1. fdHTML = HyperText Markup Language
   1. Allows programs to **STRUCTURE** content on a website using tags.
      1. **Tags** =
   2. HTML5 is the current version
   3. Basic Anatomy of a webpage:
      1. <!DOCTYPE>
      2. <HTML>
         1. <HEAD><HEAD>
         2. <BODY><BODY>
      3. <HTML>
   4. **Elements** 🡺 root element 🡺 Node = content between tags + tags
      1. Elements are fundamental to HTML functionality
2. CSS = Cascading Style Sheets
   1. Complementary to HTML and provides the **AESTHETIC** (style) to HTML webpages
      1. Properties
         1. Styling precedence based on “ways to style”: Inline > Internal > External > Browser default
         2. 3 ways to style (in order from best practice to ‘worst’)
         3. External = includes a reference to a .css file inside the <head>
            1. Separation of concerns by separating styling from structure
            2. Reusability
            3. Centralization of styling
            4. Improves developer readability by reducing cluttering
         4. Internal = styling is defined in a style element inside the <head>
         5. Inline = styling is applied to each element via a **style attritbute**
      2. Str
         1. Adds style and flavor to the HTML pages
      3. Lim
   2. Syntax
      1. Selector
      2. Declaration
      3. Properties
      4. Value
      5. Statements
   3. Selectors
      1. Properties
      2. Str
      3. Lim
   4. Inheritance
      1. Properties
      2. Str
      3. Lim
   5. Colors
      1. Properties
      2. Str
      3. Lim
   6. Units of Measurement
      1. Properties
      2. Str
      3. Lim
   7. Specificity
      1. Properties
      2. Str
      3. Lim
   8. Pseudo
      1. PSelectors
      2. PClasses = keyword at the end of a CSS selector to specify a style *iff* the element is in a certain state
         1. Properties
            1. 30+ pseudo classes
         2. Str
            1. Allows for more precise styling dependent on business needs
         3. Lim
      3. PElements = styles specific parts of an element (for example the first line of a paragraph)
         1. Properties
         2. Str
         3. Lim
   9. Combinators = describes the relationship between selectors in a CSS statement
      1. Properties
         1. 4 types of combinators
            1. Descendant
            2. Child
            3. Sibling
            4. General Sibling
      2. Str
      3. Lim
   10. Box Model
       1. Properties
       2. Str
       3. Lim
3. MVC = An architectural pattern that creates applications with a 3 layer design: Model – View - Controller
   1. Properties
      1. Models = the virtual entities that exist in the application as stand-ins for the Db
         1. Model classes use validation logic to enforce **business rules** about the data – typically retrieving and storing model state
      2. View = *components*of the UI that users interact with – depending on the model data it “illustrates”
      3. **Controller** = the centerpiece of the structure that controls the flow of data (the entry point of the application)
         1. Do not have logic – think of gateways, but no control over when they open or close
         2. Have different information that flows through:
            1. Browser requests (HTTP requests)
            2. Model state data
            3. View templates for certain model states
      4. Conceptually, V C are different layers but in the same project
         1. M is a conceptual layer that is a functional amalgamation of the business logic, models, and context (Db access)/repo
   2. Separations of concerns =
      1. Properties
         1. Applications should be built with loosely coupled layers that collaborate to achieve specialized concerns
         2. Utilizes **layers** – which cannot communicate beyond the scope of their neighbor
      2. Str
         1. Allows for loosely coupled code
         2. Not unique to .NET – means that robust understanding of MVC can allow you to understand program design architecture of MVC in other languages
      3. Lim
         1. A separate architectural design from other common patterns:
   3. Repository layer
   4. Repository pattern
      1. S-N: “pattern” = “approach”. So repo pattern means repo approach, which is just sticking everything into its own repos
   5. Decoupling
   6. Model = A class that holds public properties that is a template for DB entities/represents the state of the application and any business logic.
      1. The model is the central component of the MVC pattern
      2. Models are connected to **Views** and **Controllers** via **model binding:**
         1. “Converts **client request data** into objects that controllers can handle by importing taking the data as parameters to action methods”
   7. Model Binding
      1. Properties
         1. Process achieved by:
            1. Retrieves data from various sources such as route data, form fields, and query strings
            2. Provides the data to controllers and Razor pages in method parameters and public properties
            3. Converts string data to .NET types
            4. Updates properties of complex types
         2. Model binding executes by using **targeting**
            1. **Targets** are just things that the method binding process searches for – usually parameters for the controller action method or parameters for Razor Pages handler method
         3. Contains two types of logic:
            1. Implementation logic
            2. Business logic
      2. Str
      3. Lim
   8. View = responsible for presenting content to an interface (These are the html files that will display the views)
      1. Properties
         1. Use the Razor view engine to embed .NET code in HTML markup
            1. This interaction creates the .
         2. Any logic, if at all, should be related only to presenting content
            1. If logic is necessary to display complex data, ViewModels should be used
         3. Main VIEW folder 🡺 directories
      2. Str
      3. Lim
   9. Controller = a CLASS that contains logically similar groups of Action methods (and is responsible for initial processing of a request ANDModel instantiation - HOW?)
      1. Properties
         1. Anything that satisfies one of the following conditions is a Controller:
            1. Class name suffixed w/ “Controller” (means it was likely made with a VS Studio Controller template
            2. Inherits from a subclass of the ControllerBase class
            3. Has the [Controller] **attribute**
         2. Should have minimal logic that interacts with Model layer and View layer
         3. Should follow the **Explicit Dependencies Principle**
         4. Invokes action methods if the HTTP endpoint matches the incoming URL
            1. Default routing logic is: Broad 🡺specific
            2. (Domain?)/[Controller]/[ActionName]/[Parameters]
            3. DCAP

Graphical user interface, text, application, email

Description automatically generated

* + - 1. See above for a reference on Method binding
      2. Access to 3 main categories of helper methods:
         1. Empty Response body
         2. Non-empty response body with predefined content type
         3. Non-empty response body with client-defined content type
    1. Str
    2. Lim
  1. ASP.NET MVC = **HIGHLY TESTABLE**
     1. Properties
        1. Should have minimal logic that interacts with Model layer and View layer
        2. S-N: Recall that each aspect of MVC is a layer and already “compartmentalized out” for you
     2. Str
        1. **HIGHLY TESTABLE**
        2. Easy to update compared to other patterns (Example:)
        3. A patterns-based way to ensure separation of concerns
     3. Lim
  2. Routing =
     1. Routing logic
        1. Attribute routing
        2. Conventional routing
  3. Attribute
     1. Properties
     2. Str
     3. Lim
  4. Action Method = a method in a controller that handles requests
     1. Properties
        1. All public methods in a Controller are Action (unless they have [NonAction] attribute
        2. Parameters are bound to/correspond to request data and validated using model binding
           1. Anything model-bound is model-validated
        3. Should contain logic to link a request to a business concern
        4. CAN return anything, BUT usually return **IActionResult or Task<IActionResult>** (this one for async methods)
        5. S-N: “\_context” is an *injected* version of the context variable type
           1. Text

              Description automatically generated
     2. Str
        1. Follows MVC: the model binder is taking in an object as an input, instead of instantiating an object by intaking object properties as a parameter
           1. If the action method is the one instantiating, it contains more complex logic – and may need additional complex logic, depending on the object that needs to be instantiated
     3. Lim
  5. Model Validation
  6. Explicit Dependencies Principle
  7. Code First
     1. Str
        1. More straightforward & easier for the application to manage
     2. Lim
        1. Limited real-world application
  8. Logging
  9. Debugging
  10. HP verbs
  11. TDD
  12. Mapping means putting the FK attribute –
      1. If you’re mapping & using FK it doesn’t make the FK an FK just saying “the values are here”
      2. Basically it’s like carrerra – depending on the context, it means different things
  13. Lecture Notes [6/18]
      + 1. 1) I get that the Model is making the entity template. Why would we need
        2. a PlayerBase and PlayerDerived entities? Because Derived inherits.
        3. 2) One of the docssaid: "Model binding converts client request data into
        4. objects for controllers".Model binding just taking input parameters in the URL
        5. and invoking a CreateModel Action method?
        6. 3) When if ever, does the referencing order matter? Never matters
      1. EF only uses/accesses public (since it has to push stuff to Db)
      2. Dependency injection injects into the constructor of the dependent class
         1. The \_context is only usable/readable when the class is instantiated
         2. Graphical user interface, text, application, chat or text message

            Description automatically generated
         3. Don’t need to put Type name = new Type(); because in the startup class (in the configuration services method, you create a constructor that instantiates the db context)