**MVC Week**

**HTML –** HTML is the text content and structure of a webpage. Separate from CSS and JS.

**Element:** A piece of HTML. Denoted by tag(s).

**Attribute:** Name="value" pair inside the tag of an element. Additional properties that define the characteristics of an element.

**Doctype:** DTD. Specifies the version of HTML. In the file, given before the html starts.

**HTML 5:** Standard for html constantly being developed.

**Common Elements (list, structural, etc):** Unordered (bulleted) and ordered (numbered) lists. Blocks like Div and inline like span. Etc.

**HTML5 –**

**Audio and Video:** New in HTML5. Added to prevent plugin dependency. Give codex options (like mp3, wav) and browser picks one.

**Canvas:** New in HTML5. Added to prevent flash plugin dependency.

**New Elements:** HTML5 added new premade structural elements, like section, header, footer, and nav.

**Accessibility:** Added accessibility attributes, like ARIA.

**Forms** – (Ways to get data from view/user to controller/server)

**Inputs:** The type of form it is. For example, submit button, select (dropdown) list or text box.

**Validation:** Two types, client- and server-side. Data Annotations can apply client-side validation to given form values. Server checks model state. Give anti-forgery token and validate it later to check for cross site request forgery (CSRF).

**GET:** Form method. Adds values to the end of the URL in the query string.

**POST:** Form method. Adds values to the form as key-value pairs.

**CSS –** Cascading style sheets are for visual appearance of html on a webpage

**Rule:** A selector and declaration(s). Selector targets html elements. Declarations set property values.

**Property:** The name of a set of values that define something in css. For example, font-size or background-color.

**Cascade:** The way the applied rule is decided when multiple rules set one property. Importance (bad practice), then specificity [Inline over internal/external; more ids, then classes, then tags], then latest rule in source order.

**Box Model:** Space of an html element. Margin, Border, Padding, Content. Each has a top, right, bottom, and left.

**Inheritance:** Children can inherit properties from parent. Which ones are inherited by default generally make sense. Customizable.

**Media Query**: Check if browser has some default, and if so then apply some property.

**Positioning:** Where an element is positioned. Can be static (order in document), which is the default, relative (offset from its static pos), absolute (offset from a non-static ancestor), or fixed (offset to screen, doesn't change position as you scroll).

**Responsive Design:** 1 css file that responds to changes in size of page (screen resolution). As opposed to adaptive design, which adapts to a given resolution by switching to a more appropriate css file.

**External/internal/inline:** Linked css in head (best practice), in between style tags within head, set style attribute in element's tag.

**Selectors:** Used to target html in a css rule. For example, .classname or #idname.

**Bootstrap:** Uses 12 column grid. Parts of it are used by many companies. CSS and JS?

**MVC –** ModelViewController design pattern for separation of concerns

**Model:** Object-oriented data.

**View:** Display logic. Dependent on model.

**Controller:** Receives request and constructs response. Combines view with model. Dependent on view and model.

**ViewModel:** Display focused version of object data.

**HTTP Request Lifecycle:** Browser gives URL to Domain Name Service (DNS) and gets back IP address. Sends HTTP request to server with that IP address. Server sends back HTTP response containing html.

**ASP.NET MVC**

**Controller –**

**Actions:** Methods in controller called to handle requests. Responds with a view.

**HTTP Verbs:** Methods for RESTful. GET and POST (as well as put, patch, and delete).

**Model –**

**Model Binding:** Automatic assigning of query string and form fields to parameters of the action method in the controller.

**Data Annotations:** Can be used for client-side validation.

**View –**

**Razer:** Starts with @. Allows C# expression. Will be converted to html for view.

**Layout:** Shared between webpages. (Partial view cannot have @Layout page reference.)

**@model (strongly and weakly typed):** Makes view strongly typed.

**ViewData:** Key-value pair dictionary. Passes data to view. Goes away after request. Linked to viewbag? .

**ViewBag:** Passes data to view. Dynamically typed. Linked to viewdata ? .

**Dynamic:** Can add new properties but turns off compile-time type checking.

**TempData:** Key-value pair dictionary. Passes data to view. Values survive to next request. Default uses cookies. Regular key access marks for deletion. Can access w/o marking for deletion with peek. Can unmark for deletion with keep.

**Routing –** Determining the correct controller and action to use for a request.

**Convention:** Global routing. Give controller and action to use for matching routes (urls). Defaults will be used if not given. Order matters (first match called). Route name must be unique. Can have route parameters in the matching route.

**Attribute:** Prevents convention-based routing. Controller and each action get a route attribute (can have multiple). Can specify route parameters in the route attribute.

**Route Parameter:** Parameters for action method given in route. Often used for identification.

**Query String:** Portion of URL after the "?". Can be placed by GET.

**Validation –** Ensuring correct input from user. Both client and server side.

**Server:** Check ifModel state is valid.

**Client:** Give data annotations to fields. Can be bypassed.

**ModelState:**

**CSRF:** Cross-Site Request Forgery. Fake form given to server.

**Anti-forgery token:** Given in form in response to request. When form received, validate the token.

**Testing –**

**Mock:** Controlled simulation of objects in order to test the behavior of some other object.

**BDD:** Removing as much implementation detail as possible from the test, so that you only test the intended behavior. (Increment counter test should not assume a new counter starts at 0.)

**Filters –** Code run before or after stage in request processing pipeline.

**Authorization:** Check if user is authorized for this request.

**Resource:** Check if response is cached.

**Exception:** Catch any unhandled exceptions that have occurred.

**(then model binding)**

**Action:** Happens before and after the action (controller method) execution (manipulate argument and result). Not in Razor Pages.

**Result:** (if action successful) Happens before and after result (view) execution.

**Custom:** Can create custom filers. (useful for factoring out common logic across actions)

**Helpers –**

**HTML:** Simplifies/combine difficult/multiple html into one prepackaged statement. Looks like Razor. HTML helpers can be created. (Bind model object to html.)

**tag:** Shorthand for adding extra stuff to targeted html tag. (Help server code render html by targeting html tags.)

**custom:** Can make custom.

**common ones -**

**Html:**

DisplayFor

DisplayNameFor

EditorFor

TextBoxFor

ActionLink

DropDownListFor

**Tag:**

a (anchor)

asp-action

asp-controller

asp-route

asp-route-paramname: Gives value for route parameter

input asp-for

select asp-for asp-items

label asp-for

(span) asp-validation-for

form asp-action

**ASP.NET + EF design:**

**Code-first EF:** Design objects first, then make database.

**Migrations:** Use built models to generate database.

**Dependency Injection –**

**Singleton:** One object for all requests.

**Scoped:** One object per request.

**Transient:** One object per use (multi per request).

**FromServices:** Action parameter attribute to request a service (w/o constructor injection).

**DevOps Week**

**SDLC –** Software Development LifeCycle.

**Waterfall:** Gather requirements, design, build, test, deploy. Takes time. Not adaptable. More accountability (procedures) and security (separation).

**Iterative:** Break large app into features that are worked on in shorter, repeated cycles (iteration).

**Agile:** Ideas/practices for more responsiveness to changing requirements/user feedback during software development.

**Scrum:** Divide intouser stories, which are individually run through the stages of development.

**Kanban:** Continuous planning and reevaluation.Columns (stages) have max number of effort points that can be contained within them, to limit work in progress.Helps see bottlenecks coming.

**Scrum –**

**Sprint:** 2-3 weeks of work. Starts with sprint planning.

**Planning:** Formalize user stories of sprint w/effort points. Can incorporate velocity info feedback from previous sprints.

**User Story:** Functionality from user point of view.

**Effort:** Fibonacci number estimating difficulty of a user story.

**Capacity:** Total productive work hours/effort points available (don't count lunch, etc.).

**Velocity:** Sum of effort point completed during sprint.

**Scrum board:** Place to organize all user stories and their stage of development, with notes about dependencies, implementation, and assigned workers, etc.

**Git –**

**branching:** Should separate code that's being worked on from working code. Can have separate branches for separate features.

**merging:** Integrate separate code back together. Will make a new merge commit. If conflicts, merger needs to resolve them.

**pull request:**

**conflict resolution:** Changes in same file may cause problem, merger needs to decide what is correct.

**DevOps –**

**Motivation:** Break barrier between development (new features) and operation (stability) teams.

**Values:** Faster code into production.

**Continuous Integration:** Developer code that passes automated checks should frequently be integrated with other's code. (Pass build/compile pipeline before merging)

**Continuous Delivery:** Automated deployment with some manual approvals before production.

**Continuous Deployment:** 100% automated all the way to production.

**Build Pipeline:** Way to automate restore, build, test, analyze, package/publish and deploy.

**Azure DevOps -**

**Pipelines:**

**Boards:**

**Pipelines –**

**Build definition:**

**Task:**

**Parameter:**

**Service Connection:**

**Release Definition:**

**Designer/Yaml:**

**Static Analysis –**

**SonarQube/Cloud:** Open source static analysis tool. Can examine code quality (detect bugs, security holes, unreliable or unmaintainable code).

**Configuration:**

**Quality Gate:** Pass/fail Boolean thresholds.

**Quality Profile:** Rules, often language specific definitions of bugs?, that can be triggered.

**Technical Debt:** Idea that sloppy code now needs to be fixed later. Amount of time to fix.

**Code Smell:** Things your quality profile has flagged as less maintainable.

**Code Coverage:** Amount of your code that is being tested.

**Cloud Computing –**

**Public:** Shared.

**Private:** Just you.

**Hybrid:** Some on-premises resources and some cloud.

**Multi:** More than one provider.

**IaaS:** Infrastructure as a Service. Abstract physical concerns.

**PaaS:** Platform as a Service. Abstract everything but code.

**SaaS:** Software as a Service. Abstract but the app.

**CaaS:** Container as a Service.

**Axure services –**

**VM:** IaaS.

**SQL DB:** PaaS.

**AppService:** PaaS.

**Storage:** IaaS.

**Docker -**

**Daemon:** Persistent process that manages containers.

**CLI:** Commands start with docker. Use to build images and run containers, as well as see images and containers in your system.

**Image:** Template to start container. Has filesystem and pointer to run some program. Built upon a base image.

**Container:** The app and its dependencies packaged together.

**Virtualization:**

**Docker Compose:** YAML file for configuring services (dependencies).

**Dockerfile –**

**FROM:** Specify base image.

**RUN:** Runs given command when building the image. **CMD:** Argument passed to entrypoint. Runs when container starts.  
**ENTRYPOINT:** Set command and parameters to execute first when running container.  
**EXPOSE:** Notify of open port. That's all. **COPY:** Move files from outside into image. **WORKDIR:** Sets current working directory. Creates missing directories. **ARG:** Declare arguments that are used in FROM.