**.Net Core**

API

→ Just Razor

→ Razor Class Library is new

* used for repeated use

Web application(razor pages)

Angular

React.js

React.js and Redux

Includes its own web server

Giant http listener → **Kestrel**

* web server built into core
* Github project available as Nuget package
* included as default builder

|  |
| --- |
| WebHost.CreateDefaultBuilder(args) .useStartup<startup>(); |

**Full framework**

Web API

Everything hosted in IIS (has to be windows)

Razor is pluggable, can unplug and use dif engine

Old MVC, **should we update them to Core**?

* **No**, too much work and too many differences

**Dependency Injection**

→ data objects make for easy transfer from Sql to oracle, etc.

**EF** was completely separate from Framework(Nuget)

→ with Core, it's now built in (using statements)

**New .Net Core WebApp "CoreDemo"**

→ In Solution Explorer

* wwwroot all **static** files → no c# program files
  + *these are the files that can be directly accessed by browser*
* Home controller made for us → default index, privacy, and default error view
* Appsetting.Json → older web.config(XML → being replaced by json)
  + Don't want to embed connection string in case that changes, so it goes in this stand alone file
  + Files that are likely to change

For this→ Products and Customers to demonstrate MVC

**Full Framework → 3 ways to EF**

* Code first → automatic
* Database first → typically v1 exist, now we're doing v2 / choosing db
* Design first → don't have a DBA / creates database for us after we build

**Core EF**

* Code first → (+PowerShell (migration/update)) ← What we're doing
* Database first (+**.dotnet.exe**) - .Net helper is cl prompt tool
  + There's a command prompt for VS → list all commands to see what it can do
  + Ability to build sql database after code first project

Project

**→ New model for custome**r → Add class

**Dif between field & property**

* public int ID; ← Not a prop, a field!
* propFull snippet → properties have {get;set;}
* Interfaces don't know what fields are -- need prop

\*Forcing Intellisense → CTRL + J

→ **Add properties**

* CustomerId, Firstname, Lastname → add constraints/attributes for sql
  + We'll come back and add error messages for these attributes

[DataType(DataType.EmailAddress)] ← razor

[Required(ErrorMessage="Please Enter a Name.")]

[Range(0,400)]

[RegularExpression(".+\\!.+\\..+")]

W/ DataTypes → Razor does the rest for us (writes HTML for email address form)

pg 128 for DataTypes

**→ Add Product Class**

(When security is an issue → Products & Customers are too obvious of names)

After these tables → **create DbContext** → in Models

* Make it inherit from DBContext

→ Not tables → Entities (DbSet) - GenericClass of <Products>

*Full Framework would include the code for the connection strings for us*

*With Core,*

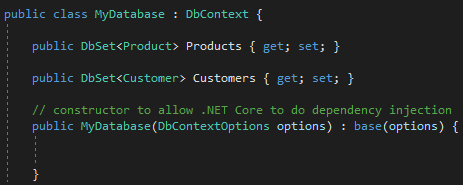
If there was a sudden change from Sql to MySql they would be able to Inject these compatible classes they want to use with a constructor

**1.Add constructor**

// constructor to allow .NET Core to do *dependency injection*

**public MyDatabase(DbContextOptions options) : base(options) {**

**}**

****

**2. Edit StartupFile**

ConfigureServices() ← add services to our project (anything I need to use in multiple places)

* cookie policy is stored here instead of every view we have of this
* AddMVC() is built in here → this adds 260 services (including support for routing, razor views)

→ Add using Micrsoft.EntityFrameworkCore;

→ modify line where connection string is(**bad idea to hard code it)**→ should go in appsettings.json → **we're going to hard code it anyway**

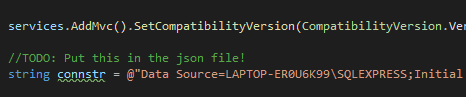
In the ConfigureServices method **add your connection string.**

**after this** → services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_2);

→ add connSt**r** = → *changed previous db used to current one*

"Data Source=LAPTOP-ER0U6K99\SQLEXPRESS;Initial Catalog=**CoreDemo**;Integrated Security=True;**(Everything after this is default and not necessary to input)**Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"

**TODO:** keyword comment that **Task list** will keep track of



**HACK:** keyword to represent that code works,

but it isn't pretty and shouldnt be deployed this way

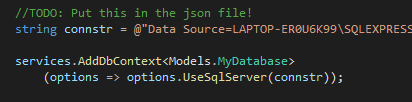
/// in front of a known structure (class)

→ inserts this where you can summarize it &

vs pulls if you need documentation



→ **Add services.AddDbContext** under connection string



→ Then go to **package manager console** in tools

*\*\*PowerShell\*\**

* add-migration InitialCreate



*warning some things will not be stored the same way (differences in char bytes from sql)*

→ we could go back and specify precision of decimal

→ or cast {get;set;} to come back however you want them to

**EF Fluent API**

→ Instead of using attributes, you go out and write code

* for when you can't find an attribute for something

→ now update-database

**Add Controller** → Add MVC Controller with Views , using EF

"CustomerController"

build → enter into url/customer → you'll see the list of customers!

* Razor takes advantage of HTML5 Date Picker
  + each browser treats this differently, some you are not able to type in date

*- better to make your own for this reason*

Lazy loading → when its gathering more than we need to use this

**EF handles data → Razor presenting the data**

In customer model,

→ we want to **add attribute [Display(name = "First Name")]**

(will add the space when razor displays it)

DOn't have to use EF

* make classes for models
* Add a controller with the code to use it

→ but with EF, we only have to worry about objects

**CustomerController**

public async Task<IActionResult>Details(int? id)

return View(away\_context.Customers.**ToListAsync();**

→ **asyncrance access** - allows us to do other request while the server is working on something else

* we won't be using this though so you can take it out

public async Task<IActionResult> Index()

return View("Index",\_Context.Customers.ToList()))

**In Index Views** → @ (RAZOR) ← code follows

Razor guesses a lot, try to minimize the guessing

@model → IEnumerable<MikesCoreDemo.Models.Customer>

→ **@model is an instruction** to compiler (NOT that code follows)

**Model → object that was passed in**

Model[0].Firstname → brings back firstname of first customer

=> anonymous function

@Html.DisplayFor(**modelItem** → Dummy placeholder - use any word

* same w/ **model** → lambda variable - use any word

**→ all 4 models have different meanings**

**→ Capital Model has the data**

**@HTML.DisplayFor** is a helper (optional)

**Running the program and how we got there**

**(I missed a lot of the steps in this analysis)**

in StartUp, default route is set → which takes us home -- but we entered /customer

Index view → layout page contains bulk of this html

Content of view is Index and middle part

F12 for developer tools, explorer and hover over table to see that it's a table

Customer display name

< a asp-action="Edit" → anchor tag that is not HTML, but RAZOR -- In browser will be <a href

→ you could just use html instead & only use razor when you need to inject something @item

Browser doesn't care about RAZOR so hackers won't know you're using it

Razor is only useful when bringing back html

→ Looking back at Edit() in Customer Controller to see that it passes an ID

→ Change the error message to something more friendly

Look more into Razor on pg 142

<label **asp-fo**r=:Firstname" ← telling razor to go look at how to display this

<input **asp-fo**r="FirstName" class="formControl"/>

→ razor also gave us **span** for error statements

→ The wizard **did not include paging**, so make sure to add pages

or you can delete this template and write your own code

Dear @Model.Firstname this is your page.<br />

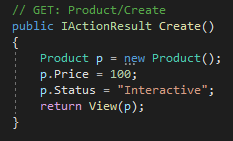
Your last name is @Model.Lastname

*When deleting records→ add them to an oldCustomerDatabase*

**Add ProductController**

→ we have create(, and a create without a parameter

Without param returns nothing. We need a blank form to add a new object

→ we add this

**http Methods**

GET → Sql Select

POST → Sql Insert

PUT/ PATCH/ MERGE → Sql Update

DELETE

*→ how data is passed and intended reason for existing*

GET

http://www.toys.com

<http://www.toys.com/product/2>

[http://www.toys.com/product/2?Name=boat&qty=12&[price=123](http://www.toys.com/product/2?Name=boat&qty=12&%5Bprice=123) ← WRONG

so we use POST instead → {[2?Name=boat&qty=12&[price=123](http://www.toys.com/product/2?Name=boat&qty=12&%5Bprice=123)} ← this is encrypted

[HttpPost] attribute ← can only get there with a get

Models - Products & Vendors

maintain product / vendor

Controllers product / vendor

TheStore

TheStoreDb