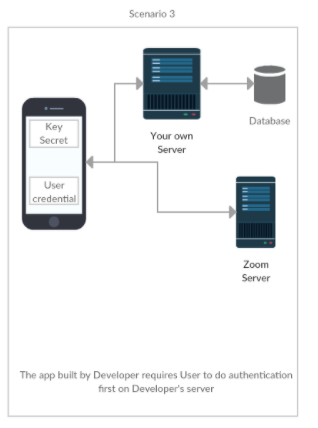
**Virtual Classroom Dashboard 🡪 Research Documentation:**

**01/03/2021 – 01/09/2021 🡪 Week 1**

**Zoom API SDK Client:** [**https://marketplace.zoom.us/docs/sdk/custom/introduction**](https://marketplace.zoom.us/docs/sdk/custom/introduction)

* Zoom API uses Zoom access tokens: investigate Access Credentials for zoom.
* Can build modules to enable meetings within a web browser.
* Access Credentials: SDK key & Secret.
* Registering and activating any Basic Zoom account will automatically provide free-trial Developer access to the Zoom API and SDKs.
* Use free trial to test the services and SDK functionality
* User Tokens and Zoom Access Tokens are required to start a meeting on behalf of a [Non-login user](https://marketplace.zoom.us/docs/sdk/native-sdks/user-login). These dual tokens are required for additional layers of security.
* Web SDK is authenticated using an API key and Secret instead of SDK.
* For Web SDK: [*Create a JWT App*](https://marketplace.zoom.us/docs/guides/getting-started/app-types/create-jwt-app)*on the Marketplace.*
* User Tokens: used to start meetings for users.
* Requesting user tokens need to send GET requests with a userID to /users/{userId}/token
* UserID is either User API or user email.
* More on Access Credentials: <https://marketplace.zoom.us/docs/sdk/native-sdks/credentials>
* Login User/SSO User – Each person needs school credentials if university or personal credentials.
* Login SSO is more beneficial User Type
* Connection type:



* How to Integrate with web App: <https://marketplace.zoom.us/docs/sdk/native-sdks/web>
* Chrome, Firefox, Edge all work with all the features within Zoom. Safari and Internet Explorer have limitations.

**AWS vs Azure hosting:** [**https://stackify.com/azure-vs-aws-comparison/**](https://stackify.com/azure-vs-aws-comparison/) **&** [**https://insanelab.com/blog/web-development/microsoft-azure-vs-amazon-web-services/**](https://insanelab.com/blog/web-development/microsoft-azure-vs-amazon-web-services/)

* AWS categorizations: content delivery and storage, compute, networking, and database.
* Azure categorizations: data management and databases, compute, networking, and performance.
* Amazon includes identity and security services such as key storage and active directory.
* Amazon also includes AWS Config, Cloudtrail, and Cloudwatch.
* Azure includes security and management tools such as Active Directory, Azure Active Directory, Multi-Factor Auth, and Azure monitoring and performance tweaks.
* Azure offers an easy-to-use Hybrid clouds and substantial support.
* Azure: Windows Server, SQL Server, Exchange, etc..
* Azure makes simple deployment for .Net apps.
* AWS is great for .Net as well but only if a certain AWS feature is needed.
* Since .NET is easier to integrate with Azure prob best to use Azure.

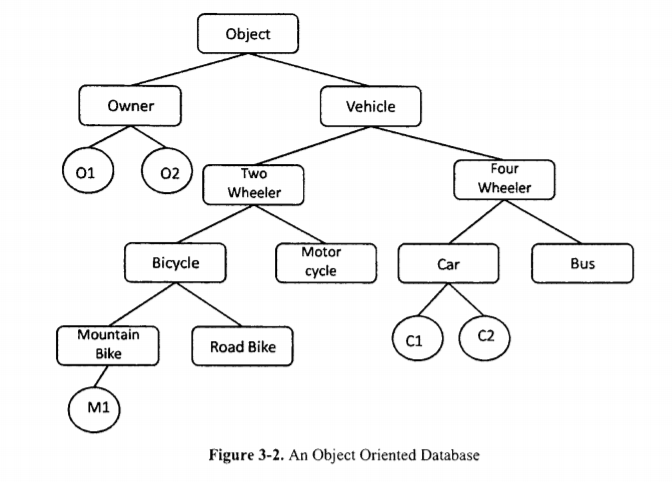
**Implementing large scale databases: LinkedIn Learning video 🡪 Database Foundations Core Concepts & Elmasri, Ramez, and Shamkant Navathe. *Fundamentals of Database Systems*. 7th ed., Pearson, 2015 &** [**https://www.freecodecamp.org/news/database-indexing-at-a-glance-bb50809d48bd/**](https://www.freecodecamp.org/news/database-indexing-at-a-glance-bb50809d48bd/) **&** **Self-adjusting multi-granularity locking protocol for object-oriented databases**

* Databases are highly structured data files that allow data input, organization, and retrieval.
* We use tables for the storing, sorting, and filtering of the data.
* A database is just a structured data storage system
* DBMS 🡪 structural rules, data entry, and data protection
* Retrieving data 🡪 creates one or more indexes, queries data
* Managing Databases 🡪 DBMS supports highly structed and efficient data
* Flat File DB 🡪 2-D tables, rows, and columns, stored as delimited text files
* Flat File Structure 🡪 rows store records(individual items), columns store attributes.
* A simple data structure that follows basic organizational principles.
* Hierarchical Databases 🡪 consists of tables that are related by some piece of data.
* Limitations are when they need to display more complicated relationships.
* Relational Databases 🡪 No restrictions and is a combination of flat file and hierarchical databases.
* Uses unique identifiers keys: Primary, foreign, candidate, and super keys
* Foreign keys reference primary keys in another table.
* Database Fundamentals🡪 data types (Consistent data is entered): an attribute must only have a single data type, Constraints(no dup values, rules, default values, NULL values, etc) What are the rules of the data, Referential Integrity 🡪 Ensures the validity and completeness of the data, SQL 🡪 language that is used to create structures, update, modify, delete data
* Microsoft uses T-SQL which is just their version of SQL
* Database Server 🡪 dedicated or virtual machine
* Users log in over a network and commands are used remotely, processed, and the results are returned by the DBMS
* Server Hardware 🡪 extremely fast r & w disk speeds, large amounts of memory, and fast network connections
* Multiple server machines tired together in clusters and physical proximity not needed.
* SQL Server Management Studios(SSMS) – connect to the instance of SQL server
* Views 🡪 Multiple tables data no actual data storage.
* “A proper index can be created only when you know exactly what your query & data access patterns look like”(free code camp)
* Indexing maps search keys that correspond to data on disks by using in memory & on disk data structures.
* Possible keys 🡪 what all available indices are there
* Key column 🡪 which index is going to be used
* In ANSI SQL standards 🡪 PK’s should be comparable
* Define id field as AUTO INCREMENT
* EXTENDED 🡪 shows all indices not usable
* Differences between key and index 🡪 Key: constraint on the behavior of the column & index: special data structure that facilitates data.
* Non\_unique🡪 1 not unique, 0 is unique
* Key\_name 🡪 name of index, PK is always PRIMARY
* Seq\_in\_index 🡪 sequence # of col in index, if multiple columns are in the index them, they will be assigned based on how they were ordered during the creation.
* Collation 🡪 how col is sorted with the index. A ascending, D descending, NULL not sorted
* Cardinality 🡪 est. # of unique values in index. More Cardinality: higher chance query optimizer will pick index for queries
* Sub\_part 🡪 index prefix, NULL if the entire col is indexed. Otherwise it shows the # of indexed bytes if partially indexed.
* Packed 🡪 how key is packed, NULL if not
* NULL 🡪 YES if may contain null values and blank if not
* Index\_type 🡪 which indexing structure is used: BTREE, HASH, RTREE, FULLTEXT
* Clustered Index 🡪 collocated with data in the same table or same disk file & can be a BTREE whose leaf nodes are the actual data blocks on disk.
* This kind of index physically organizes the data on disk as per the logical order of the index key.
* Physical data organization 🡪 data is organized on disks across thousands or more of disk/data blocks.
* Cluster indexes do not require that all the disk blocks are contagiously stored.
* Advantage of Clustered Indexes 🡪 faster, possibility data is buffered in memory.
* Reduces # of disk IO by collocating related data as much as possible causing improvement in performance.
* If queries are based on PK’s performance will be faster.
* Constraints with Clustered indexes 🡪 impacts physical organization of data so only one clustered index per table is allowed.
* Structure of PK Index(Clustered) 🡪 indexes are maintained as B+ Tree on disk & in memory and stored in blocks on disks.’
* Advantages of Primary Index 🡪 locality of fata can be provided & any query that uses PK is faster.
* Disadvantages 🡪 since uses direct reference to data blocks through virtual addressing space and dick blocks are physically organized, every time the OS does some disk page split due to DML operations(SELECT, UPDATE, DELETE), the P index needs to be updated.
* DML operations places pressure on the performance of the p index.
* Secondary indexing 🡪 any indexing but clustered indexing. Does not impact physical storage locations.
* EX of Query: CREATE INDEX secondaryIndexName ON tableName(col)
* Structure of secondary index 🡪 also uses B+ Tree and is sorted as per the key. Leaf nodes contain the primary index.
* Secondary index references primary index
* Secondary Index 🡪 Primary Index 🡪 Data Blocks
* Advantages 🡪 can create infinite secondary indices. (Not recommended to use a lot, only what is required)
* Disadvantages 🡪 DML operations(DELETE & INSERT), secondary index also needs to be updates.
* Secondary indices can essentially create issues.
* UNIQUE key index 🡪 can contain null values and are similar to primary indices.
* Composite index 🡪 defines indices on multiple columns(MAX = 16)
* Columns using composite indices are concatenated together and are stored in sorted order in a B+ Tree.
* When would you use a composite index?
* If certain fields appear together in multiple queries
* If there is a high cardinality
* Covering Index 🡪 a type of composite index
* SELECT & WHERE are part of composite index
* Indexing Guidelines 🡪 indices take up more memory so use only the number of indices you need, write operations are costly with indices, Cardinality is important, and indices require maintenance.
* OODBMS 🡪 Object-Oriented Database Management System: supports modeling and creation of data as objects
* Uses Objects(“Real world entities”), Object identifiers(“unique identifier associated with the object”), Class(“objects are instances of a class and classes consist of methods and attributes”), and Class Inheritance(“use of subclasses that belong to one superclass – the subclasses inherit attributes from the superclass.”)
* Concurrency Control 🡪 Updates of data must be done in concurrent executions and failures
* A 🡪 atomicity: effects of transactions are reflected in the database or none are
* C 🡪 consistency ensures the state of the database is consistently maintained before and after transactions
* I 🡪 isolation: concurrent transactions are isolated from each other
* D 🡪 durability: no transactions are lost after system failures
* Concurrency Control in RDB 🡪 responsible for resolving conflicts among transactions
* Serializability guarantee consistency
* “A Lock is a database system object associated with a database object that prevents undesired operations of other transactions by blocking them”.

**01/10/2021 – 01/16/2021** 🡪 **Week 2**

**Database Continuation from previous week: Self-adjusting multi-granularity locking protocol for object-oriented databases**

* Shared and Exclusive locking 🡪 with share a data item can be read but cannot write and with exclusive locks it can both read and write items.
* Deadlock protocol 🡪 2 or more transactions are waiting on each other to finish so there is a protocol to prevent such an event from occurring
* Two-phase locking 🡪 growing and shrinking phases
* Growing: locks accumulated, shrinking locks released
* Rigorous two-phase locking 🡪 locks held until the transactions are committed
* Two-phase locking in general is to help prevent deadlock and is known as a pessimistic method because it requires locks to be acquired before given access to data
* Timestamping is another method that prevents deadlocks where a number is assigned to each transaction before anything is executed.
* Concurrency Control with Object Oriented DB’s 🡪 database functionality combined with OO concepts
* Design & Development of SAML 🡪 SAML: self-adjusting multi-granularity locking protocol improves concurrency control of OODB’s
* Adjusts locking granularity bases on requirements of transaction
* 3 levels of granularity locking: instance lock, class lock, and hierarchy lock
* RDBMS lock hierarchy 🡪 Database – Areas – Relations – Tuples
* Pg. 19 OODBMS



* Explicit lock 🡪 explicitly requested by and granted to transactions directly
* Implicit locks 🡪 need to be acquired for either direction of the hierarchy of the database

**Advanced Threading in C#:** [**https://www.linkedin.com/learning/advanced-threading-in-c-sharp/two-way-signaling?u=52983649**](https://www.linkedin.com/learning/advanced-threading-in-c-sharp/two-way-signaling?u=52983649) **&** [**https://www.tutorialspoint.com/csharp/csharp\_multithreading.htm**](https://www.tutorialspoint.com/csharp/csharp_multithreading.htm)

* Thread 🡪 execution path of a program, can perform multiple jobs at once by creating multiple threads.
* Lightweight processes 🡪 concurrent programming: using threads to save CPU cycle wastage and increase the efficiency of your app.
* Thread life cycle 🡪 starts when an object is created from System.Threading.Thread class and ends when it is terminated.
* The unstarted state: instance of thread is created but the start method has not been called yet.
* The ready state: thread is ready but waiting for CPU cycle
* The not runnable state: thread is not executable, meaning sleep or wait methods have been called or are blocked by the input output operations
* The dead state: thread has completed its cycle or is aborted
* Main thread 🡪 allows for the creation of individual threads for a multi-threaded application.
* The very first thread is the main thread, upon execution of the program the main thread is executed first.
* The threads created after the main are called child threads
* To access the threads, you must call the CurrentThread property of the thread class.
* Refer to the tutorialspoint website for examples, methods, and properties of threads.
* Multithreaded code needs thread safety
* Thread safety 🡪 Code is thread safe if shared data structures are modified to ensure all threads behave properly.
* Items must be locked before making any modifications
* Code is all about predictability
* All tasks run concurrently, and the order doesn’t matter because one task can take precedent over the other
* In the .Net Framework the static methods are thread safe but not all static method that you create are thread safe.
* When you develop a static method make sure you create them to be thread safe
* Thread Affinity 🡪 thread that instantiates an object is the only thread that can access its members
* With this you may not need to use a lock to access objects inside the thread
* Synchronizing among threads is signaling.
* AutoResetEvent 🡪 used when a thread needs exclusive access to a resource, one thread access at a time, automatically closes
* Two-way signaling 🡪 more interaction between the threads.
* ManuelResetEvent 🡪 allows for as many as you need threads to run like a gate that controls the flow of the threads.
* Console.ReadKey() 🡪 reads any key the user presses
* Signaling Countdown 🡪 a construct for synchronization. Until it gets signaled it will be waiting .
* Task Parallel library(TPL) 🡪 System.Threading or System.Threading.Tasks
* Adds parallelism and concurrency to maximize code
* Value is ability to scale degree of concurrency dynamically, handles partitioning of work, scheduling, and canceling
* Handles state management but not all code is suitable
* Threading has small overhead might not help performance
* Parallel is not good for small data sets.
* To cancel use readkey then use cancellationTokenSource.Cancel()
* Interlock provides thread safety between variables used in parallel threading
* PLINQ 🡪 Automate parallelization, declarative not imperative
* Take, select, selectmany, skip, takewhile, skipwhile, elementat prevent query from being parallelized
* Anomalies: join, groupby, groupjoin, distinct, union, intersect, except
* PLINQ can change to sequential if it feels like it is faster than parallelism
* You can force parallelism by calling asparallel()
* PLINQ can be used to parallelize I/O-intensive operations like API and database calls
* ForAll statement 🡪 takes care of merging unlike the foreach loop
* ConcurrentBag 🡪 executes in parallel then sorts the information into order
* If not using concurrent bag when using PLINQ you can use asOrdered()
* Merge Options in PLINQ 🡪 NotBuffered, AutoBuffered, FullyBuffered
* Tasked Asynchronous Patter (TAP) 🡪 Returns Task or Task<TResult>, Uses Async suffix, accepts cancellation token, returns quickly to the caller, frees up the thread if I/O bound
* Do not make a main method async

**ASP.NET Security:** [**https://www.linkedin.com/learning/asp-dot-net-security-2/unhackable-asp-dot-net-applications?u=52983649**](https://www.linkedin.com/learning/asp-dot-net-security-2/unhackable-asp-dot-net-applications?u=52983649)

* Top 10 security risks(OWASP) 🡪 injection, broken authentication, sensitive data exposure, xml external entities, broken access control, security misconfiguration, cross-site scripting, insecure deserialization, using components with known vulnerabilities, and insufficient logging and monitoring.
* All points on the list might not apply to asp.net because it is already built to be secure
* XXS(Cross-site scripting) 🡪 java script injections, html injections.
* Client sends JavaScript to a server and then the server bounces the script back.
* This is dangerous especially for shop applications.
* Using @ or <%=……%> but using @ helps with defense but try not to use @Html.Raw() because it could allow cross-site scripting.
* JavascriptStringEncode(), if using JavaScript, you have to use this just in case of hexacode use
* Sam-Origin Policy 🡪 3 pieces of info that needs to be the same.
* Protocol must use same beginning because of the server so http or https must be the same throughout
* With domain https://www. Or https:// not the same
* Ports: https://.....:4444 not the same as one without a port
* Cross-origin resource sharing 🡪 when using a http request like a get request you cants see the information returned from server but using CORS you can have access to the information
* Enabling CORS 🡪 Web API: [EnableCors(origins: “webname.com”, headers: “\*”, methods: “GET”)]
* [DisableCors()]
* Try to avoid using these but if you do be as specific as possible
* Enable in controller and webApi config files
* SQL injection: ADO.NET 🡪 raw sql can be manipulated by the user, if user supplies a command in an input instead of proper value it could affect your program
* SQL injection: Entity Framework 🡪 Avoid execution of raw SQL

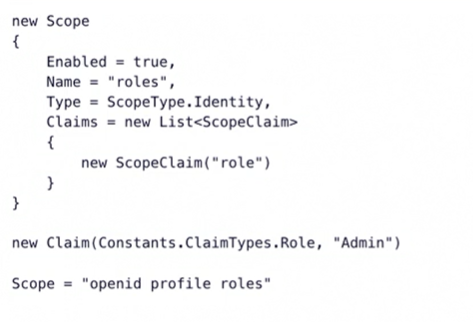


* Return IEnumerable instead of IQueryable : .toList
* Fixing SQL injection 🡪 use parameters or find a simpler option that does not allow SQL injection or only allow for specific input.
* Cross-Site Request Forgery(CSRF) 🡪 using cookies for authentication but can be abused by making the user send HTTP request to attackers’ server and attacker sends a new request.
* Because a cookie was used the attacker could send an authenticated HTTP request.
* Preventing CSRF 🡪 using tokens alongside the request to make sure information what is inside the cookie: AntiForgeryToken(), [HttpPost], [ValidateAntiForgeryToken]
* Must use everywhere necessary in the application(views and controllers)

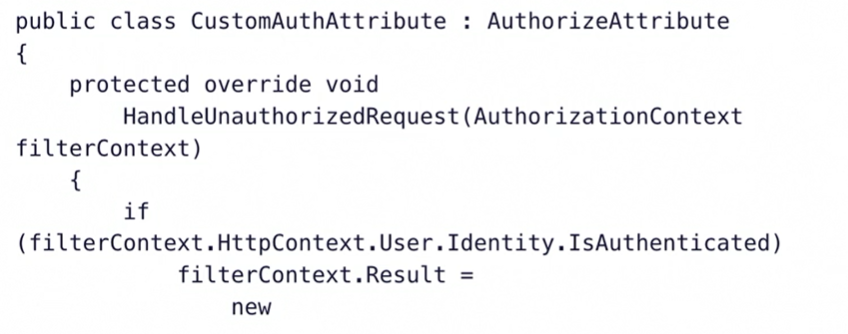
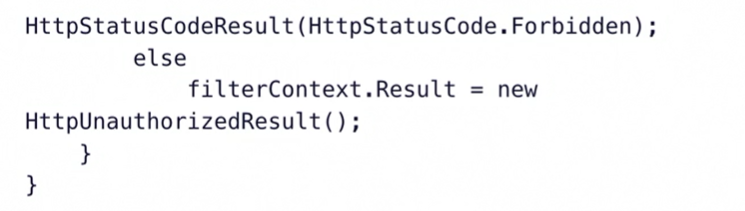
**01/17/2021 – 01/23/2021** 🡪 **Week 3**

**ASP.NET Security: Continuation**

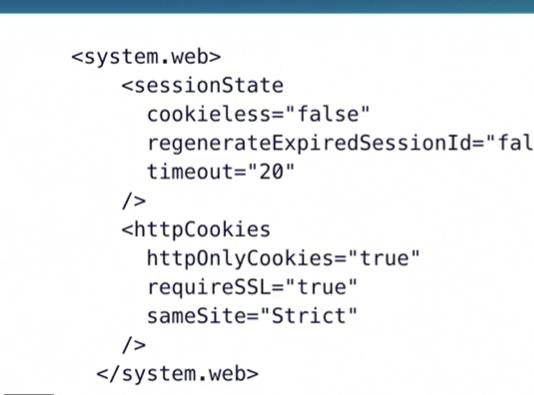
* Storing secrets in Web.config file 🡪 Cannot be downloaded via HTTP, built in protection for files that should not be available(default protection by ASP.NET), still debatable whether it should contain sensitive info (web.config file)
* Externalizing Web.config settings 🡪 use the app settings set file to topsecret.settings.config….<appSettings file = “”><add key =”name” value = “myValue”/> </appSettings>
* <connectionStrings configSource = “Topsecret.connectionstrings.config”> this option does not allow merging
* Encrypting Web.config 🡪 Encrypt: aspnet\_regiis -pe “connectionStrings” -app “app name”, Decrypt: aspnet\_regiis -pd “connectionStrings” -app “app name”: both of these get set in the Administrator: Command prompt
* <system.web> <machineKey validationKey=”058…” decryptionKey=”8A7..” validation = “SHA1’ decryption=”AES” /> </system.web> Place this in the main machine.conflict file
* Azure key Vault 🡪 Set up the key vault, create the key vault in a recource group, use Azure CLI, az keyvault create, az keyvault secret set
* Reading from the key Vault 🡪 var client = new KeyVaultClient(…); & var value = awaitclient. GetSecretAsync(…); small delay to receive information but is stored securely
* Managing the Key Vault with Azure shell 🡪 za group create -- name “name goes here” --location location, az keyvault create --name “name” --resource-group “resource” --location location (May come with a cost)
* Now can store secrets by: az keyvault secret set --vault-name “name of the vault” --name “name of the secret” --value “….”
* To retrieve: az keyvault secret show --vault-name “” --name “”
* Password Hashing🡪 A function cannot be reversed, store number of iterations…hash the info into the database, use Login method and hash helper
* On IdentityServer 🡪 OAuth 2.0 Core – standard for authorization maybe also consider OpenID connect, Identity Server for .Net applications
* To configure IdentityServer🡪 Installation: directly install into visual studio using nuget packet manager(identityServer) | Configuration: Define clients, users, and scopes | Startup: go to start up class and set up IdentityServer there
* Authenticate users within the regular app using cookies to maintain the fact a user is still logged in: Microsoft.Owin.security.Cookies..UseCookieAuthentication, Configure OpenID connect Middleware: Microsofst.owin.Security.OpenIdConnect…UseOpenIdConnectAuthentification()

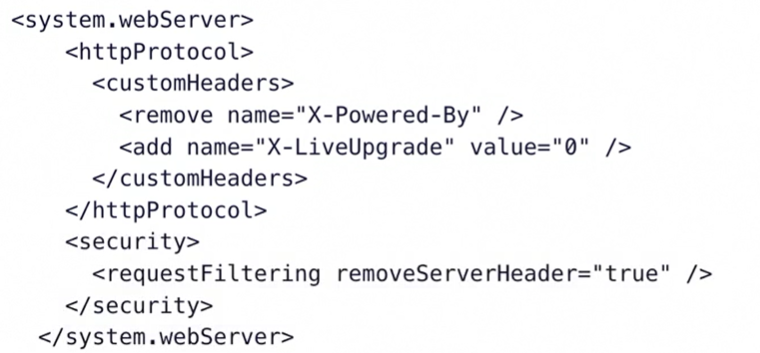
* Custom authorization inheriting from IdentityServer 🡪

* Cookies require extra protection
* Cookie options 🡪 var cookie = new HttpCookie();, cookie.Secure = true;, cookie.HttpOnly = true;, cookie.SameSite = SameSiteMode.Lax;
* Sessions 🡪 virtual concept where if the user does not use the website for a certain amount of time, they get logged out. This uses session ID’s
* Place sessions in the web.config file 🡪

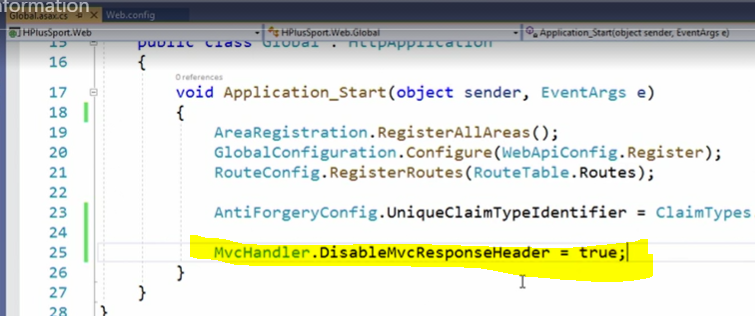


* Enforcing HTTPS 🡪 this is a must: redirect – Check HttpContext.Current.Request.IsSecureConnection(), Rewrite - <system.webServer> <rewrite><rules> …. Both options hare the same weakness but this third approach is best – HSTS HTTP-Strict-Transport-Security this guarantees HTTPS requests only
* Error Handling 🡪 1.Custom Error Handler place in the global aspxfile, 2, Custom Error messages in web.config file
* Removing Revealing Http Headers 🡪 web.config file



* Browser developer tools reveal info that should not be shared so to remove you do the following:

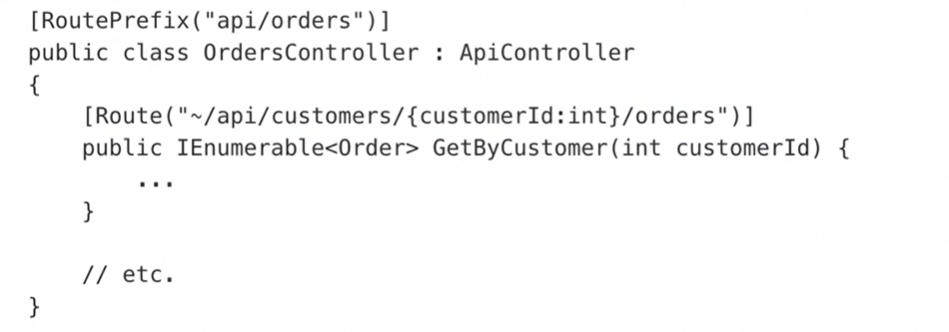




* Security HTTP Headers 🡪 X-FRAME-OPTIONS: can prevent that a page is loading within a iFrame…prevents click jacking, X-XSS-Protection: for browsers without X-XSS-Protectioin but can be used anyway for browser that already have it built in, Content-Security-Policy: important for application to do, Referrer-Policy: tells the browser where the url is coming from(Might not want another website to know that).

**Building Web APIs with ASP.NET Web API 2.2:** [**https://www.linkedin.com/learning/building-web-apis-with-asp-dot-net-web-api-2-2-2/the-power-of-web-apis?u=52983649**](https://www.linkedin.com/learning/building-web-apis-with-asp-dot-net-web-api-2-2-2/the-power-of-web-apis?u=52983649)

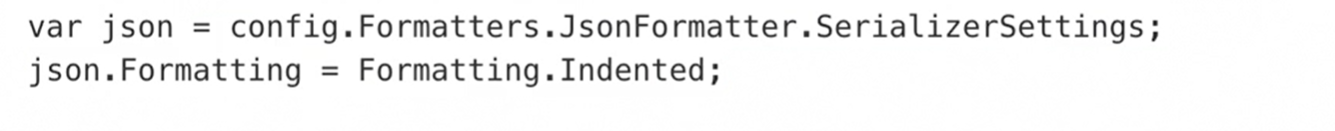
* Convention-based routing🡪routing table: be consistent with your naming conventions, Request URI(GET/api/order) 🡪 Controller Action(OrderController.GetOrders())
* Parameter Binding Conventions🡪Getting data into your api
* Type of value 🡪 Mapped From
* Most common Rest Methods 🡪 Get(), Post(), Put(), Patch(), Delete()
* Dto 🡪 data transfer object
* Parameter Binding Attributes 🡪 Override the source of your parameters
* Attributes ^ 🡪 [FromBody] & [FromUri]
* Http Verb Attributes 🡪 [HttpDelete], [HttpGet], HttpHead], HttpOptions], [HttpPatch], [HttpPost], [HttpPut], & [HttpVerb]
* Return Values 🡪 Output options for data from your SPI
* Returning void(not recommendable), Retuning HttpResponseMessage(Allows to generate custom responses), Return IHttpActionResult(all APIs use, most generic, downfall: not self-documenting), Returning objects or list of objects(Best but does not allow validation returns)
* Ok method will help returning IHTTP action result and query results serialized
* ActionResult Helper Methods 🡪 Ok(), BadRequest(), Conflict(), Content(), Created(), InternalServerError(), NotFound(), Unauthorized(), StatusCode()
* Validating Models 🡪 Ensure only valid data is sent to your APIs
* Validation in Web API 2.2 🡪 System.ComponentModel.DataAnnotations, base.ModelState.IsValid
* Using Formats beside JSON 🡪 Because sometimes you need XML
* XMLSerializer 🡪 older, Opt-out serialization of properties/fields, [XmlIgnore] attribute to opt out
* DataContractSerializer 🡪 Newer, 10% faster, Opt-in serialization of properties/fields, [DataMember] attribute to opt in
* Attribute Routing 🡪 Defining routes inline: configure routes using Map Http routes where it will start looking for the corresponding attributes.
* Use constraints to make routes distinguishable ex: [Route(“api/pagename/{id:int}”)] 🡪 the int part is the constraint.
* Customizing attribute routes 🡪 using consistent route prefixes for all your routes, you can specify the route prefix once in the controller
* Escaping RoutePrefix 🡪



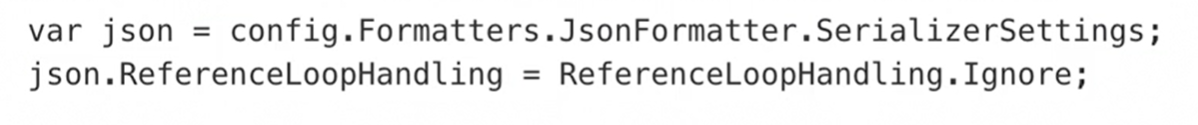
* Constraints 🡪 alpha, bool, datetime, decimal, double, float, guid, int, length, long, max, maxlength, min, minlength, range, regex
* Models and DTOs 🡪 Getting data in and out of your API: Models(Client App 🡨 REST Calls 🡪 API App 🡨 ORM Calls 🡪 DB) 🡨 Change to 🡪(Client App(DTO)🡨 REST Calls 🡪 API App(DTO/Model)🡨 ORM Calls 🡪 DB)
* The change uses less bandwidth resulting in faster calls, using DTOs only sends and receives info you need and not a model that sends everything
* CRUD operations with Entity Framework 🡪 Quickly create, read, update, and delete
* Useful Json.NET Settings 🡪 Configuring JSON serialization: Json.NET is the serialization library
* DateTimeKind 🡪 .Local: current time in the local time to computer, .UTC: universal coordinated time +-0 time zone, .Unspecified: type was never specified
* DateTimeZoneHandling mode 🡪 .Local: converts UTC to local before serializing it, .UTC: converting local date time to UTC before serialization, .Unspecified: converts datetime to local before serializing them and converts date string that are unspecified to local as well, .RoundTripKind: preserves time zone information or using unspecified
* DateFormatHandling 🡪 .IsoDateFormat, .MicrosoftDateFormat
* ContractResolver 🡪 you can replace with camelCaseResolver



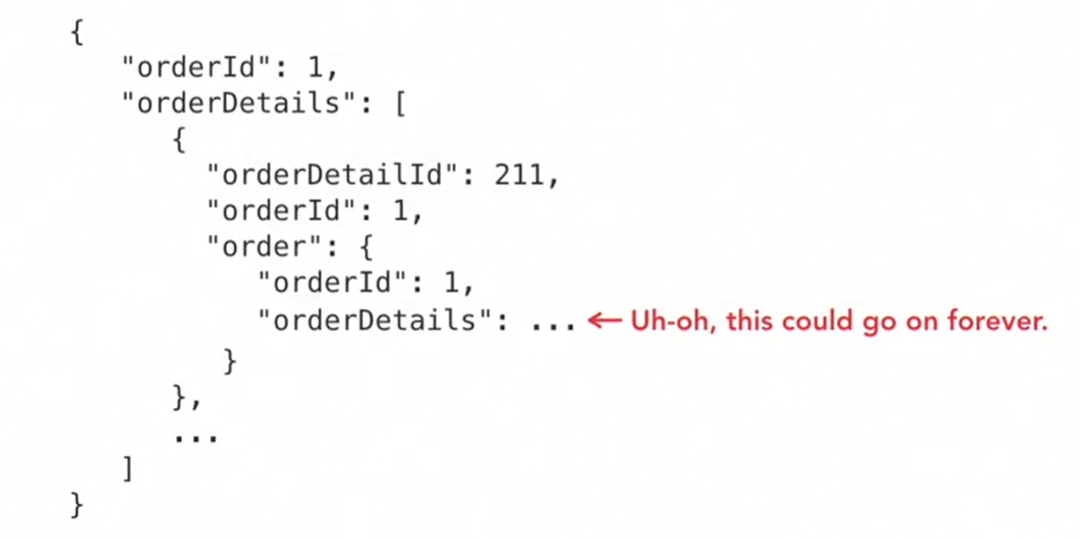
* Formatting to indented:



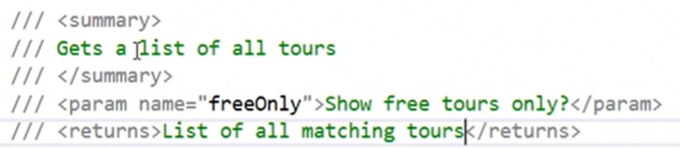
* ReferenceLoopHandling:



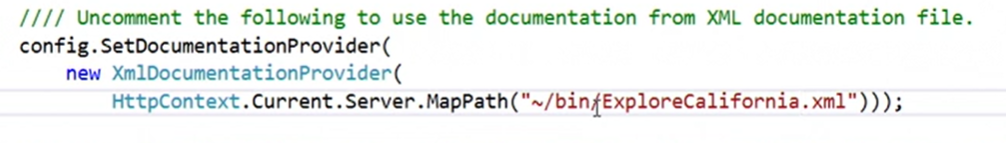
* BE AWARE OF LAZY LOADING
* ReferenceLoopHandling 🡪for JSON.NET



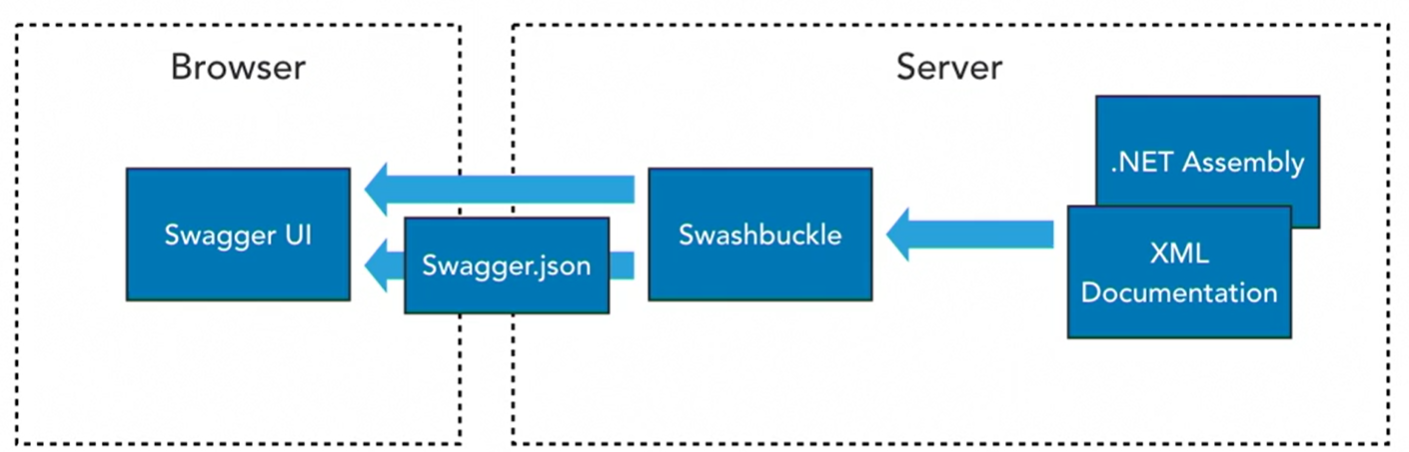
* Allows for .error, .ignore, or .serialize(will serialize forever be careful)
* 4 different ways to handle exceptions 🡪 placing handlers inside controllers, Using exception filters, exception loggers, global exception handler.
* Compiler directive 🡪 tells he program to compile certain sections of the code
* You can create handlers globally or specifically for a single method by creating a handler class
* Documenting with Help Pages 🡪 Auto-generate docs with Web API help pages



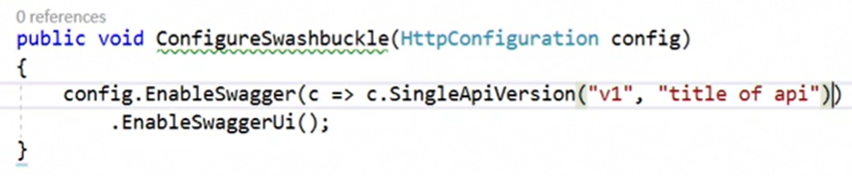
* Property pages 🡪 Build 🡪 Output 🡪 check XML documentation file 🡪 on NuGet Pages download Microsoft.AspNet.WebApi.HelpPage
* Replace from the whole project GlobalConfiguration.Configuration with Startup.HttpConfig

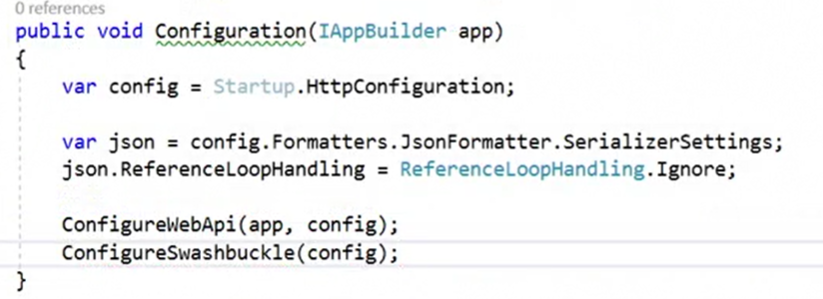


* Documenting with Swashbuckle and Swagger 🡪 API documentation frameworks



* Nuget package that allows for the use of swagger is swashbuckle.
* Om projects use swashbuckler.core
* In startup.cs 🡪





* Adding the XML comments into ConfigureSwashbuckle 🡪



* API Testing Client Postman 🡪 Test, mock, build, and monitor your APIs