CHALMERS

Webbaserad datavisualisering av producerad energi från solceller

Examensarbete inom Data- och Informationsteknik

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2020

EXAMENSARBETE

Webbaserad datavisualisering av producerad energi från solceller

Dara Khadjehnouri
Philip Hellberg

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg 2020

Webbaserad datavisualisering av solceller

Dara Khadjehnouri
Philip Hellberg

Examinator: Jonas Duregård
Handledare: Sakib Sistek

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Webbaserad datavisualisering av producerad energi från solceller

Institutionen för Data- och Informationsteknik
Göteborg 2020

Sammanfattning

Projektet har genomförts på ett svenskt mjukvaruföretag som arbetar med IT-lösningar och molntjänster för företags varuflöden inom logistik, industri och e-handel. Företaget har solpaneler på taket.

Syftet med detta examensarbete är att presentera befintlig data kring solcellerna på ett sätt så att de anställda och besökare på företaget kan ta del av informationen via en lokal hemsida.Tillsammans med företaget togs en idéskiss fram. Vi ska ta reda på hur man kan använda Angular, .NET Core och Highcharts för att visualisera datan.

Solpaneler genererar ström genom att fotoner strålar på två halvledare. Eftersom dessa har olika laddning finns en potential mellan dessa och en elektron kan sättas i rörelse och på så vis generera ström.

För att kunna skapa hemsidan som visar solenergidata behövdes en del tekniska verktyg. Man kan dela in det i två huvudsakliga delar: frontend och backend. I detta projekt används primärt Angular och Highcharts för frontend. JavaScript/TypeScript är programmeringsspråken..NET Core hanterar, med hjälp av *dapper* och *mediatR,* backend med C# som programmeringsspråk och Microsoft SQL Server Management Studio som databashanterare. Med hjälp av dessa bibliotek och ramverk skapas en lokal hemsida.

Produkten blev en liten och modulär lokal hemsida där grafer ritas upp med den data som databasen tillhandahåller och design enligt idéskiss.

Vi hade många idéer om vad man hade kunnat göra i projektet. Hade man fortsatt jobba med projektet så skulle man till exempel kunna koppla på ett API från en väderstation som visar hur mycket sol som förväntas komma. Vi skulle då kunna visa en graf som visar förväntad energiproduktion. Man skulle även kunna visualisera exempelvis hur många hushåll den producerade energin kan försörja.

Nyckelord: JavaScript, TypeScript, Angular, .NET Core, frontend, backend, ramverk, C#(c-sharp), API, dapper, mediatR, Highcharts.

Abstract

The project has been made through a swedish software developing company who works with IT and cloud-based services for companies' commodity flows within logistics, industry and e-commerce. The company has solar panels on the roof.

The purpose with this bachelor thesis is to present existing data about the solar panels in a way the employees and company visitors can take part of the information through a local webpage. Together with the company an idea sketch was made. We will work out how to use Angular, .NET Core and Highcharts to visualize the data.

Solar panels generate current through photons radiating on two semiconductors. Since these semiconductors have different electrical charges, there is a potential between them. This way an electron can be put into motion and hence generate current.

To create the webpage that shows the solar energy data a couple of technical tools were needed which can be divided into two main categories: frontend and backend. In this project Angular and Highcharts was primarily used for the frontend, where JavaScript and TypeScript are the programming languages. Dapper, .NET Core manage the backend with C# as a programming language and Microsoft SQL Server as the database management studio. Through these libraries and frameworks a local webpage is created.

The product resulted in a small and modular local webpage where graphs are drawn with data provided by the database and is designed according to the idea sketch.

We had a lot of ideas about what could be done with the project. If you were to continue working on the project you could for example connect an API for a weather station that shows how much sun is expected. You could then show a graph with expected energy production. You could also visualize how many households the produced energy would support.

Keywords: JavaScript, TypeScript, Angular, .NET Core, frontend, backend, ramverk, C#(c-sharp), API, dapper, mediatR, Highcharts.

Förord

Det här examensarbetet genomfördes av Dara Khadjehnouri och Philip Hellberg som studerar högskoleingenjörsprogrammet i datateknik. Arbetet motsvarade 15 högskolepoäng och gjordes på helfart, alltså ca åtta veckor.

Bland många vill vi i synnerhet tacka teamet på företaget som hjälpte oss med idéer och visioner. Tack Albin, Tobias, Fredrik och Andrea. Vi vill också tacka Sakib, vår handledare på Chalmers, som stöttat oss och spridit motivation till oss under hela projektet.

Begrepp/Förkortningar

HTML	**H**yper**T**ext **M**arkup **L**anguage
CSS	**C**ascading **S**tyle **S**heets
SASS	**S**yntactically **A**wesome **S**tyle **S**heets
SPA	**S**ingle-**P**age **A**pplication. En applikation som renderar om sidan istället för att länka till helt ny sida.
Cross-platform	En mjukvara som kan köras på flera olika operativsystem.
Frontend	Det en användare ser på en webbsida är ett gränssnitt skapat av frontend
Backend	Det som gömmer sig bakom det visuella. Oftast dataextrahering genom kontakt med databas via någon form av server.
ORM	**O**bject **R**elational **M**apping
API	**A**pplication **P**rogramming **I**nterface. Ett gränssnitt mellan applikation och, oftast, databas.
Injiceras	Dependency Injection. En teknik som används när man vill åstadkomma ett beroende sinsemellan objekt.
Provider	Någonting som tillhandahåller något annat.
Service	Ett objekt som har som huvuduppgift att dela data, modeller och funktioner med andra komponenter i ett Angular-projekt.
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
KPI	**K**ey **P**erformance **I**ndicator. Nyckeltal på svenska.

Innehållsförteckning

1. Inledning

I detta kapitel beskrivs bakgrunden till arbetet samt företaget där arbetet genomförts. Företaget kommer i denna rapport vara anonymt och benämnas endast F. Rapportens syfte och tidsram presenteras och vilka avgränsningar som gjorts beskrivs.

1.1 Bakgrund

F är ett svenskt mjukvaruföretag som arbetar med IT-lösningar och molntjänster för företags varuflöden inom logistik, industri och e-handel.

F:s kontor är byggt för att vara klimatsmart och miljövänligt. Flera sensorer som är utplacerade i byggnaden samlar in data som styr byggnadens temperatur, ventilation etc. Som en del av detta finns det solceller installerade på husets tak. Solcellerna producerar el som används för att tillgodose en del av husets energibehov.

Datan om hur mycket solcellerna producerar samlas in och lagras i en databas, tillsammans med övrig data från sensorerna i byggnaden. Det är dock få på företaget som har insikt i denna data och om hur solcellerna presterar och det finns i dagsläget inget sätt för de anställda eller andra besökare att ta del av datan.

F saknar alltså en webbsida eller ett program för att visualisera datan. Framför allt är F intresserade av att visualisera information och prestanda för solcellerna. Detta projekt kommer fokusera på att visualisera information och prestanda för solcellerna på byggnadens tak.

Datan önskar F att presentera på flera skärmar runt om i byggnaden och vill därför att datan ska vara informativ och enkelt utformad, så att anställda och besökare enkelt ska se hur mycket energi solcellerna producerar.

F gav oss en idéskiss att utgå från (*se Figur 5.1*).

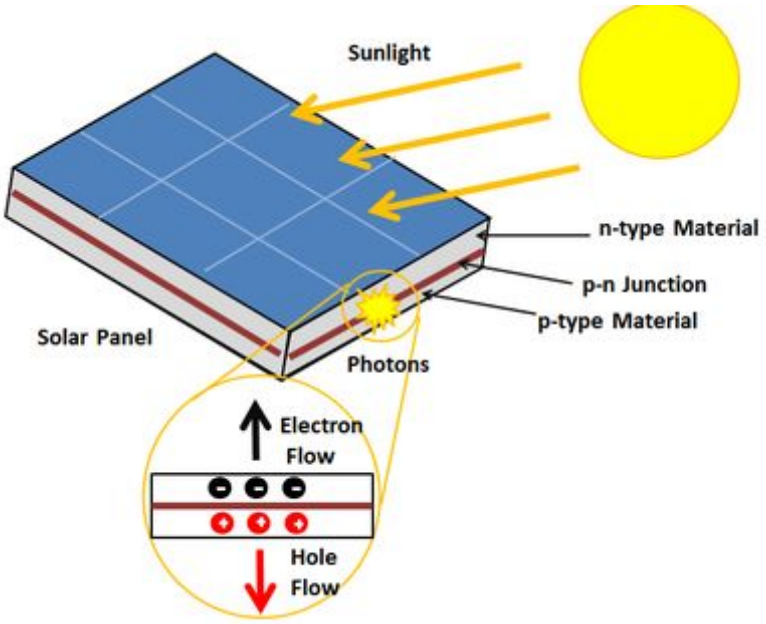
1.1.1 Solceller - Fotovoltaisk effekt och process

Fotovoltaik är teknik som utnyttjar den fotovoltaiska effekten för direkt omvandling av ljusenergi från solen till elektrisk energi genom så kallade solceller, eller fotovoltaiska celler.

Fotovoltaisk effekt upptäcktes först av fransmannen Edmond Becquerel då han i sin fars laboratorium lyckades bevisa att vissa material kunde utvinna solens energi och omvandla den till elektrisk energi [1]. Fenomenet inträffar när en fotovoltaisk cell blir utsatt för fotoner (ljus) och genererar en spänning eller elektrisk ström [2].

Vidare är dessa fotovoltaiska celler uppbyggda av två sorters silikonhalvledare: typ-p samt typ-n. Skillnaden mellan dessa är att silikonet med typ-p har blandats med grundämnet bor och silikonet med typ-n med grundämnet fosfor. Detta medför att typ-p har en elektron mindre och är positivt laddad, och typ-n har en elektron mer och är negativt laddad [3]. Efter sammansättning av de olika silikonen bildas ett elektriskt fält mellan dem där elektronerna rör sig mot det positivt laddade typ-p-silikonet och elektronhålet[1] rör sig mot det negativt laddade typ-n-silikonet. När en foton med rättmätig våglängd träffar en cell så överförs energin från fotonen till elektronerna i det elektriska fältet mellan silikonen. Detta orsakar ett s.k tillståndshopp hos elektronerna, även kallat excitering, där de, pga energitillskottet, har blivit labila och nu tenderar att attraheras mot typ-n-silikonet istället. På så vis skapas en elektrisk ström som man kan extrahera (se *Figur 1.1.1)*.

[1] Motsatsen till en elektron. Samma magnitud med motsatt polaritet [4]



Figur 1.1.1: En solcells funktionalitet [2]

1.2 Syfte

Syftet med examensarbetet är att kunna presentera och visualisera data av solcellernas energiproduktion till företagets anställda och besökare.

1.3 Mål

Målet med projektet är att skapa en webbsida som visualiserar datan som solcellerna producerar.

1.4 Precisering av frågeställningen

Hur kan vi med hjälp av Angular, .NET Core och Highcharts visualisera F:s data från deras solceller?

1.5 Avgränsningar

Avgränsningarna som görs i projektet är följande:

- Enbart beröra de tekniska detaljer om solceller som är väsentliga för rapporten
- Enbart beröra de övergripande basfunktionaliteterna i hårdvaran som är väsentliga för rapporten
- Enbart beröra den övergripande implementeringen för webbsidan
- Enbart konstruera en webbsida som passar F, inte något generiskt

1.6 Tidsplan

Tidsplanen för projektet finns bifogat, se Gantt-schema i Appendix 1. Fram till och med läsvecka 5 ska vi lära oss de ramverk och diverse tekniska bitar som behövs för att utveckla produkten samtidigt som vi faktiskt utvecklar den. Innan läsvecka 5 ligger fokus alltså på att utveckla produkten. Efter läsvecka 5 ändras fokus till rapportskrivandet.

2. Teknisk bakgrund

För att kunna skapa webbsidan som visar solenergidata behövdes tekniska verktyg. Man kan dela in dem i två huvudsakliga områden: frontend och backend.

2.1 Frontend

I kommande avsnitt kommer olika bibliotek, ramverk och verktyg som använts för frontend att beskrivas.

2.1.1 Angular

Angular är ett open-source ramverk skapat av Google och används för att skapa det visuella, i.e. användargränssnittet på en webbsida. Det är ett sätt att organisera kod och brukar vara effektivt för SPAs. I en SPA används länkar, om det finns några, till att rendera om sidan istället för att slussas vidare till en ny sida. Högst upp i kod-hierarkin finns enbart en HTML-sida som dynamiskt uppdateras med hjälp av en s.k. rot-komponent. Går man djupare ner i hierarkin kan man se flera olika komponenter som hjälper till att bygga upp denna rot-komponent [5].
Angular CLI (Command Line Interface) är ett tillskott i en senare version av Angular som möjliggör att skapa Angular-objekt via kommandotolken.

2.1.2 Highcharts

Highcharts är ett bibliotek för att visa upp grafer av olika slag och skrivs i JavaScript. Genom att importera stöd för Highcharts i sin Angular-kod och därefter skapa data för graferna kan man via Angular sedan rita ut dessa på sin webbsida [6].

2.1.3 TypeScript

JavaScript är ett skript- och programmeringsspråk som tillåter implementering av komplexa och kraftfulla webbsidor [7]. Det används ofta i samband med HTML samt CSS-filer för att göra webbsidan responsiv. Till exempel, när en användare klickar på en knapp på en webbsida så är det JavaScript-koden som talar om för webbsidan vad den ska göra vid knapptryckningen. TypeScript är en utvidgning av JavaScript och tillsätter dels möjligheten att ha typer/returtyper på variabler/funktioner men också användandet av moduler [8]. I ett typiskt Angular-projekt används TypeScript-filer istället för JavaScript-filer. TypeScript kompileras till Javascript vid kompilering.

2.1.4 HTML

HTML används för att strukturera och organisera en webbsida [9]. Det är inte ett programmeringsspråk i den innebörd att man inte kan lägga till dynamisk funktionalitet, utan det är ett formateringsverktyg. Det liknar mer ett textredigeringsprogram än ett programmeringsspråk. Som verktyg till formateringen finns element, eller taggar som det kallas. Taggarna ser ut så här: *<tagg></tagg>,* där *tagg* är ett element och oftast öppnas och avslutas på det viset. Olika typer av taggar kan ha olika typer av attribut och då kan det se ut så här: *<tagg attribut1="värde" attribut2="värde"> </tagg*>. De kan även ha ett värde mellan taggen, som på paragrafer *<p>Detta är en paragraf</p*>. HTML tillför inte något estetiskt tilltalande på en webbsida, utan man lägger bara till element man vill ha och sedan stajlas dessa element med hjälp av CSS.

2.1.5 CSS/SASS

CSS används för att göra HTML-element på en webbsida visuellt tilltalande. SASS är en utvidgning av CSS med tillgång till ytterligare funktionalitet och används oftast i industrisammanhang [10].

2.1.6 Bootstrap

Bootstrap är ett ramverk, eller open-source-verktygslåda, som tillhandahåller ett enklare sätt att bygga upp en sida på. Man kan använda sig av saker som responsiva rutnät, förbyggda komponenter, JavaScript-plugins och mycket mer [11].

2.1.7 Adobe Illustrator

Adobe Illustrator är ett verktyg som tillhandahåller möjligheten att skapa grafiska komponenter och är ett vanligt val när det kommer till vektorgrafik [12].

2.2 Backend

I detta avsnitt presenteras bibliotek, ramverk och verktyg som använts för backenden.

2.2.1 .NET, .NET Framework, .NET Core

.NET är ett stort ramverk med många verktyg och är till för att utveckla olika typer av applikationer [13]. Det finns två huvudsakliga implementationer av ramverket: .NET Framework och .NET Core. Den förstnämnda är den första utvidgningen av .NET och används till att köra webbsidor, lokala applikationer och mycket annat. .NET Core är open source och är i princip en nyare version med tillägget att det är cross-platform och kan köras på Linux, Windows samt macOS. Programmeringsspråken som används vid utveckling i .NET är C#, F# och Visual Basic. I detta projekt används C#.

2.2.2 Object Relational Mapping

ORM är en programmeringsteknik för att hantera konvertering av data mellan en databas och en applikation [14]. Det kan underlätta för utvecklare att använda en ORM under produktion om ett mer objektorienterat arbetsflöde föredras.

2.2.3 Dapper

Dapper är en mikro-ORM där man ställer native SQL-frågor direkt inbakade i sin C#-klass för att fråga efter data i en databas [15]. Biblioteket utvecklades av StackOverflow och passar bra för utvecklare som föredrar “rena” SQL-frågor.

2.2.3 Mediator-mönstret och MediatR-biblioteket

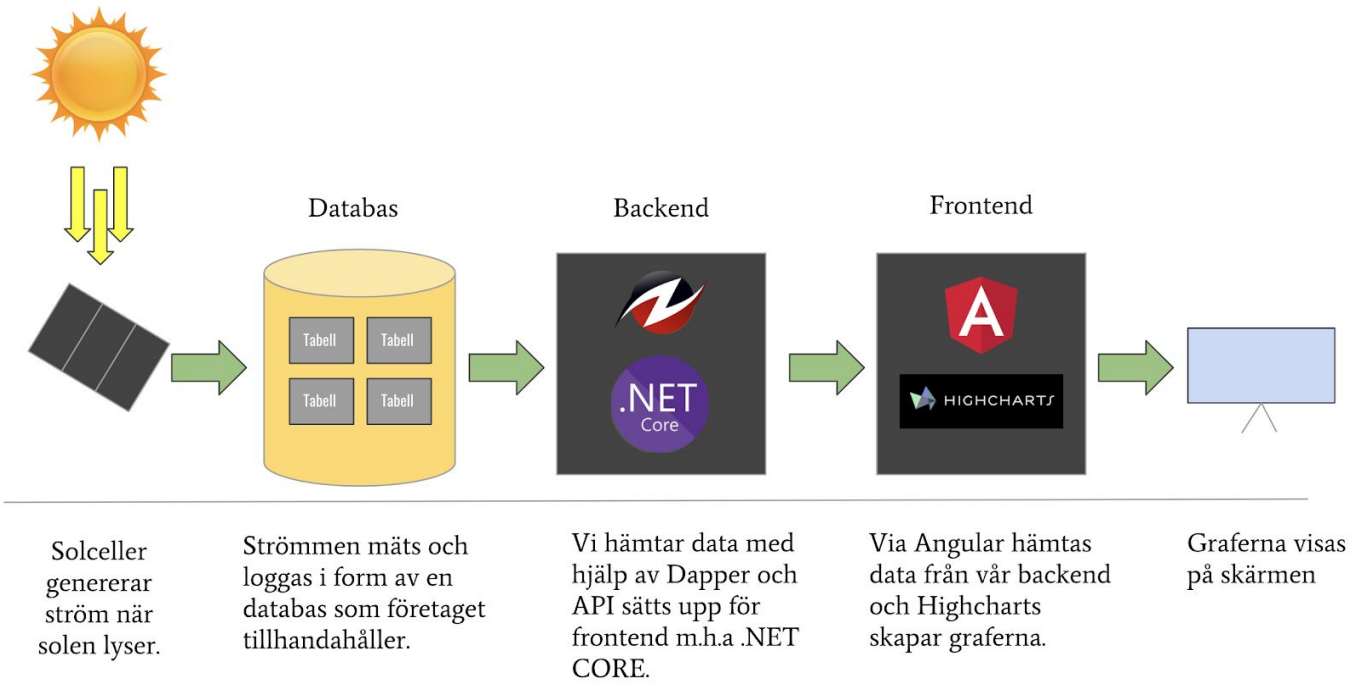
Ett sätt att designa sin produkt är genom att använda mediator-mönstret. Istället för att ha flera objekt som kommunicerar med varandra direkt och är beroende av varandras förändringar så kan man ha en sk. mediator som sköter kommunikationen mellan objekten/klasserna helt och hållet. Inget objekt känner till något annat vilket leder till mindre koppling mellan klasser i systemet [16]. MediatR är ett bibliotek i .NET, skapat av Jimmy Bogard, bestående av C#-klasser samt gränssnitt som gör mediator-mönstret möjligt [17].

2.2.4 Microsoft SQL Server Management Studio

Microsoft SQL Server Management Studio (SSMS) är en utvecklingsmiljö för att hantera SQL-infrastrukturer, i.e. databaser [18].

2.3 Översikt

I figur 2.3 ser man hur hela informationsflödet ser ut från att solen lyser till det att informationen visas på en skärm.



Figur 2.3 Flödesdiagram

3. Metod

För att kunna börja göra en webbsida som presenterar data på producerad solenergi behövdes först kunskap i vad målet med webbsidan var.

Under ett möte med VD och anställda på F, tidigt i projektet, växlades tankar där olika idéer bollades och mål sattes. Lösningar till olika svårigheter undersöktes, till exempel hur representationen av datan skulle se ut för att de som tittar på den skulle kunna ta åt sig informationen.

För att kunna presentera datan snyggt på en webbsida krävdes först kunskap i hur en webbsida byggs upp och hur den kommunicerar med en databas. Man kunde dela upp det i två huvudsakliga delar: backend och frontend. I backenden, byggs en webbserver upp som hämtar, tolkar och delar ut data från databasen till klienter som frågar efter den. I frontenden (klienten) skickas HTTP-förfrågningar om datan som sedan visas upp på ett snyggt sätt.

För backend används .NET Core, ett kraftfullt ramverk som kan användas till de flesta typer av applikationer. För frontend används Angular, ett ramverk utvecklat av Google som är bra för SPAs (single-page applications). Arbetsflödet går ut på att bygga komponenter som man fyller sidan med. Dessa komponenter kan vara ett artikelkort på en e-handelssida, t.ex. För att koppla ihop backenden med frontenden används så kallade tjänster där HTTP-förfrågningar görs till webbservern.

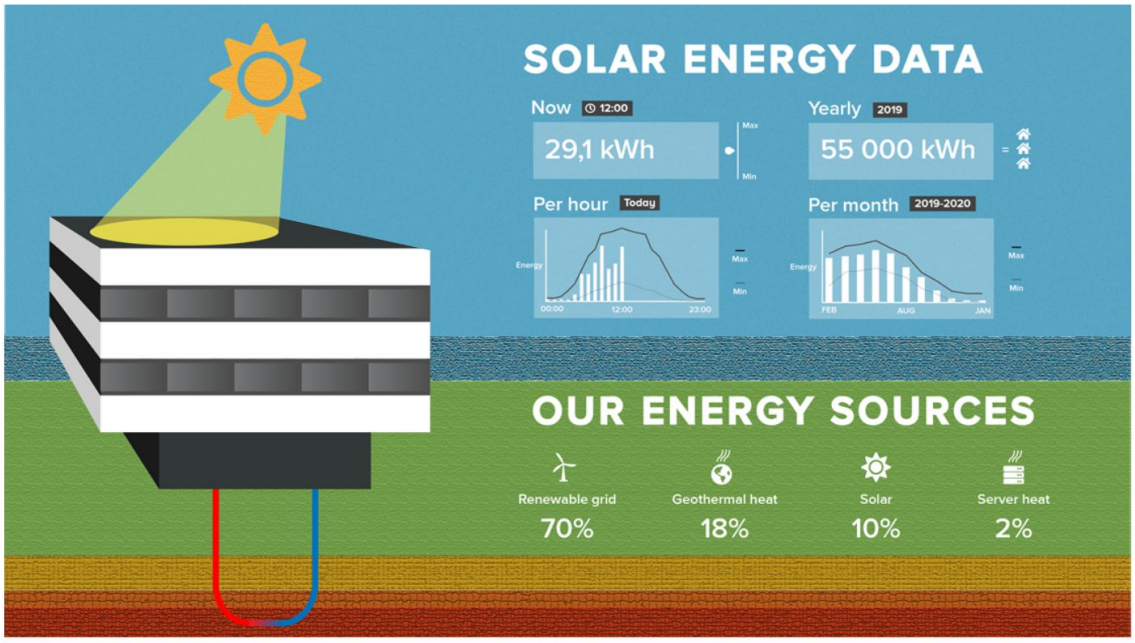
4. Etik och hållbarhet

Ur ett etiskt perspektiv ser vi inga problem med att genomföra detta arbete. Vi anser inte att vårt arbete påverkar någon person, företag eller annan part negativt ur etiska och samhälleliga aspekter.

Företagets val att installera solceller på taket är hållbart ur ett miljöperspektiv då dom kan utnyttja solens resurser. Eftersom vi i detta projekt kommer visualisera den utvunna energin av solcellerna hoppas vi detta kan inspirera andra företag och anställda att välja hållbara energikällor, såsom solenergi.
Vi kan därför inte heller se några negativa aspekter med att utföra arbetet ur ett miljöperspektiv.

5. Minimum Viable Product

Under projektets uppstart blev det snabbt tydligt vad för typ av produkt som skulle utvecklas och hur den skulle se ut. Dels tillhandahölls vi en kravspecifikation men också en skiss på hur produkten kunde komma att se ut. *Figur 5.1* illustrerar idéskissen.



Figur 5.1. Idéskiss på sidan

Inledningsvis var kraven att webbsidan:

- Skall visa ett antal KPI:er och hur de förändras över tid, i form av två grafer
- Skall uppdateras utan användarinput
- Skall vara en intern sida (SPA)
- Skall byggas i Angular
- Skall vara estetiskt tilltalande på stor skärm (ca 5m * 3m)

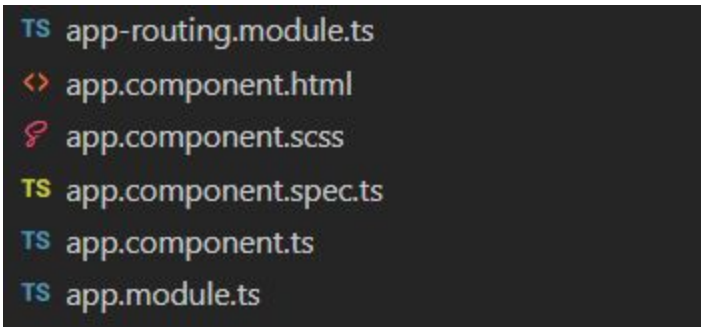
Efter dessa tillkom ytterligare önskemål, eller framtidsvisioner, som nämns under rubriken *Diskussion*.

6. Utförande av frontend

I det här avsnittet kommer implementationen av frontend att förklaras och demonstreras. Hur användandet av Angular, i kombination med Highcharts, och realiserade idéskissen (*se Figur 5.1*) till en lokal webbsida.

6.1 Angular

För att skapa ett nytt projekt använder man, via Angular CLI, kommandot: "*ng new myNewApp*" där *myNewApp* är önskat namn på projektet.
När detta gjorts skapas filerna enligt *Figur 6.1*.

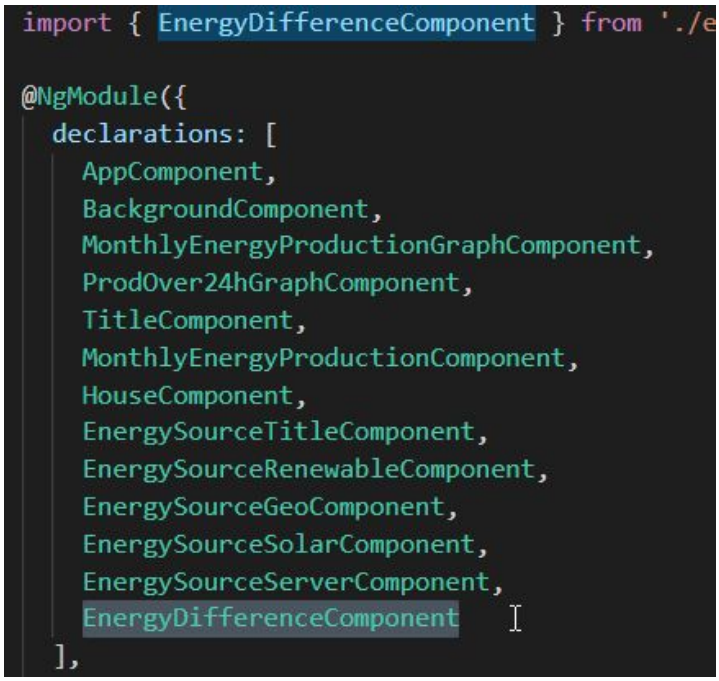


Figur 6.1. Nyskapade filer efter "ng new myNewApp"-kommandot körts.

Filen "*app.component.ts*" är s.k *förälder* till de olika *objekten* som används genom hela projektet. Medans "*app.module.ts*" är filen där Angular-objekt importeras och refereras.
Innan vi började jobba med Angular hade F satt upp ett repo som vi fick tillgång till, där Angularprojektet redan var skapat.

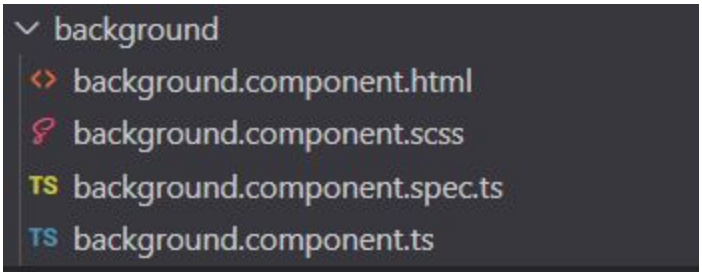
6.1.1 Angular Components

En typ av objekt som används i Angular är s.k *komponenter*.
Vi skapade nya komponenter genom att skriva "*ng generate component componentName*" där *componentName* är namn på önskad komponent.
När detta görs skapas det filer en simpel komponent behöver som dessutom automatiskt importeras och deklareras i app.module.ts (*Figur 6.2)*



Figur 6.2. Utklipp från app.component.ts där EnergyDifferenceComponent automatiskt importeras och deklareras.

Genom att köra kommandot skapas fyra filer som tillhör skapad komponent, alla samlade i en mapp.
Till en början skapade vi en bakgrundskomponent. Bakgrundskomponenten var tänkt att ligga till grund för allt som ska visas på bild och även där våra mindre komponenter laddas in (*Figur 6.3)*.



Figur 6.3. Bakgrunds-komponenten "background":s fyra autogenererade filer.

Filen*"background.component.spec.ts"* är en s.k testfil som vi inte utnyttjat i vårt projekt. De andra tre filerna är; en HTML-fil, en SCSS-fil och en TypeScript-fil.

Eftersom vi skulle göra en SPA insåg vi att det var enbart bakgrundsytan som skulle visas och modifieras.
Vi började därefter att dela upp ytan i olika storlekar, beroende hur mycket plats vi ville att komponenterna skulle få. Först skissat på ett papper, därefter delades dessa ytor upp i rader och kolumner som överfördes till HTML-kod (*Figur 6.4)*.

```
<div class="row titleRow">
  <div class="col-md-6"></div>
  <div class="col-md-6"><app-title>
</div>
<div class="row">
  <div class="col-md-6">
```

Figur 6.4. Utklipp från "background.component.html"

Till denna HTML-kod lade vi till CSS med hjälp av Bootstrap så att rader och kolumner anpassar innehållet till vilken skärm det visas upp på (*Figur 6.5)*.

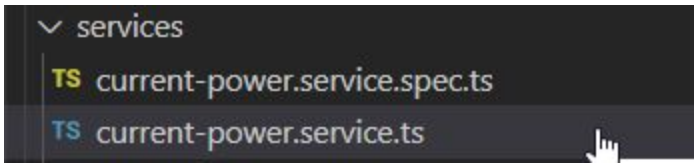
```
.titleRow {
    padding-top: 2em;
}
.sourceRow{
    padding-top: 1.5em;
}
.box {
    padding-left: 5.15em;
```

Figur 6.5. Utklipp från "background.component.scss"

6.1.2 Angular Services

Utöver att skapa komponenter så har Angular stöd för att hämta data. Ett sätt är att skapa s.k *services*.
Services använder sig av en Angular-klass vid namn *HTTPClient* som kan via en funktion att göra "*HTTP Get requests"*. Det är detta funktionsanrop som kommunicerar med vår backend och hämtar datan.

För att skapa en service använder man kommandot "*ng generate service serviceName*" där *serviceName* är önskat namn (*Figur 6.6)*.



Figur 6.6. Nyskapade filer efter kommandot körts.

En *service* är en tjänst som komponenter *konsumerar* genom att s.k *injicera* servicen. Genom att injicera komponenten med servicen ges komponenten tillgång till servicens funktionalitet.
När servicen är skapad behöver man ange servicen som en *provider* i app.modules.ts *(Figur 6.7)*.

```
imports: [
  BrowserModule,
  AppRoutingModule,
  HttpClientModule,
  HighchartsChartModule
],
providers: [TotalWattsPerMonthService, CurrentPowerService, Kwh24Service],
```

Figur 6.7. Utklipp från "app.modules.ts" där CurrentPowerService anges vara provider.

Injiceringen sker genom att man dels importerar sin service i den komponent man önskar ha funktionaliteten. Dessutom behöver man ange den importerade servicen till komponentens konstruktor som argument (*Figur 6.8*).

```
import { CurrentPowerService } from '../services/current-power.service';
import { CurrentPower } from '../interfaces/CurrentPower';

@Component({
        selector: 'app-monthly-energy-production',
        templateUrl: './monthly-energy-production.component.html',
        styleUrls: ['./monthly-energy-production.component.scss']
})
export class MonthlyEnergyProductionComponent implements OnInit {
        currentEnergy: any;
        currentMonth: any;
        currentYear: any;

        constructor(private fetch: CurrentPowerService) { }
```

Figur 6.8. Utklipp från "monthly-energy-production.component.ts" där en service injiceras.

Via servicen "*fetch"* av typen CurrentPowerService får komponenten "*MonthEnergyProductionComponent"* ett s.k *observable*-objekt som man sedan *prenumererar på*(*subscribe* på engelska). Prenumerationen medför att datan som HTTP get requestet tillhandahåller går att använda (*Figur 6.9)*.

```
ngOnInit() {
        //Init graph
        setTimeout(() => {
                this.fetch.getData().subscribe(data => {
                  this.updateCurrentEnergy(data);
                });
              }, (0));
```

Figur 6.9. Fortsättning ur "monthly-energy-production.component.ts" där observable-objektet prenumereras.

Omedelbart när datan tas emot använder man den till vad den behövs till. I vårt fall har vi brutit ut koden till en stödfunktion vid namn *updateCurrentEnergy* som uppdaterar komponenten med inkommande data.

6.1.3 Highcharts

Själva visualiseringen av datan sker med grafer via ett externt bibliotek kallat *Highcharts*. Efter att ha installerat Highcharts så importerar man in det i sitt projekt (*Figur 6.10)*.

```
import * as Highcharts from 'highcharts';
```

Figur 6.10. Utklipp från en grafkomponent.

Därefter bestämde man sig för vilken graf man ville ha och skapade därefter rätt inställningar för hur just den specifika grafen skulle se ut (*Figur 6.11)*.

```
public options: any = {
  yAxis: {
    gridLineWidth: "0",
    labels: {
      enabled: false
    },
    plotLines: [{
      label: {
        text: '9 kWh', // Intersection value.
        style: {
          color: 'grey',
          fontFamily: 'Helvetica',
          fontSize: '15px'
```

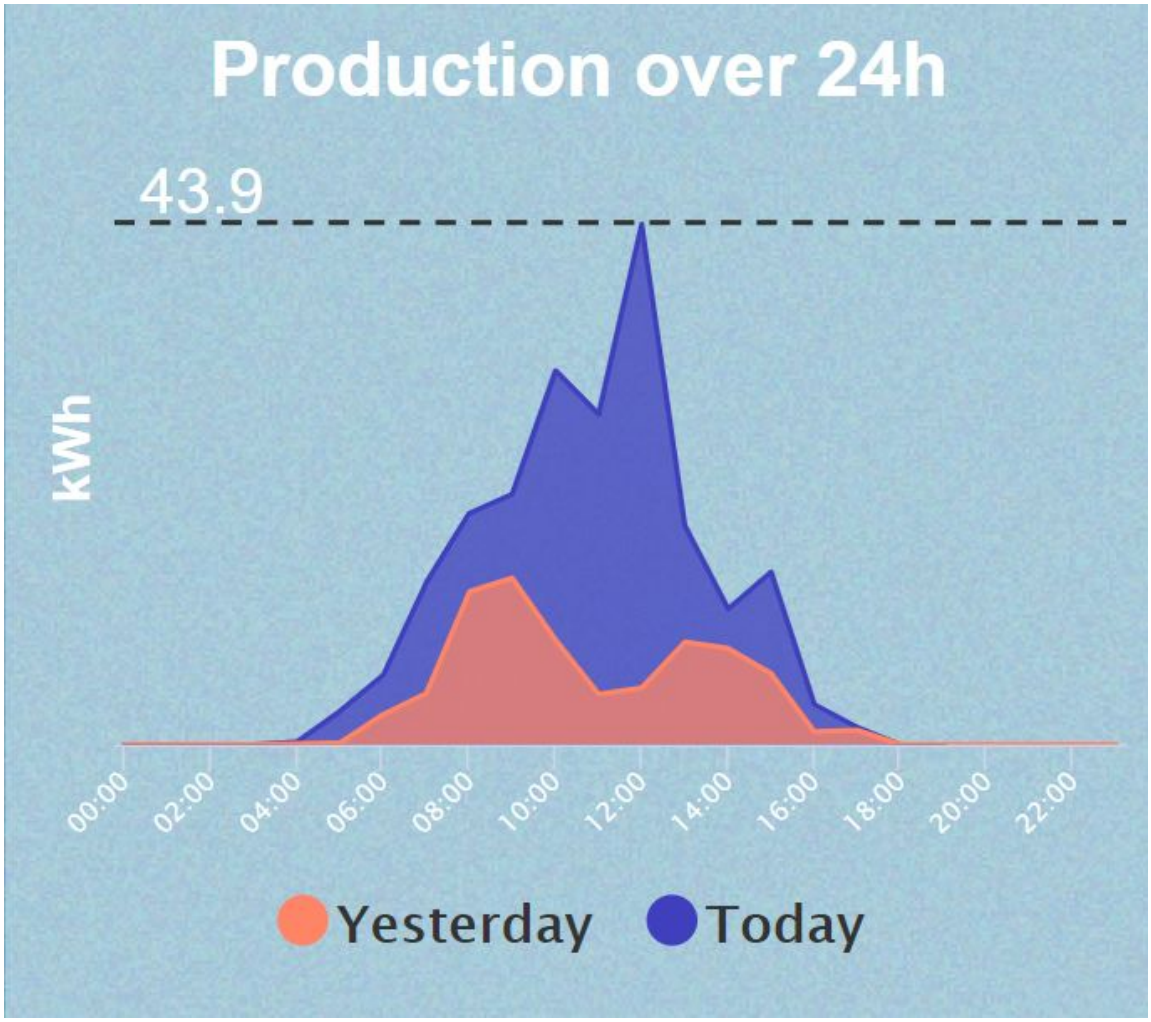
Figur 6.11. Exempel på hur en del av inställningarna för en graf kan se ut.

När inställningarna är skapade behöver man dessutom rita ut grafen med korrekt inställningar.
Detta görs genom att använda sin importerade Highchart och ange inställningarna som argument till specifik graf och HTML id-tagg, i detta fall “container-2” (*Figur 6.12)*.

```
this.chart = Highcharts.chart('container-2', this.options);
```

Figur 6.12. Koden ovan ritar ut grafen med inställningar från “options”-variabeln och skapar grafen nedan.

Figur 6.13 illustrerar resultatet av tidigare nämnda operationer.



Figur 6.13. Bilden visar grafen för “container-2”.

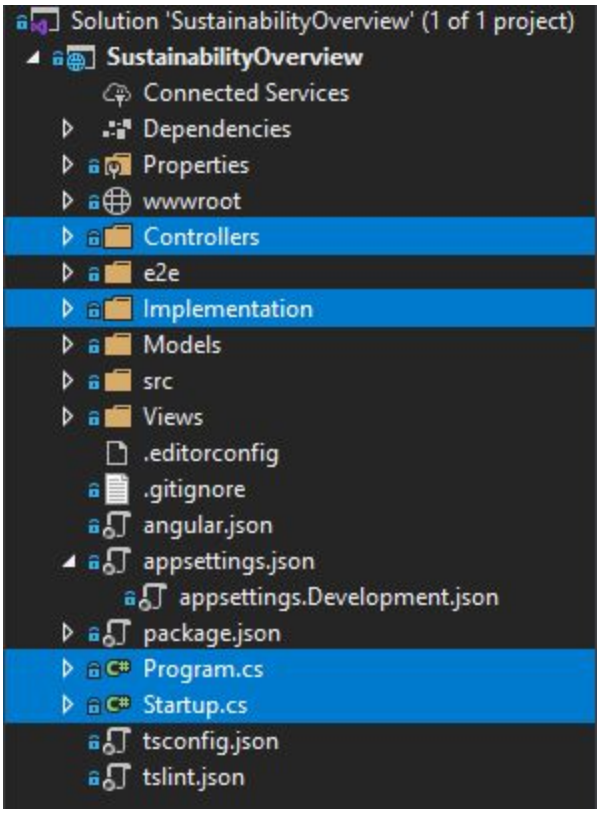
7. Utförande av backend

I det här avsnittet kommer implementationen av backend att förklaras och demonstreras. Hur användandet av databasen, .NET Core, Dapper och MediatR realiserade idéskissen (*se Figur 5.1*) till en lokal hemsida.

Värdena som ska visas i graferna är hämtade från en databas som lagrar olika typer av data gällande solcellerna. Det finns många tabeller som modellerar olika relationer men innehållet i tabellerna är inte helt kända, varken för projektdeltagarna eller anställda på företaget. Det finns dock kolumner där typen av data är trivial, såsom *id* och *tidsstämpel.* Men när det kommer till energi har det uppstått förvirringar. För att underlätta detta har två stycken vyer, eller *views*, tilldelats som man kan göra sina SQL-förfrågningar på. I den ena finns data om hur mycket energi som produceras per månad och i den andra hur mycket som produceras varje dag. Alltså har inga större åtgärder gjorts för att hantera databasen, annat än att tyda den och kommunicera med den genom .NET Core och dapper.

7.1 .NET Core

I projektmappen finns det två huvudsakliga mappar som sköter backenden: *Implementation* och *Controllers (Figur 7.1)*.



Figur 7.1. Illustration av filstruktur

Utöver dessa mappar finns det även två stycken standardfiler som körs när applikationen startar: *Startup.cs* och *Program.cs*. Med standardfiler menas filer som skapas när man första gången skapar projektet. *Program.cs* är en klass som innehåller en main-metod där det enda väsentliga som händer är att kalla på klassen *Startup*. I filen *Startup.cs* ligger konfigureringar för vad som ska hända vid uppstart av applikationen. Man lägger bland annat till tjänsterna (*service*) som används (*Figur 7.2)*.

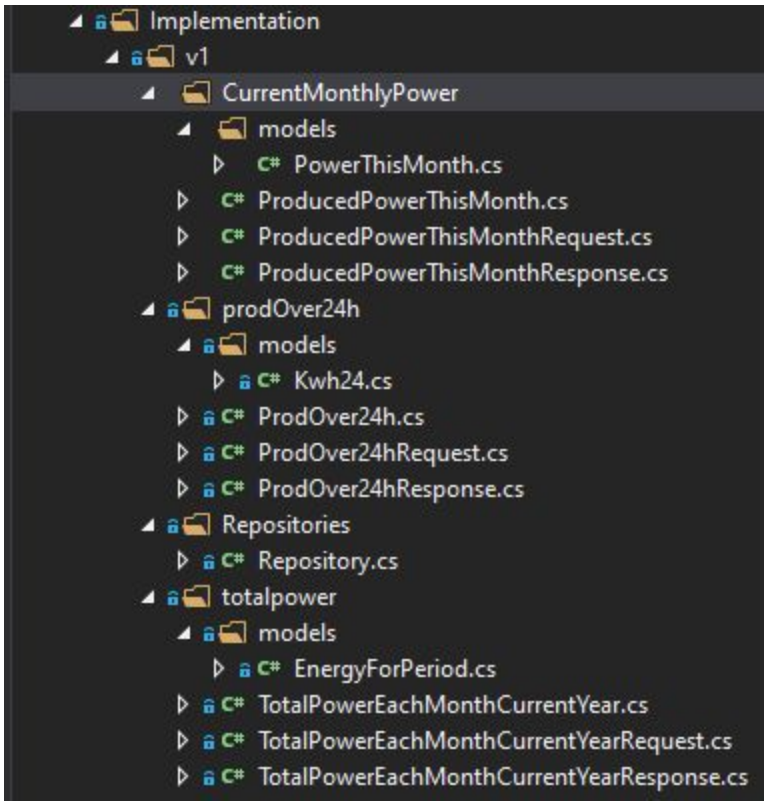
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMediatR(typeof(TotalPowerEachMonthCurrentYear));
    services.AddMediatR(typeof(PowerThisMonth));
    services.AddControllersWithViews();
    services.AddScoped(typeof(IRepository), typeof(Repository));
}
```

Figur 7.2. Konfigurering i Startup.cs där olika tjänster läggs till

I vårt fall utgör *MediatR* men också *Repository* en stor del i backenden. Hur dessa implementeras och vad de faktiskt är och gör kommer i senare avsnitt.

7.1.1 Implementation

I *Implementation* finns alla objekt som är tänkta att utgöra de olika API som interagerar och hämtar data från databasen (*Figur 7.3)*. För att hämta från och skriva till databasen ska en HTTP GET- respektive HTTP POST-operation utföras. Men eftersom datan som produceras av solcellerna redan finns lagrad i databasen så är den enda operationen som krävs HTTP GET.



Figur 7.3. Tydligare bild från filstrukturen i backenden

Till att börja med skapas *Repositories*. Det är i *Repository.cs* som anslutningen till databasen görs *(Figur 7.4)*.

```
private readonly IDbConnection _connection;
private readonly IConfiguration _configuration;


public Repository(IConfiguration configuration)
{
  _configuration = configuration;
  var connectionString = _configuration.GetValue<string>("ConnectionString");

  if (string.IsNullOrEmpty(connectionString))
  {
    throw new ArgumentNullException("Empty connection string");
  }

  _connection = new SqlConnection(connectionString);
}
```

Figur 7.4. Konstruktorn i klassen Repository som sköter databasanslutning

I konstruktorn skapas anslutningen. I variabeln *_configuration* lagras filen *appsettings.Development.json* där anslutningssträngen till databasen finns. Sedan hämtas värdet (anslutningssträngen) från nyckeln "*ConnectionString"* och lagras i variabeln *connectionString*. Slutligen skapas anslutningen med den lagrade strängen.

För att sedan göra förfrågningar till databasen via *Repository.cs* finns en instansmetod, *GetObjectsAsync*, som bl.a tar en SQL-sträng som inparameter och med hjälp av *dapper*-metoden *QueryAsync* exekverar förfrågningen och mappar resultatet asynkront.*(Figur 7.5)* .

```
public async Task<IEnumerable<T>> GetObjectsAsync<T>(string sql, object parameters)
{
  return await _connection.QueryAsync<T>(sql, parameters);
}
```

Figur 7.5. Instansmetoden i Repository som gör en SQL-förfrågning m.h.a. dapper

För att ta ett objekt som exempel går vi igenom *totalpower* för att visa hur processen för att skapa ett objekt, som sedan ska interagera med databasen, ser ut. Datan som hämtas presenterar hur mycket energi som producerats per månad, för aktuellt år.

Först skapas en modell, *EnergyForPeriod.cs*, för hur datan som hämtas från databasen skall representeras (*Figur 7.6)*.

```
public class EnergyForPeriod
{
  public List<dynamic> TotWatt { get; set; }

  public List<dynamic> Period { get; set; }
}
```

Figur 7.6. Modellen för hur datan som hämtas ska tolkas. Typen “dynamic” är en dynamisk typ som fastställs under exekvering.

Efter det skapas tre filer som kommer att sköta vilka förfrågningar som görs till databasen:

- *TotalPowerEachMonthCurrentYear.cs*,
- *TotalPowerEachMonthCurrentYearRequest.cs*
- *TotalPowerEachMonthCurrentYearResponse.cs*.

Den väsentliga filen är den förstnämnda. Klassen implementerar gränssnittet *IRequestHandler* som har en metod, *Handle*, som måste implementeras. I konstruktorn injiceras *Repository* för att, via den, kunna skicka förfrågningar till databasen.

I *Handle* initieras först variabler som skall användas i SQL-förfrågningarna följt av SQL-förfrågningarna själva (*Figur 7.7)*.

```
var todayDate = DateTime.Today.ToString("yyyy-MM-dd");
var currentYear = DateTime.Today.ToString("yyyy");
var totWatt = await GetObjectsAsync(@"SELECT EnergyForPeriod
                                      FROM [                    .[ESolarEnergyByMonth]
                                      WHERE PeriodDate >= '" + currentYear + "-01-01 00:00.000'" +
                                      " ORDER BY PeriodDate ASC");

var date = await GetObjectsAsync(@"SELECT PeriodDate
                                   FROM [                    .[ESolarEnergyByMonth]
                                   WHERE PeriodDate >= '" + currentYear + "-01-01 00:00.000'" +
                                   " ORDER BY PeriodDate ASC");
List<dynamic> energy = new List<dynamic>();
List<dynamic> period = new List<dynamic>();
```

Figur 7.7. Initiering av variabler som används i förfrågningarna

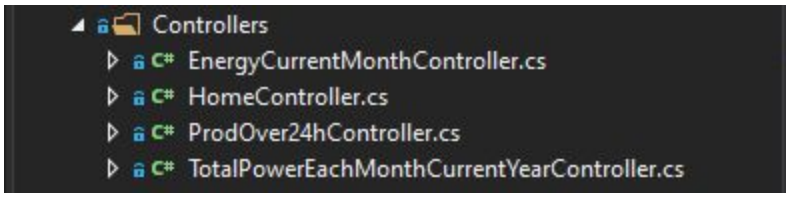
Därefter lagras de hämtade värdena i listvariablerna *energy* och *period* och returnerar dom som ett *EnergyForPeriod*-objekt (*Figur 7.8)*.

```
return new EnergyForPeriod()
{
  TotWatt = energy.ToList(),
  Period = period.ToList()
};
```

Figur 7.8. Returnering av objektet med data från databasen

7.1.2 Controllers

För att faktiskt skapa API:t och lägga till navigation till det så används *Controllers* - en för varje API (*Figur 7.9)*.



Figur 7.9. Controllers för varje objekt

I respektive controller skickar *mediatorn* ett request som hanteras av *handlern* för det objektet (se föregående avsnitt). Med *[Route(“watts”)]-* samt *[HttpGet]*-annoteringarna talar man om för controllern vad sökvägen till API:t är respektive att det är en *GET*-operation som skall utföras (*Figur 7.10)*.

```
[ApiController]
[Route("watts")]
public class TotalPowerEachMonthCurrentYearController : ControllerBase
{
    private readonly IMediator _mediator;

    public TotalPowerEachMonthCurrentYearController(IMediator mediatR)
    {
        _mediator = mediatR;
    }
    [HttpGet]
    public async Task<EnergyForPeriod> Index()
    {
        return await _mediator.Send(new TotalPowerEachMonthCurrentYearRequest());
    }
```

Figur 7.10. En controller och dess mediator som hanterar kommunikation med objekt

I *Startup.cs* lägger man sedan till alla controllers (*Figur 7.11).*

```
services.AddControllersWithViews();
```

Figur 7.11. Tillägg av alla använda controllers

Därefter kan man navigera till API:t (*Figur 7.12)*.

localhost:44354/watts


```
{"totWatt":
[286.47299999999814,753.984000000004,3827.0859999999957,7519.762000000002,8949.515,8441
.156999999992,7538,6636.109000000011],"period":["2020-01-01T00:00:00","2020-02-
01T00:00:00","2020-03-01T00:00:00","2020-04-01T00:00:00","2020-05-01T00:00:00","2020-
06-01T00:00:00","2020-07-01T00:00:00","2020-08-01T00:00:00"]}
```


Figur 7.12. Data från det nyligen skapade API:t

På så sätt skapas API:t som man i frontenden kan göra HTTP GET-operationer på (se *avsnitt 6.1.2*).

8. Azure och versionshantering

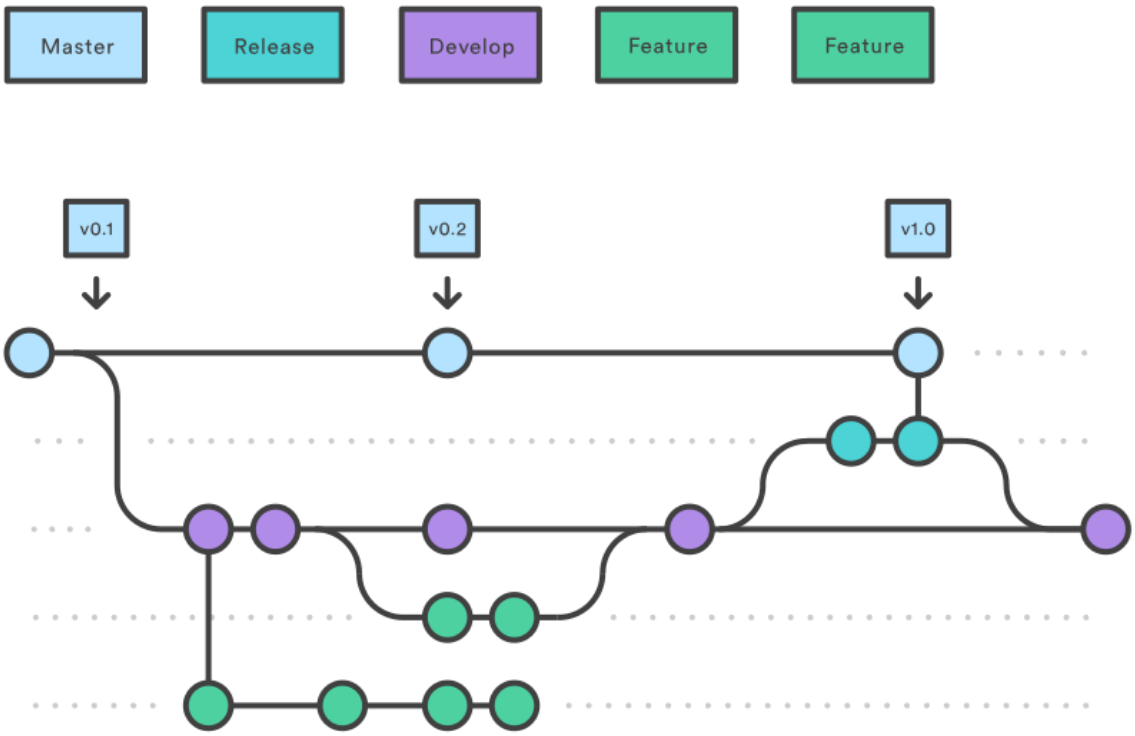
Microsoft Azure, eller Azure, är en molnplattform med lösningar såsom *Infrastructure as a Service (IaaS), Platform as a Service (PaaS)* och *Software as a Service (SaaS)* [19]. Den kan användas i analytiska ändamål, som lagring och mycket annat. Molntjänsten är pålitlig gällande säkerhetskopiering av data och kan användas till att vara värd för, utveckla, distribuera och underhålla en applikation. Med stöd för kontinuerlig integration och kontinuerlig distribuering (CI/CD) kan deltagare i ett projekt oftare leverera kod för att hålla varje förändring så liten som möjlig [20].

I detta projekt har Azure använts för versionshantering i huvudsak. Ifall en uppföljning eller distribuering av applikationen skulle vara av intresse så ligger projektet på molnet där andra kan fortsätta där vi slutade.

Det finns olika typer av principer och arbetsflöden när det kommer till versionshantering: *Centralized Workflow, Feature Branch Workflow, Gitflow Workflow* och *Forking Workflow*, för att nämna de mest populära [21]. I det här projektet har det lutat mot *Gitflow Workflow.*

8.1 Gitflow Workflow

I versionshanteringen brukar man dela upp utvecklingen av en applikation i olika grenar (branches). Man brukar främst prata om *Master-*, *Develop-* och *Feature*-branchen. På *Master*-branchen ska en färdig och stabil version av applikationen, den officiella utgåvan, till slut hamna i. I början av ett projekt brukar man “brancha ut” från *Master*-branchen och skapa en *Develop*-branch där kontinuerlig integration av funktionalitet, eller *features*, sker. När man har branchat ut från *Develop*-branchen och skapat en *feature*-branch och är klar med funktionaliteten för den så sammanfogar man (*merge:ar*) den branchen in i *Develop*-branchen så att den har den nya funktionaliteten. Man gör så många *feature*-branches som krävs för att applikationen ska uppfylla sina krav och när den gör det så klonar man *Develop*-branchen och skapar en *Release*-branch (även kallat för en fork-operation). I den branchen får inga nya features läggas till utan enbart bug-fixes och dokumentation. När applikationen är redo att distribueras merge:ar man *Release*-branchen in i *Master*-branchen med ett versionsnummer, och tar bort *Release*-branchen. Den här proceduren upprepas sedan för nyare versioner av applikationen [21]. Se *Figur 8.1.*



Figur 8.1: Illustration av hur arbetsflödet kan se ut i versionshanteringsverktyget Git [21]

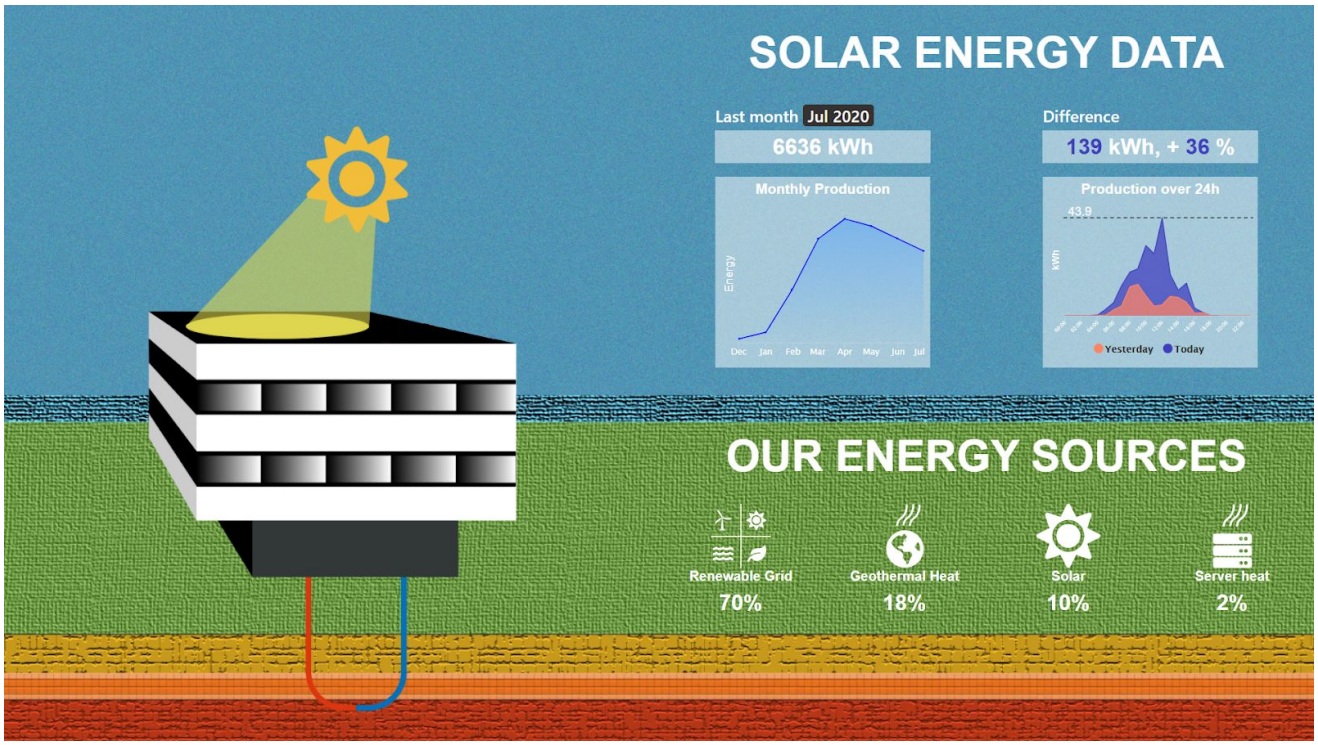
9. Resultat

Resultatet som medfördes av sammansättningen mellan backend och frontend var en sida som uppfyllde de ursprungliga kraven som vi kan se nedan.

- Skall visa ett antal KPI:er och hur de förändras över tid, i form av två grafer
- Skall uppdateras utan användarinput
- Skall vara en intern sida (SPA)
- Skall byggas i Angular
- Skall vara estetiskt tilltalande på stor skärm (ca 5m * 3m)

Samtliga krav ovan är uppfyllda. Vi har m.h.a Angular skapat en intern sida som visar två grafer som uppdateras utan användarinput. Denna sida kan visas och ser bra ut på en stor skärm, ungefär 5 m * 3 m. Grafen till vänster (se *Figur 9.1*) visar månatlig energiproduktion för aktuellt år och komponenten ovanför visar månadens värde. Grafen till höger visar två efterföljande dagars energiproduktion och komponenten ovanför visar den producerade energin för dagen som genererade mest energi, och skillnaden mellan dom i procent.

För att hålla backend och frontend någorlunda separerade från varandra sköttes utvecklingen av de i Visual Studio 2019 respektive Visual Studio Code. Det hela resulterade alltså i en sida där graferna uppdateras varje timme med ny data (om det finns någon sådan att hämta), där storleken är adaptiv (med vissa undantag) och möjlighet för enkla förändringar finns. Siffrorna under “Our energy sources” är och skall vara statiska siffror som man får ändra på manuellt. Bakgrunden och huset är gjorda i Adobe Illustrator.



Figur 9.1. Resulterande webbsida med live-grafer och live-etiketter

10. Diskussion

Syftet med examensarbetet var att kunna presentera och visualisera data av solcellernas energiproduktion till företagets anställda och besökare. I början av projektet lades det ner mycket tid till att planera och strukturera arbetet under projektets gång för att säkerställa att vi fokuserade på rätt saker.

För att nå vårt mål med att skapa en webbsida insåg vi tidigt att vi behövde separera dataextraheringen och datauppvisningen i en backend respektive frontend för att enkelt hålla isär kopplingen mellan dom. Med tanke på att det fanns en idéskiss att utgå ifrån kunde vi från början visualisera vad vårt arbete skulle resultera i. Vi ville ha två grafer som skulle visa upp någon form av data från solcellerna för att på ett enkelt sätt kunna visualisera den. Vad det var för data var upp till oss. Den ursprungliga idén var att i det ena diagrammet ha en graf som visade föregående års prestation, månatligt, och i samma diagram ha en graf som visar aktuellt års prestation. På så sätt kunde man jämföra produktionen per år. I den andra grafen ville vi ha produktion från två dagar tillbaka samt produktion från gårdagen och lägga dem som areor på varandra. Den andra grafen uppfyllde det ursprungliga kravet. Dessa två nyckeltal diskuterades gemensamt fram mellan projektdeltagarna och valdes för att det, rent intuitivt, kändes som en bra jämförelse: att ställa år mot år och dag mot dag för att se någon rimlig förändring. Det fanns även önskemål om att ha en live-graf som uppdateras var femte minut med hur mycket energi som producerats men den idén slopades ganska tidigt då vi insåg att det inte säger så mycket för gemene man. Det är normalt sett lättare att få en uppfattning om energiproduktion om man kan ställa ett resultat i relation mot ett annat.

Det fanns även planer på att ha med ikoner för hur mycket solenergin motsvarade i mer greppbara termer, som gemene man enklare hade kunnat ta till sig. Till exempel antalet hushåll som kunnat drivas med dagens producerade solenergi (*se figur 2 avsnitt 5*). Avgränsningen i skärmstorlek gjorde dock att det hade blivit för svårt rent visuellt att presentera.

Några andra idéer var att, genom användning av ett API från en väderstation, animera solen så att den lyste växlande starkt, ha med moln, soluppgång/solnedgång etc. beroende på väderprognoser samt att husets olika våningar skulle lysa grönt eller rött beroende på vilka som var mest energikrävande. Alla dessa idéer är noterade men tyvärr fanns inte tiden att implementera dessa.

Det var även planerat att skicka ut ett formulär till de anställda på företaget där de skulle få svara på några frågor om sidan. Frågor gällande hur relevant designen och informationen som visades var och eventuella förslag på vad som skulle visas istället. Vi skulle då kunna anpassa produkten beroende på vilken återkoppling vi fått. Tyvärr stötte vi på några tekniska problem med backend och frontend som tog längre tid än förväntat att lösa och det medförde en förskjuten tidsplan. Vi insåg att enkäten i sig skulle medföra arbete vi inte skulle hinna med, så vi valde därför att inte genomföra undersökningen.

Eftersom F inte har kunnat visualisera den redan framtagna datan från solcellerna för anställda och besökare, så fyller denna webbsida en funktion som de inte haft tidigare. Om F vill komplettera och skapa ytterligare komponenter så går det att koppla på dem på webbsidan i efterhand. Om önskemål finns kan man i framtiden implementera idéerna som nämndes ovan.

10.1 Kritisk diskussion

Hade vi haft mer kunskap och erfarenhet inom de olika ramverken hade mer tid kunnat läggas på andra delar av projektet. I början av projektet fick vi lägga mycket tid på att lära oss de olika ramverken. Ingen av oss har sedan tidigare arbetat med webbutveckling i denna form tidigare, utan bara skrapat på ytan av det mest fundamentala, såsom HTML, CSS och JavaScript. Hade denna djupare kunskap funnits från början hade mer fokus och tid kunnat läggas på att integrera mer data eller fler funktioner, som nämns ovan i diskussionen.

Eftersom vi inte kunde ramverken till en början visste var det också svårt med tidsplanering och avgöra hur lång tid vissa delar skulle ta. Exempelvis hade vi planerat för att genomföra en enkätundersökning, men eftersom vissa delar av utvecklingen tog längre tid än vi beräknat hamnade vi i tidsbrist och fick prioritera bort detta.

Med facit i hand hade vi valt att göra saker på olika sätt. Till exempel hade vi valt att göra bakgrundsbilden m.h.a. CSS och inte som en bild i Adobe Illustrator för att försäkra kvaliteten på bakgrunden, oavsett upplösning på skärm. Vi hade också gjort enhetstester för komponenterna, både för backend och frontend, för att säkerställa funktionalitet och minska risken för att applikationen kraschar. Slutligen, för att få en bättre struktur på backenden, hade vi skapat gränssnitt för respektive API-modell där man organiserar alla metoder som är tänkta att interagera med databasen, istället för att direkt i en *handler* göra förfrågningarna.

Referenser

[1] Energifakta.nu. *Solceller fungerar genom den fotovoltaiska effekten.* [online] Available at: https://energifakta.nu/solceller-genom-fotovoltaisk-effekt/ [Accessed 13 Oct. 2020]

[2] Hanania. J, Stenhouse. K, Donev. J. (2015). *Photovoltaic effect.* [online] Available at: https://energyeducation.ca/encyclopedia/Photovoltaic_effect [Accessed 13 Oct. 2020]

[3] Pickerel. K. (2018). *The difference between n-type and p-type solar cells.* [online] Available at: https://www.solarpowerworldonline.com/2018/07/the-difference-between-n-type-and-p-type-solar-cells/ [Accessed 13 Oct. 2020]

[4] Hanania. J, Stenhouse. K, Donev. J. (2018). *Electron hole.* [online] Available at: https://energyeducation.ca/encyclopedia/Electron_hole [Accessed 13 Oct. 2020]

[5] Bodrov-Krukowski, I. (2018). *Angular Introduction: What It Is, and Why You Should Use It.* [online] Available at: https://www.sitepoint.com/angular-introduction/ [Accessed 27 Sep. 2020]

[6] Highcharts.com. (2020). *Highcharts.* [online] Available at: https://www.highcharts.com/ [Accessed 27 Sep. 2020]

[7] MDN-contributors. (2020). *What is JavaScript?.* [online] Available at: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript [Accessed 27 Sep. 2020]

[8] TypeScriptlang.org. (2020). *What is TypeScript?.* [online] Available at: https://www.typescriptlang.org/ [Accessed 27 Sep. 2020]

[9] Domantas, G. (2019). *What is HTML? The Basics of Hypertext Markup Language Explained.* [online] Available at: https://www.hostinger.com/tutorials/what-is-html [Accessed 27 Sep. 2020]

[10] Sass-lang.com. (2020). *Sass Basics.* [online] Available at: https://sass-lang.com/guide [Accessed 30 Sep. 2020]

[11] Getbootstrap.com. (2020). *Build fast, responsive sites with Bootstrap.* [online] Available at: https://getbootstrap.com/docs/4.5/getting-started/introduction/ [Accessed 30 Sep. 2020]

[12] Wood, B. (2016). *See what you can create with Illustrator.* [Video file] Available at: https://helpx.adobe.com/illustrator/how-to/what-is-illustrator.html [Accessed 24 Oct. 2020]

[13] Dotnet.microsoft.com. (2020). *.NET, .NET Framework and .NET Core.* [online] Available at: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework [Accessed 5 Oct. 2020]

[14] ORM (INTE KLAR). (2020). *.NET, .NET Framework and .NET Core.* [online] Available at: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework [Accessed 5 Oct. 2020]

[15] Codedigest.com. (2017). *What is Dapper? How to User Dapper in Asp.NET MVC.* [online] Available at: http://www.codedigest.com/quick-start/17/what-is-dapper-how-to-use-dapper-in-aspnet-mvc [Accessed 9 Oct. 2020]

[16] W3sdesign.com. (2018). *Mediator Design Pattern.* [online] Available at: http://w3sdesign.com/?gr=b05&ugr=proble#gf [Accessed 9 Oct. 2020]

[17] Bogard, J. (2019). *Sharing Context in MediatR Pipelines.* [online] Available at: https://jimmybogard.com/sharing-context-in-mediatr-pipelines/ [Accessed 10 Oct. 2020]

[18] Docs.microsoft.com. (2019). *What is SQL Server Management Studio (SSMS)?.* [online] Available at: https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15 [Accessed 15 Oct. 2020]

[19] McCoy. L. *Microsoft Azure Explained: What It Is and Why It Matters.* [online] Available at: https://ccbtechnology.com/what-microsoft-azure-is-and-why-it-matters/ [Accessed 16 Oct. 2020]

[20] Sacolick. I. *What is CI/CD? Continous integration and continous delivery explained.* [online] Available at: https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html [Accessed 16 Oct. 2020]

[21] Atlassian.com. *Gitflow Workflow.* [online] Available at: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow [Accessed 20 Oct. 2020]

Appendix 1 - Gantt-schema tidsplan