In partnership with: Microsoft Reactor

.NET Conf 2022

Student Zone

Build an API for Water consumption tracking

Chris Noring @chris_noring Senior Advocate at Microsoft

Story so far

- We’ve IoT devices setup collecting data in CSV files
- Next steps,
  - Transform CSV to Database tables
  - Create an API serving data from our database
  - Deploy API to Azure

CSV to database

Time	WindowDuration	Weight	AvgAccX	AvgAccY	AvgAccZ	RangeAccX	RangeAccY
10/14/2022 2:47:36 AM	00:00:03.1220000	-0.20000000298023224	-0.0441176477162277	0	1	0.10000000149011612	0
10/14/2022 2:48:06 AM	00:00:06.6790000	-0.7999999970197678	-0.008333333043588532	-0.0736111189756128	0.9833333326710595	0.9000000357627869	1.0000000298023224
10/14/2022 2:48:30 AM	00:00:05.0860000	-0.6000000238418579	-0.0036363634196194734	-0.040000[illegible]0086697665	0.9909090898253701	0.5000000149011612	0.4000000134110451
10/14/2022 2:48:52 AM	00:00:07.8860000	0.6000000238418579	0.05294117725947324	-0.11882[illegible]2864041049	0.90117647227119	1.699999988079071	1.4000000357627869
10/14/2022 2:49:17 AM	00:00:11.6280000	-0.20000001788139343	0.029600001037120818	-0.10080[illegible]00025033951	0.9343999986648559	1.100000023841858	1.600000023841858
10/14/2022 2:50:20 AM	00:00:04.4280000	-0.699999988079071	-0.0437500006519258	0	1	0.10000000149011612	0
10/14/2022 2:50:34 AM	00:00:03.9090000	-0.5	-0.07380952508676619	-0.0142[illegible]5714498588018	1	0.4000000134110451	0.4000000059604645
10/14/2022 2:51:05 AM	00:00:11.1590000	147.70000612735748	-0.08083333453784386	0	1.0008333335320154	0.10000000149011612	0
10/14/2022 2:51:56 AM	00:00:10.5980000	77.39999389648438	-0.09736842250353411	0	1	0.10000000149011612	0
10/14/2022 2:52:21 AM	00:00:07.9040000	70.79998779296875	-0.09529411906705183	0	1	0.10000000149011612	0
10/14/2022 2:52:44 AM	00:00:07.7500000	-68.5	0.17710843448897443	-0.09[illegible]397590618176632	0.9036144578313253	1.0000000149011612	0.30000001192092896
10/14/2022 2:53:04 AM	00:00:08.0970000	-59.80000305175781	0.1666666665810278	-0.0[illegible]52873564006268296	0.9137931024206096	1.199999988079071	0.3000000044703483[illegible]
10/14/2022 2:53:24 AM	00:00:05.7580000	0.6999969482421875	-0.01774193574824641	-0.[illegible]1612903261857648	0.9951612872462119	0.4000000059604645	0.5000000149011612
10/14/2022 2:53:54 AM	00:00:05.9560000	-32	0.23749999841675162	0.[illegible]6875000160653144	0.8703125030733645	1.0999999791383743	0.4000000134110451
10/14/2022 2:54:15 AM	00:00:05.2770000	-40.900001525878906	0.28596491092129755	-[illegible]0.0280701758568747	0.8508771940281517	1.2000000029802322	0.3000000044703483[illegible]
10/14/2022 2:54:28 AM	00:00:06.7690000	-98.6000030040741	0.40273972539460823	-[illegible]0.023287671579890055	0.6520547976028429	1.4000000357627869	0.3000000044703483[illegible]

IoT readings

WaterEntry	
Id	Primary key
DateTime	When measurement was made
Consumption	Human readable value

Database table modelling

Your API, – what it should contain

Routes

Data source

Authentication

Logging

Our plan
- Use minimal API for our API

Use	Define	Integrate
Use Entity Framework, an ORM to manage our database (Sqlite)	Define database as code first (tables and relations defined in code)	Integrate Entity Framework with Minimal API

Scaffold API

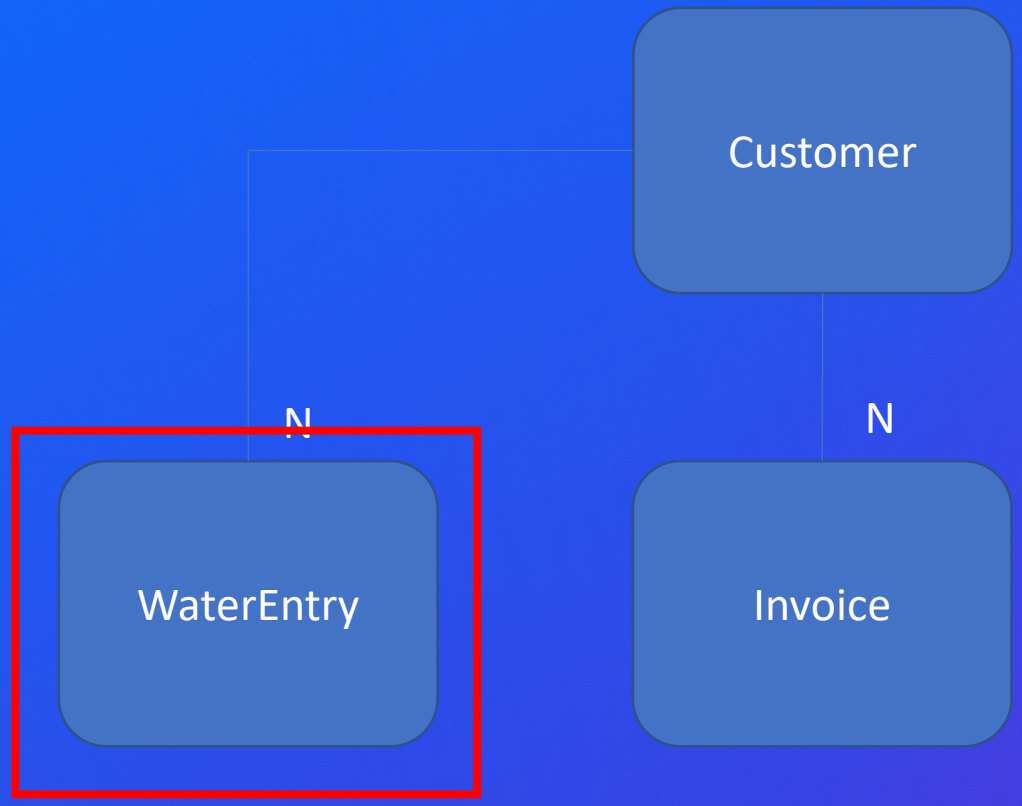
```
dotnet new web -o Api -f net6.0
```

```
var builder =
WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```

Entity Framework

- ORM, object relational mapper, handles comms with your Database
  - CRUD, **C**reate, **R**ead, **U**pdate and **D**elete
  - Seed, initial data
  - Migrations, changes to our database

Define data model



Create entities

```
class WaterEntry
{
  public int Id { get; set; }
  public int Consumption { get; set; }
  public DateTime DateTime { get; set; }
};
```

The database file

```
class WaterConsumptionDB : DbContext
{
  public WaterConsumptionDB(DbContextOptions options) :
base(options) { }

  public DbSet<WaterEntry> WaterEntries { get; set; } = null!;
}
```

Configure API to use EF
– install tools and dependencies

```
dotnet add package
Microsoft.EntityFrameworkCore.Sqlite --
-version 6.0
```

Add support for Sqlite with EF

```
dotnet tool install --global dotnet-ef
```

Tools for EF Core perform design-time development tasks. For example, they create migrations, apply migrations, and generate code for a model based on an existing database.

```
dotnet add package
Microsoft.EntityFrameworkCore.Design --version 6.0
```

Contains all the design-time logic for EF Core to create your database.

Configure in code

```
var connectionString =
builder.Configuration.GetConnectionString(“WaterConsumption") ??
"Data Source=WaterConsumption.db";
```

Creates a connection string that says how to connect, in this case also name of DB

```
builder.Services.AddSqlite<WaterConsumptionDB>(connectionString);
```

Adds a Sqlite provider – i.e Sqlite as a capability

Generate DB structure instruction – with a migration

dotnet ef migrations add InitialCreate

A migration can be applied “Up”
A migration can also be removed “Down”

A migration can:
Create/Remove/Alter table
Adding data

Up – applying changes

Down – rolling back changes

```
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace api.Migrations
{
    public partial class InitialCreate : Migration
    {
        protected override void Up(MigrationBuilder
migrationBuilder)
        {
        }

        protected override void Down(MigrationBuilder
migrationBuilder)
        {
        }
    }
}
```

Apply migration

```
dotnet ef database update
```

```
protected override void Up(MigrationBuilder migrationBuilder)
{
  migrationBuilder.CreateTable(
   name: "WaterEntry",
   columns: table => new
   {
     Id = table.Column<int>(type: "INTEGER", nullable: false)
    .Annotation("Sqlite:Autoincrement", true),
    Consumption = table.Column<int>(type: "INTEGER", nullable:
false),
    DateTime = table.Column<DateTime>(type: "DATETIME", nullable:
false)

   },
   constraints: table =>
   {
     table.PrimaryKey("PK_Resumes", x => x.Id);
   });

  migrationBuilder.InsertData(
    table: "WaterEntry",
    columns: new[] { "Id", "DateTime", "Consumption" },
    values: new object[] { 1, "2020-01-01T11:06", 100 }
  );
  // the rest is omitted

}
```

Add seed data – initial data for the DB

Create Initializer class that “seeds”/adds data to DB

Ensure this data exists in database

```
public class DbInitializer
{
  private readonly ModelBuilder modelBuilder;

  public DbInitializer(ModelBuilder modelBuilder)
  {
    this.modelBuilder = modelBuilder;
  }

  public void Seed()
  {
    modelBuilder.Entity<Resume>().HasData(
new Resume() { Id = 1, DateTime =
DateTime.Parse("2020-01-01T11:06") }
    );
 …
);
);
}
```

Run seed() at DB creation

```
class WaterConsumptionDb : DbContext
{
  public WaterConsumptionDb(DbContextOptions options) : base(options)

  protected override void OnModelCreating(ModelBuilder modelBuilder)
  {
    base.OnModelCreating(modelBuilder);
    new DbInitializer(modelBuilder).Seed();
  }

  // entities definition below…
}
```

OnModelCreating, run at creation

Add route

- Define route /Consumption
- Inject WaterConsumptionDB
- full JSON response

```
app.MapGet("/Consumption", (WaterConsumptionDb db) => {
return Results.Ok(db.WaterEntry.ToList());
});
```

What about deployment?

- App Service for deployment
- Sqlite -> Azure SQL Server
- Security? We will use API key

Prepare code for deployment

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

```
var connectionString =
builder.Configuration.GetConnectionString("DB");
builder.Services.AddSqlServer<WaterConsumptionDb>(connect
ionString);
```

Program.cs

```
"ConnectionStrings": {
  "DB" : "<replace with SQL server connection string>"
},
```

appsettings.json

Remove initializer code from DbContext class

Security, we want to rely on an API key

```
app.MapGet("/Consumption", ([FromHeader(Name = "dotnetconfstudentzone")] string ?
key, WaterConsumptionDb db) => {

  string ? secret = Environment.GetEnvironmentVariable("secret");
  if (key == secret) {
    return Results.Ok(db.WaterEntry.ToList());
  } else {
    return Results.StatusCode(401);
  }
});
```

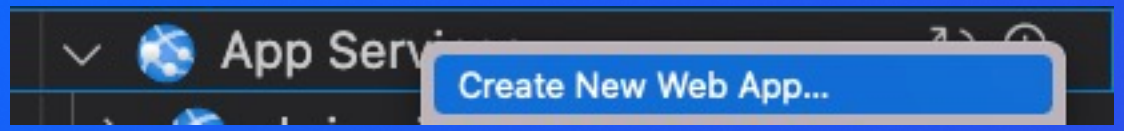
In appsettings.json, add key value for dotnetconfstudentzone

Provision cloud resources

- Portal.azure.com
- Provision SQL Server
- Provision SQL Database
- Create tables and insert data, query interface (you can ask EF for it)
- Copy connection string, assign value to DB key in appsettings.json
- Enable Azure resources to access DB (many different options here, you should add dedicated users belonging to different roles, reader, writer admin = least privilege)

Deploy

- More than one way to do it
- We will use Visual Studio Code



DEMO

Student Resources
http://aka.ms/learnstudent

Cloud Skills Challenge
https://aka.ms/dotnetstudentcsc

GitHub Repo
https://github.com/microsoft/dotnetconf-studentzone