Blazor & IoT

Källa: "Hans efter fyra" = FÖRELÄSNING - CLOUD PLATFORMS & INTERNET OF THINGS (NETFUND21-1)-20220328_160008-Meeting

I. Internet of things - Vad ska vi göra

Nuts & bolts

I teorin

Vi ska skapa en webbapplikation som redovisar värden från (IoT)enheter ute på nätet.
Vi hämtar värdena från ett API som placerats i Molnet. Molnet är i detta fall Azure
API'et är kopplat till en (IoT)Hub som också finns i Azure.
Hub'en tar mot data från enheterna och lagrar dem i en databas. I detta fall en Cosmos NoSqlDb i Azure.

I praktiken

Vi har inget av allt detta, så vi ska skapa alla delarna, inklusive sensorerna.

Skiss

FUNKTION:	Klient(er)	API	DB	HUB	Enhet(er)
SKAPAS SOM:	WebApp	WebFunction	Cosmos	IoTHub	ConsoleApp
PLATFORM:	Azure->Browser	Azure	Azure	Azure	Azure,PC, mobil m.fl.

Stegen

Grundpaketet

Single application - i två steg

Vi gör det steg för steg, först med "dummydata" direkt i klientappens kod, sedan som ett objekt ur en klass som definierar vår struktur. Då är den provkörd och klar, för nu.

Client - Server - Lokalt, men ändå http GET

Då skapar vi APIet och ändrar klientappen så den hämtar data från från denna. Ännu är allt lokalt på egna PC'n
Men nu har vi ändå kommunikation och hämtar data. Klienten är nu fullfjädrad, medans API'et serv'ar dummydata.

API'et flyttar till Azure

Med lite förberedelser är det bara att trycka på publish och vips! så har vi en molnapplikation att hämta från.

Klientkoden flyttar till Azure

På samma sätt som ovan. Nu behöver man bara en webbläsare lokalt.

Utökningen - IoT, Hub & DB

Iot-Hub

Sensorerna (Devices) kommunicerar med en Hub, som vårt API sedan hämtar data från.
(Mer riktigt är kanske att APIet går till Databasen som Hub'en skriver data till)

II. Skapa resurser i Azuremolnet

Källa: FÖRELÄSNING - CLOUD PLATFORMS & INTERNET OF THINGS (NETFUND21-1)-20220401_130122-Meeting

Förutsätter att du har ett konto och loggat in

WebApp

- Create a resource
- WebApp
- Namnge resursgrupp (gem. för din 'solution'?)
- Döp WebbAppen
- West Europe
- Serviceplan Gratis
- Ingen Deploy
- Ingen Monitorering
- Kontrollera och Deploya
- Vänta till den fått upp någonting på details - gå inte ifrån för snabbt

Hans rekommenderar f.ö. att lägga upp dina favoriter (stjärna) och sortera i sidomenyn.

FunctionApp

- Samma som ovan
- Plan: Consumption
- Kontrollera och Deploya

Cosmos DB account

- Samma som ovan
- Capacity mode: Provisioned (With free tier: Free) . ~~Serverless~~
- Kontrollera och Deploya

IoT Hub

- Samma som ovan
- Kontrollera och Deploya

Hans kommenterar: Inget av ovanstående kostar någonting.

III. Kodning i Visual Studio

Källa: FÖRELÄSNING - CLOUD PLATFORMS & INTERNET OF THINGS (NETFUND21-1)-20220328_160008-Meeting

WebApp - Blazor WebAssembly front-end (Statisk sida)

Skapar en frontend med dummydata som fungerar som 'scaffold' i nästa avsnitt

[index.html]

Startsidan för HTML-koden med <head>sektionen, där min lägger till länkar.

Fontawesome & Bootstrap

- Lägg till:

```
<script src="https://kit.fontawesome.com/41c8b474e2.js" crossorigin="anonymous"></script>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
<link href="css/app.css" rel="stylesheet" />

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>
    <script src="_framework/blazor.webassembly.js"></script>
</body>
```

Loading-spinner

Då det i realiteten tar tid att ladda sidan, är det juste att markera detta för användaren:

```
<body>
    <div id="app">
        <div class="d-flex justify-content-center align-items-center vh-100">
            <div class="spinner-border" role="status">
                <span class="visually-hidden">Loading...</span>
            </div>
        </div>
    </div>
```

[index.razor]

Huvudfönstrets kod

Genom uppdelning av koden i underavdelningar kommer denna bli mer överskådlig.
Här skapas heading och med Bootstrap skapas 12 kolumner, radvis orientering samt en gutter.
I detta fall vill jag ha tre 'cards' vars kod länkas in med taggar.

```
@page "/"

<PageTitle>My IoT page </PageTitle>

<h1>Embedded Control</h1>

<div class="row g-3">
    <div class="col-12">
        <Devices />
    </div>
    <div class="col-7">
        <Warnings />
    </div>
    <div class="col-5">
        <Activities />
    </div>
</div>
```

Moduler med dummies/placeholders för data (för provkörning)

[Devices.razor]

Skapa en tom razor-page i Pages-mappen och ersätt med något i denna stil:

```
<section class="card shadow">
    <article class="card-body">
        <h5 class="card-title">Devices</h5>
        <table class="table">
            <thead>
                <th scope="col">DeviceId</th>
                <th scope="col">Placement</th>
                <th scope="col">Status</th>
```

```
                <th scope="col">Sensor Type</th>
                <th scope="col">Last Updated</th>
                <th scope="col">Measurements</th>
                <th scope="col">Notifications</th>
                <th scope="col"></th>
            </thead>
            <tbody>
                <tr>
                    <td>@Guid.NewGuid().ToString()</td>
                    <td>connected</td>
                    <td>Conference Room</td>
                    <td>Temperature Sensor</td>
                    <td>@DateTime.Now</td>
                    <td>Temperature : 22'C, Humidity: 44 %</td>
                    <td></td>
                    <td>
                        <i class="fa-solid fa-play-pause"> </i><i class="fa-solid fa-pen-to-square"></i>
                    </td>
                </tr>
                <tr>
                    <td>36d5e179-cb7b-Udfb-a10c-25b1925a1404</td>
                    <td>connected</td>
                    <td>Fridge 1</td>
                    <td>Temperature Sensor</td>
                    <td>@DateTime.Now</td>
                    <td>Temperature : 10'C, Humidity: 21%</td>
                    <td><i class="fa-solid fa-triangle-exclamation"> Overtemp</i></td>
                    <td>
                        <i class="fa-solid fa-play-pause"> </i><i class="fa-solid fa-pen-to-square"></i>
                    </td>
                </tr>
            </tbody>
        </table>
    </article>
</section>
@code {

}
```

[Activities.razor]

```
   <section class="card shadow">
       <article class="card-body">
            <h5 class="card-title">Activities</h5>
            <table class="table">
                <thead>
                    <th scope="col">Activity Time</th>
                    <th scope="col">Message</th>
                </thead>
                <tbody>
                    <tr>
                        <td>2022-03-29 19:46:25</td>
                        <td>New IOT device was added</td>
                    </tr>
                </tbody>
            </table>
        </article>
    </section>

@code {

}
```

[Warnings.razor]

```
<div class="card shadow">
    <article class="card-body">
        <h5 class="card-title">Warnings</h5>
        <table class="table">
            <thead>
                <th scope="col">Incident Time</th>
                <th scope="col">DeviceId</th>
                <th scope="col">Message</th>
            </thead>
            <tbody>
                <tr>
```

```
                    <td>@DateTime.Now</td>
                    <td>36d5e179-cb7b-Udfb-a10c-25b1925a1404 </td>
                    <td>Check temp!</td>
                </tr>
            </tbody>
        </table>
    </article>
</div>

@code {

}
```

Blazor WebAssembly för API (Dynamisk hämtning)

15 min in i föreläsningen

> Gör om front-end-koden från sektion I till att kommunicera med ett API för datainhämtning och skapa API-koden med dummydata.

[Devices.razor]

Nu ska alla dummies bort och bara tomma <td> finnas. Därtill behövs lite kod

```
<section class="card shadow">
    <article class="card-body">
        <h5 class="card-title">Devices</h5>
        <table class="table">
            <thead>
                <th scope="col">DeviceId</th>
                <th scope="col">Placement</th>
                <th scope="col">Status</th>
                <th scope="col">Sensor Type</th>
                <th scope="col">Last Updated</th>
                <th scope="col">Measurements</th>
                <th scope="col">Notifications</th>
                <th scope="col"></th>
            </thead>
            <tbody>
                <tr>
                    <td</td>
                    <td</td>
                    <td</td>
                    <td</td>
                    <td</td>
                    <td</td>
                    <td></td>
                </tr>
            </tbody>
        </table>
    </article>
</section>
@code {

}
```

[DeviceList.razor]

Ändra koden så att den hämtar data (kod kopieras från WeatherForecast i FetchData.razor) enlig modellen i klassen DeviceItem.cs (som skapas snart…)

```
@using BlazorApp1.Models
@inject HttpClient Http

<section class="card shadow">
```

```
    <article class="card-body">
                .
                .
              <th scope="col">Notifications</th>
              <th scope="col"></th>
          </thead>
          <tbody>
               @foreach(var device in devices)
               {
                  <DeviceListItem Device="@device"/>
               }
          </tbody>
      </table>
    </article>
</section>
@code {
    private DeviceItem[] devices;
    protected override async Task OnInitializedAsync()
    {
        devices = await Http.GetFromJsonAsync<DeviceItem[]>("");
    }
}
```

Tillägg för att fånga upp om vi inte har några data ännu:

```
             @if(devices==null)
              {
                   <tr>
                      <td colspan="8">Loading ...</td>
                   </tr>
              }
              else
              {
                  @foreach(var device in devices)
                   {
                      <DeviceListItem Device="@device"/>
                   }
              }
```

[DeviceItem.cs]

Med utgångspunkt från (DeviceList.razor) skapar man filen i mappen Models

```
namespace BlazorApp1.Models
{
    public class DeviceItem
    {
        public string DeviceId { get; set; }
        public string Status { get; set; }
        public string Placement { get; set; }
        public string SensorType { get; set; }
        public string LastUpdated { get; set; }
        public string Temperature { get; set; }
        public string Humidity { get; set; }
        public string Alert { get; set; }

    }
}
```

[DeviceListItem.razor]

Ändra koden så att den använder fälten i DeviceItem.cs

```
@using BlazorApp1.Models
<tr>
    <td>@device.DeviceId</td>
    <td>@device.Status</td>
    <td>@device.Placement</td>
    <td>@device.SensorType</td>
    <td>@device.LastUpdated</td>
```

```
    <td>Temperature: @device.Temperature C, Humidity: @device.Humidity%</td>
    <td><i class="fa-solid fa-triangle-exclamation"></i></td>
    <td>
        <i class="fa-solid fa-play-pause"> </i>
        <i class="fa-solid fa-pen-to-square"></i>
    </td>
</tr>

@code {
    [Parameter]
    public DeviceItem item { get; set; }
}
```

FunctionApp - API - Azure function

24 min in i föreläsningen

Här skapar vi ett API som vi förser med Dummydata. Det körs lokalt på vår egen maskin så vi kan testköra med vår webbapp över nätverket (även om det iofs bara är inom PC'n).

[GetDevices.cs]

Skapa projekt under Solution

- new Azure function - döp till GetDevices
- Http-Trigger
- Auth anonymous
- Ta bort raderna 20-31

Resultat:

```
    public static class GetDevices
    {
        [FunctionName("GetDevices")]
        public static async Task<IActionResult> Run(
            [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)] HttpRequest req,
            ILogger log)
        {
            return new OkObjectResult(responseMessage);
        }
    }
```

[DeviceItems.cs]

För att kunna testköra utan enheter eller databas kopierar vi klassen DeviceItems från Webbappen

- Skapa mappen Models
- Skapa ny klass DeviceItems.cs
- Kopiera klassen från WebAppprojektet och klistra över den tomma klassen i filen
- Spara

Tillbaka till [GetDevices.cs]

- Skapa instansen **list** av klassen **DeviceItem**
- Skapa en record
- Lägg till Dummy-kod
- Ta bort "post" på rad 19
- Byt ut "responseMessage" mot "list" på sista raden
- Resulterande kod:

```
    public static class GetDevices
    {
        [FunctionName("GetDevices")]
        public static async Task<IActionResult> Run(
            [HttpTrigger(AuthorizationLevel.Anonymous, "get", Route = null)] HttpRequest req,
```

```
            ILogger log)
        {
            var list = new List<DeviceItem>()
            {
                new DeviceItem()
                {
                    DeviceId = "36d5e179-cb7b-Udfb-a10c-25b1925a1404",
                    Status = "connected",
                    Placement="Fridge 1",
                    SensorType="Temperature Sensor",
                    LastUpdated=DateTime.Now.ToString(),
                    Temperature="10'C",
                    Humidity="21%",
                    Alert="Overtemp!"
                }
            };
            return new OkObjectResult(list);
        }
    }
```

Provkörning och hämtning av lokal URL

- Välj Azurefunction (uppe under menyraden)
- Kör
- Kopiera URL'en (Den kommer att visas när den kompilerat , i command-fönstret)
- Kör den i webläsaren och ovanstående JSONdata kommer att visas!
- Klistra nu in den även i WebApp'en:

[GetDevices.razor]

sista raderna:

```
@code {
    private DeviceItem[] devices;
    protected override async Task OnInitializedAsync()
    {
        devices = await Http.GetFromJsonAsync<DeviceItem[]>("http://localhost:xxxx/api/GetDevices");
```

CORS-problem

Cross Origin blah-blah är ett skydd vi måste ta bort, det görs lokalt

[localSettings.json]

lägg till en rad:

```
{
    "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;AccountName=embedcontrolapistorage;  . . . . ",
    "FUNCTIONS_WORKER_RUNTIME": "dotnet"
  }
  "Host": { "CORS": "*" }
}
```

Kompilering och körning av båda projekten samtidigt

t.b.d

Publicering i Azure

Börja med AzureFunction

- Högerklicka
- Välj Publish

Connectionstringen till ditt API

- Gå till dashboarden i Azure
- Välj din API-app
- I vänstermenyn, välj Funktioner
- Välj din funktion (GetDevices)
- Välj, Hämta funktions-URL
- Kopiera den

[GetDevices.razor]

- Klistra in/ersätt connection-stringen.
- spara