engine can pass parameters to actions
    - http://localhost/Users/NikolayIT
    - Routing pattern: Users/{username}
  - URL query string can contains parameters
    - /Users/ByUsername?username=NikolayIT
  - HTTP post data can also contain parameters

Posting data to controllers

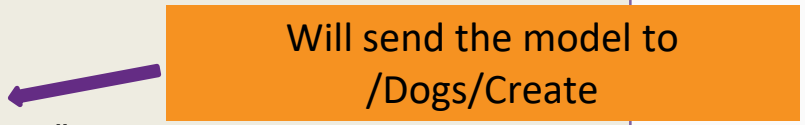
Person
FirstName=James
LastName=Smith

```
[HttpPost]
public IActionResult GetName(Person person)
```

Post data – View side

- **The model specified in the view will reach the controller specified in form element**

```
@model Models.Dog

<h2>Create</h2>
<h4>Dog</h4>
<hr />

<form asp-action="Create" asp-controller="Dogs">
    <label asp-for="Name" class="control-label"></label>
    <input asp-for="Name" class="form-control" />
    <span asp-validation-for="Name" class="text-danger"></span>
</form>
```



Post data – Controller side

- The action method in the controller accepts the values posted from the view.
- The view form fields must match the same names in the controller.

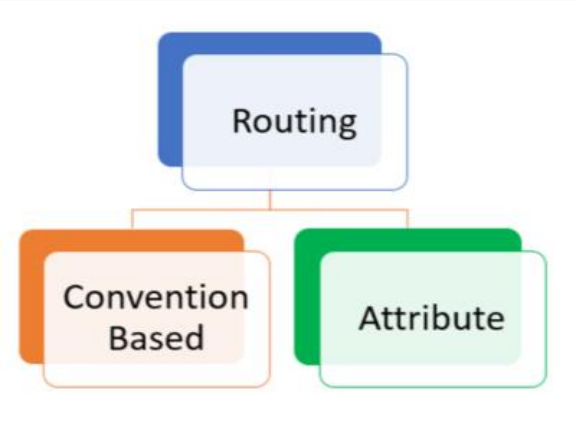
```
[HttpPost]
public IActionResult Create(Dog model)
{
   if (ModelState.IsValid)
   {
     // save the dog in database
     return RedirectToAction("List");
   }

  return View(model);
}
```

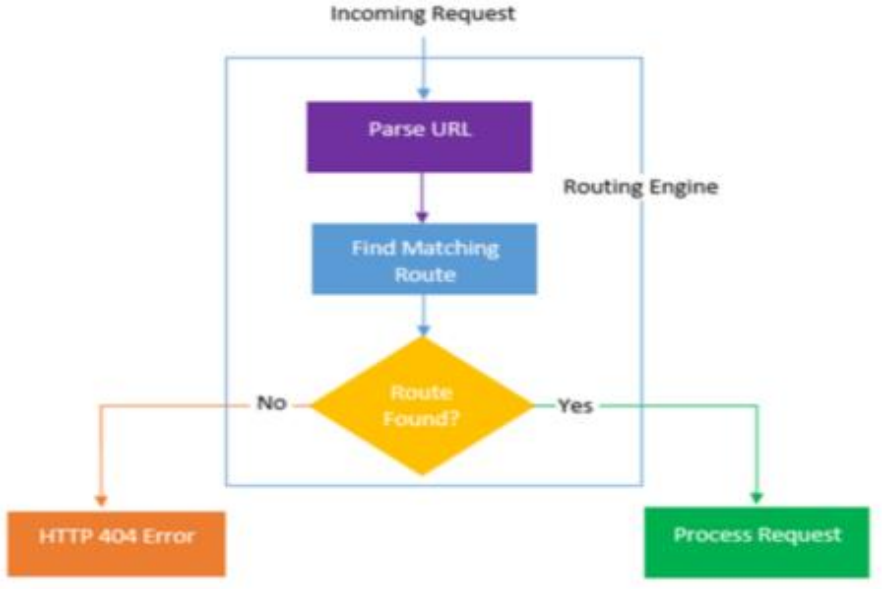
Model passed from the View

Routing

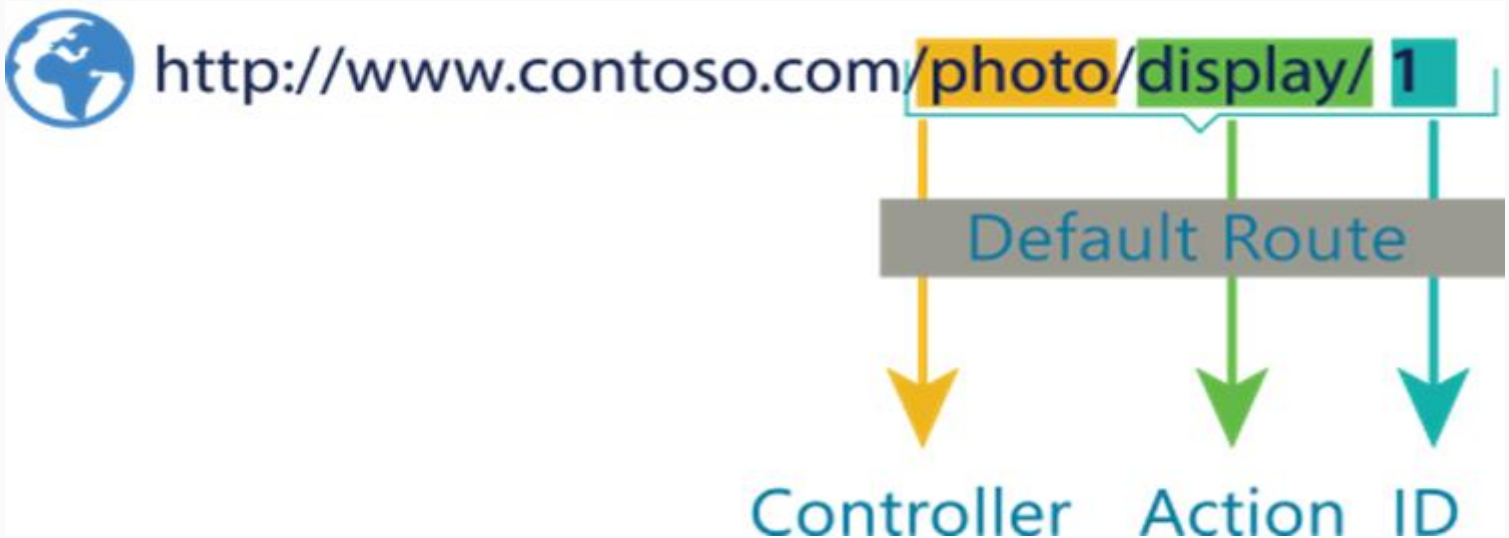
Routing

- **The Routing module is responsible for mapping incoming browser requests to particular MVC controller actions.**





Routing


http://www.contoso.com/photo/display/ 1
Default Route
Controller
Action
ID

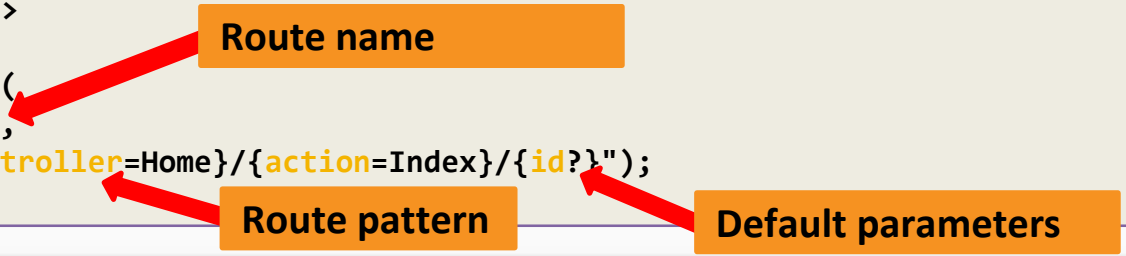
Convention Based Routing

- **Setup places:**
  - **Startup.cs file**

```
app.UseMvcWithDefaultRoute();
```

```
http://www.mysite.com/Products/ById/3
// controller = Products
// action = ById
// id = 3
```

```
app.UseMvc(routes =>
{
    routes.MapRoute(
    name: "default",
    template: "{controller=Home}/{action=Index}/{id?}");
});
```



Attribute Routing

```
public class ProductController : Controller
{
   [Route("")]
   [Route("Product")]
   [Route("Product/Index")]
   public IActionResult Index()
   {
      return View();
   }
}
```

- Attribute routing with Http[Verb]

```
[HttpGet("/products/{id}")]
public IActionResult GetProducts(int id)
{
   // ...
}
```

- Attribute routing tokens

```
[HttpGet("[controller]/[action]/{id}")]
public IActionResult Edit(int id)
{
   // ...
}
```

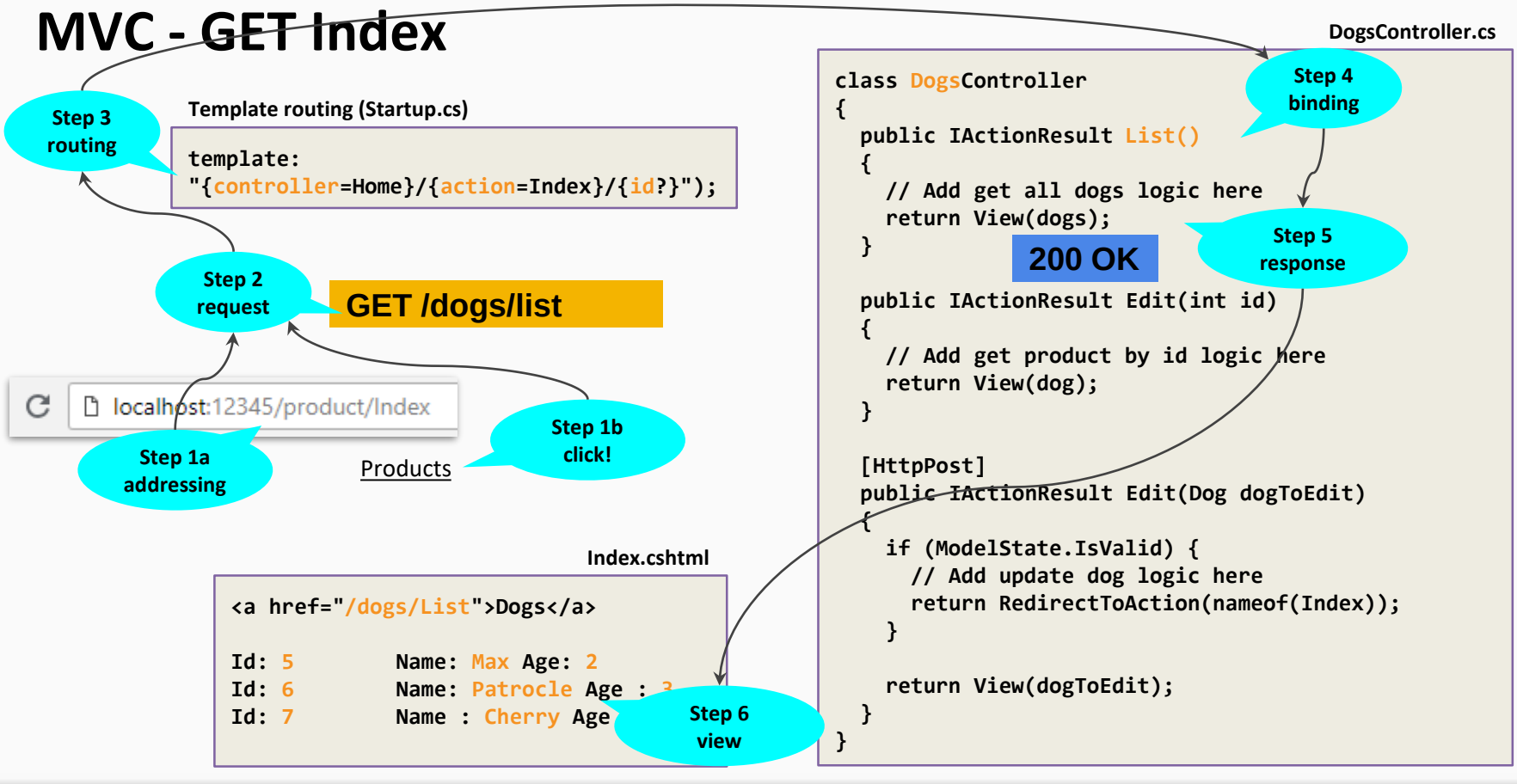
Attribute Routing Constraints

alpha - Matches uppercase or lowercase Latin alphabet characters (a-z, A-Z)
bool - Matches a Boolean value.
datetime - Matches a DateTime value.
decimal - Matches a decimal value.
double - Matches a 64-bit floating-point value.
float - Matches a 32-bit floating-point value.
guid - Matches a GUID value.
int - Matches a 32-bit integer value.
length - Matches a string with the specified length or within a specified range of lengths.
long - Matches a 64-bit integer value.
max - Matches an integer with a maximum value.
maxlength - Matches a string with a maximum length.
min - Matches an integer with a minimum value.
minlength - Matches a string with a minimum length.
range - Matches an integer within a range of values.
regex - Matches a regular expression.

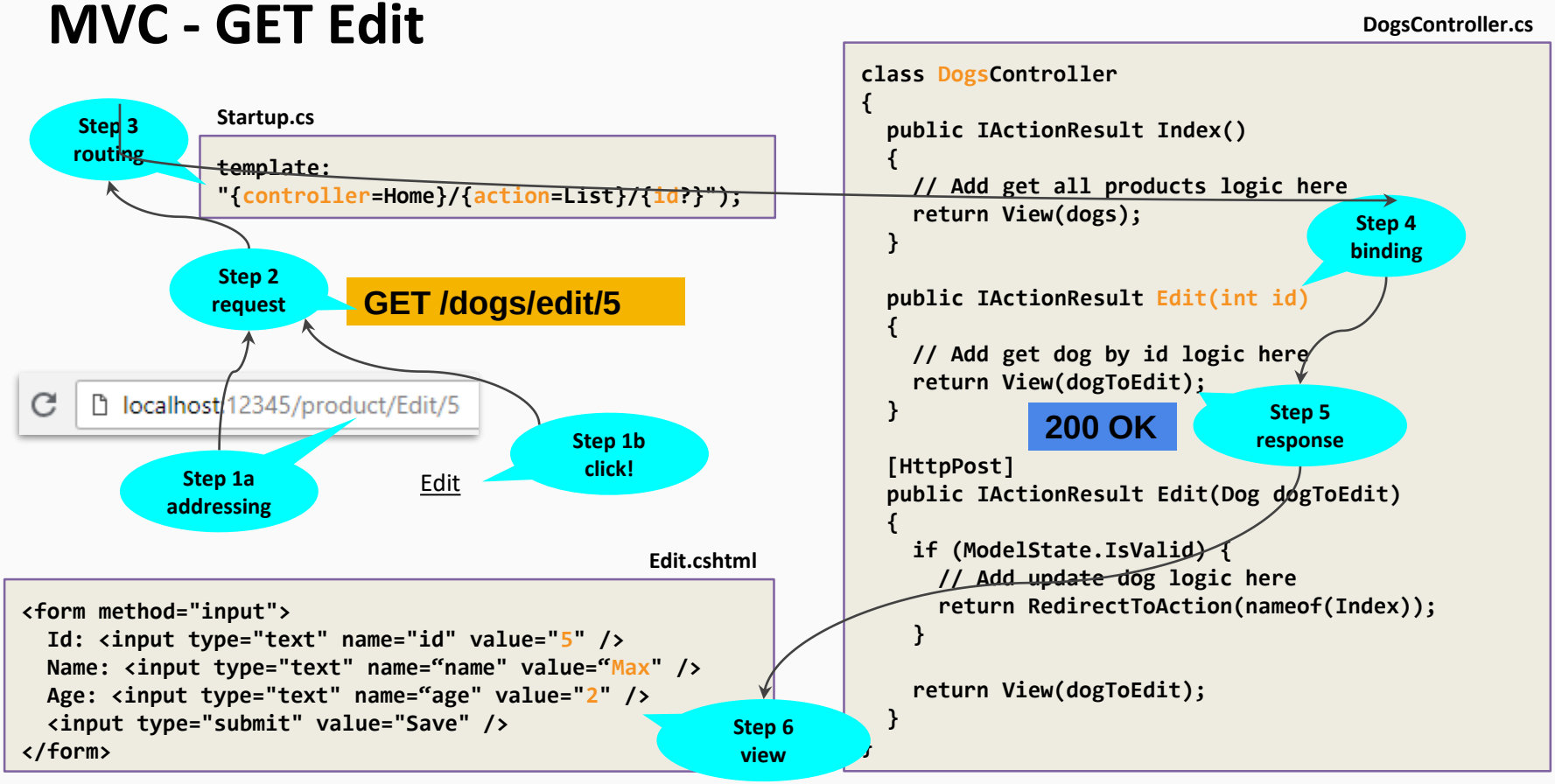
```
// combining multiple constraints using colon (:)

"/{id:alpha:minlength(6)?}"
```

Post- Redirect- Get

MVC - GET Index



MVC - GET Edit


Step 3
routing
Startup.cs
template:
"{controller=Home}/{action=List}/{id?}");
Step 2
request
GET /dogs/edit/5
localhost:12345/product/Edit/5
Step 1a
addressing
Edit
Step 1b
click!
DogsController.cs
class DogsController
{
public IActionResult Index()
{
// Add get all products logic here
return View(dogs);
}
public IActionResult Edit(int id)
{
// Add get dog by id logic here
return View(dogToEdit);
}
Step 4
binding
200 OK
Step 5
response
[HttpPost]
public IActionResult Edit(Dog dogToEdit)
{
if (ModelState.IsValid) {
// Add update dog logic here
return RedirectToAction(nameof(Index));
}
return View(dogToEdit);
}
Edit.cshtml
<form method="input">
Id: <input type="text" name="id" value="5" />
Name: <input type="text" name="name" value="Max" />
Age: <input type="text" name="age" value="2" />
<input type="submit" value="Save" />
</form>
Step 6
view

MVC - POST-Redirect-GET Pattern