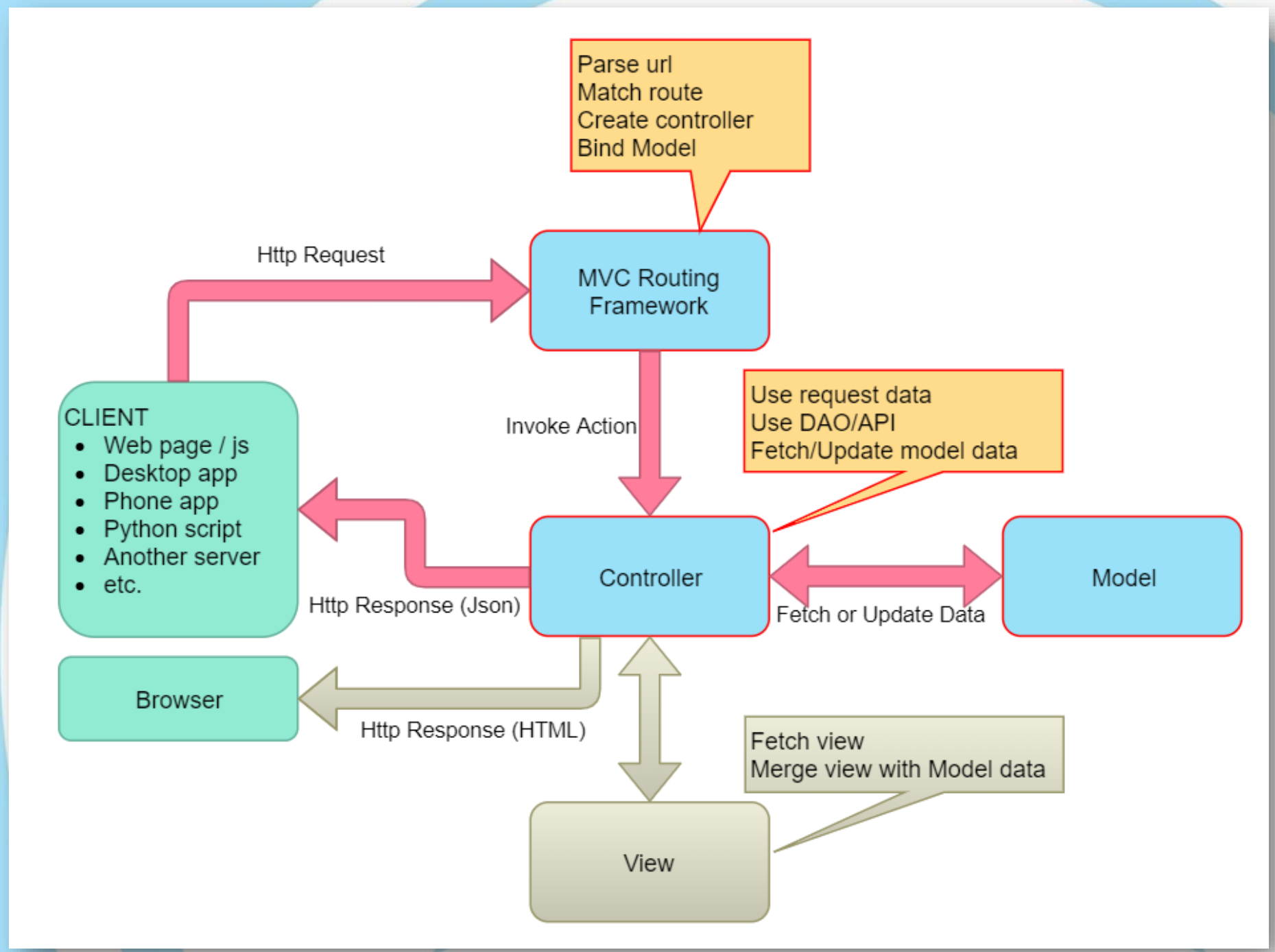
Module 2 Day 14

Creating APIs - Part 2

MVC Workflow



Client-Server Demo

- Azure data center (US-East-2)
- Multiple client computers and types of clients

IActionResult

- Actions may return *IActionResult*
- IActionResult can return a status and content
- Types of IActionResult:
  - OkResult, NotFoundResult, CreatedResult, ForbidResult, StatusCodeResult
  - FileContentResult, FileStreamResult
  - JsonResult
  - And many more

Form Data Validation

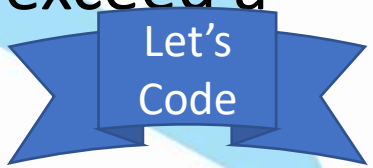
- Client-side validation
  - Ensures data sent to server is good; if not, no data is sent
  - Tells the user there is an issue before data is sent to the server
  - Implemented using HTML5 controls and/or JavaScript
  - Provides a good user experience, **nice to have**
  - Can be bypassed
- Server-side validation
  - Implemented on server, after data is sent
  - Last layer of protection for the data, **must have**
  - Cannot be bypassed

Server-side Data Validation

- Model
  - Model defines the “rules” for valid data
  - [Attributes] are used to declare the rules
- The MVC Framework validates the data
  - Data is validated before the Controller is called
  - Error result is returned if model data is not valid
- *System.ComponentModel.DataAnnotations* namespace
- https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.dataannotations?view=netcore-2.2

Validation – The Model – Data Annotations

- [CreditCard]: Validates that the property has a credit card format
- [Compare]: Validates that two properties in a model match
- [EmailAddress]: Validates that the property has an email format
- [Phone]: Validates that the property has a telephone number format
- [Range]: Validates that the property value falls within a specified range
- [RegularExpression]: Validates that the property value matches a specified regular expression
- [Required]: Validates that the field is not null.
- [StringLength]: Validates that a string property value doesn't exceed a specified length limit
- [Url]: Validates that the property has a URL format



Dependency Injection

- Don’t let this blow your mind!
- Inversion of Control pattern
- Removes the task of creating DAO’s in the Controllers
- Tells the framework to create and pass DAO’s into the controller
- Why?
  - Our class is dependent upon an *interface*, not an *implementation*
  - We can write an in-memory version of the DAO while DB is being developed
  - We can switch later from SQL Server to Oracle, or to a no-SQL implementation
  - We can pass Mock DAO's in for testing