Data usually in a DB. Accessin gthe DB can be slow (in many cases on a different machine).

Can do two things:

* Cache data in memory on the server. For small apps, this could be the server which executes the app.. but when we switch to load balancing.. we will have to move the **memory cache to another machine.**
* Cache data on the client. The client can then request the data, but attach a **tag** to the request, indicating **when the data was last fetched**. The server can then inform the client the data hasn‘t changed by returning **HTTP 304**, or if the data has changed.. return it with **HTTP 200.**

**ETags**

Etag is basically a **unique identifier** for web caching validation. It is a **unique key** generated at the server. This key **represents a resource (URL)**, if the **RESOURCE CHANGES => NEW ETAG FOR THAT RESOURCE**.

Let‘s say we make a request to /api/courses/{id}/students which we haven‘t done.

**Get results + Etag**.

If we issue the same request, by including the **„If-None-Match“** header with the **Etag value**..

The server will compare the header Etag with the resource requested.. **IF ETAG MATCHES.. 304 (NOT MODIFIED) with no content..**

On the other hand, if we wanted to issue a PUT/PATCH request we would include the „If-Match“ header, which returns **412 IF NO MATCH**.. meaning the data has changed.

--Request-->

🡨 Etag and results—

--If-None-Match: Etag🡪

304 if Etag is same for resource.. otherwise new Etag and results..

CacheOutput:

* ClientTImeSpan (CacheControl MaxAge HTTP Header)
* MustRevalidate (indicates whether origin server requires revalidaiton of a cache entry on any subsequent use when the cache entry becomes stale)
* ServerTimeSpan.. how long cached on server side..

Anyway.. you decorate methods with [CacheOutput(ClientTimeSpan = 30, ServerTimeSpan = 30) i.e. CACHE THE ACTION FOR 30 SECS on both client and server..