

- Partial view, Tag Helpers
- Filter :- Action Filter, Result Filter
- Session
- ADO :- Connected, Ef
- Web API
- Web API with React integration.

[MVC Core Life Cycle]

Client
Browser

Home/Index

IIS / kernal

80

map with Routes
Collection

Split URL :-

```
[ "controller": "HomeController",  
  "action": "Index" ]
```

load Microsoft.AspNetCore.
Mvc.dll

```
ControllerFactory fact = new ControllerFactory();  
Controller ctr = fact.GetController("HC");
```

HTTP Response
Packet

Header Content Type
text/html

Pure
HTML

Model
Binder
class

View Engine class
Obj. Render
process Razor
syntax + HTML

```
Emp emp = new Emp() {  
    Name = " ", Add = " " };
```

```
.Invoke("action", ctr)  
{ "action"  
  ctr  
  para emp.
```

```
// call given to View Engine class  
: ViewName: "Index", model = List{Emp}
```

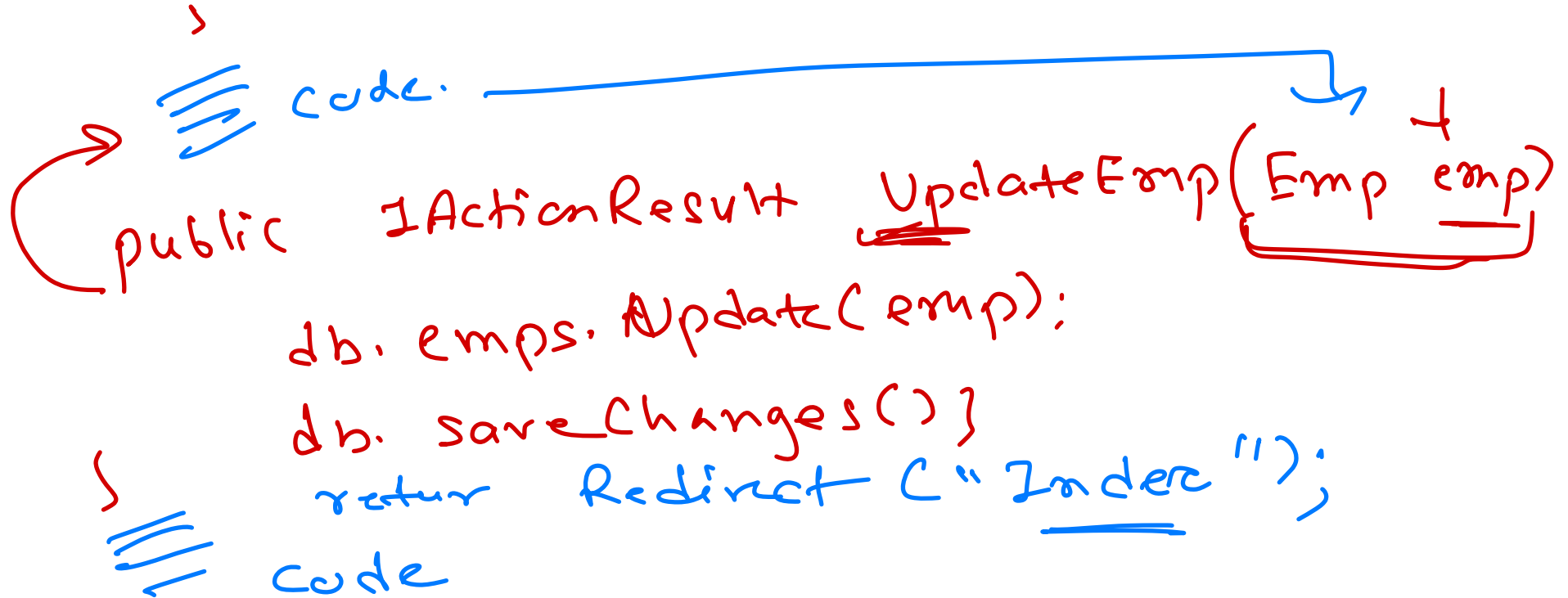
Home
Controller

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

code.

```
public IActionResult UpdateEmp(Emp emp)  
{  
    db.emps.Update(emp);  
    db.SaveChanges();  
    return Redirect("<u>Index</u>");  
}
```

code



return View ("~/views/Home/Index.cshtml");

Request Packet

HomeUpdate
<u>Body</u>
Name = A]
Addr = NYC

<input type="text", id="txt1",
name="Name" />

<input type="text" id="txt2",
name="Address" />

body Data
FormsCollection
class
data = ["Name": "A]",
~~data~~ "Address": "NYC"]

ModelBinder binder = new ModelBinder();

binder → Emp emp = new Emp();

{ emp.Name = data.Name, emp.Addr }

index.html

@model :- which Model class obj / collection we are receiving in content with this current View

@using FullyQualifiedNamespace resolve,

Model (either Emp or Cust)

hence we can resolve

model.name belongs to which Model Class.

Hence only one @model directive is allowed per .cshtml View

Broz of Single Model approach → we design ViewModel classes with properties of emps and custs types.

Partial view

Index.cshtml

div

<partial view>

/div

Demo.cshtml

div

<partial view>

/div

Gallery.cshtml

div

<partial view>

/div



<table>

<form>

- partial view.cshtml

Client Home\Index → (80) 123 / vestrav

1. Map Dir
2. Map Route Collection
3. Split the URL
4. Load CLR, M. AspNetCore.Mvc.dll
5. Get Controller using factory.
6. Get Model Binder factory
7. Action Filter: OnActionExecuting()
8. InvokeMember()
- 8.1: GetViewEngine factory()
9. OnActionExecuted(): AF

Razor View Engine

1. -ViewStart: Layout
2. -ViewImports: Model
3. Result Filter:
 - ↳ OnResultExecuting()
 - ↳ Render - -Layout.cshtml
 indent.cshtml
 - ↳ Output: - html, css, JS
4. OnResultExecuted()

Add Scoped()

DB App. → DbContext

m1() → m2()

U1 →

U2 →

DbContext

m1()

m2()

Add Singleton()

LoggerApp. (SingleLoggerObj.)

U1 →

m1()

m2()

U2 →

m1()

m2()

Add transient()

Email Service App. (Obj)

U1 →

m1()

m2()

U2 →

m1()

m2()

obj₁

obj₂

obj₃

obj₄

DI Container:-

A tool that manages the creation, configuration and lifetimes of the object and their dependencies.

Client → ⑧ IIS | kestrel

HTTP Pipeline
(req, resp, URL)

- Route collection : RouteData
- **DI Container**

Singleton | Scoped | Transient

Your application object along with all dependencies.

Service: Controller, View Engine, Filter
DbContext, ISpellchecker, ILogger, Session

- app. : Configure Middlewares
e.g) Static Files

- app.Run()

- Response Redirect
- Controller factory
- Controller obj
- on Action Executing
- ModelBinder
- InvokeAction ("action")

- onAction Executed ()

- ViewEngine obj
- obj. FindView ("viewName");
- onResultingExecuting ()
: Layout.

→ obj. Render ()

This method creates the blend
of html + C# code [html, CS, JS,
XML, JSON]

→ Response obj.

- Added to HTTPResponseObj → sent this
to client

Final
Response

→ onResultExecuted ()
↳ manipulates ResponseObj. async.

Client

Sends
HTTP Response
packet to
client.