Aarhus Universitet

Village Smart Grid

DAB HandIn 4

Tobias Kjær Henriksen	au479319
Jacob L. F. Kurtzhals	au537301
Mikkel Overgaard	au549290

Indhold

Introduktion

I denne opgave skal der udvikles et system til at kunne håndtere et Smart Grid. Et Smart Grid kan anvendes i mindre byer eller mellem lande til at kunne handle strøm med hinanden. Idéen bag en hel by som anvender Smart Grid er at alle indbyggere, virksomheder samt landbrug producer strøm og de forbruger selvfølgelig også strøm, det betyder at alle i byen er Prosumers. En prosumer kan så enten bruge for meget strøm, så bliver prosumeren nød til at købe strøm fra prosumers, hvis prosumeren derimod generere for meget strøm, kan prosumeren sælge dette til andre prosumers. Det som denne opgave fokuserer på er at lave det system, som de forskellige prosumers kan købe og sælge strøm over. System vil indeholde tre databaser, to SQL og en NoSQL database. Hvor diverse info omkring, prosumers, Smart Griddet og priser vil bliver persisteret. Der lægges et REST Api nedover så data kan tilgås ved brug af http kald, og der oprettes et konsol program til at prosumers kan "logge" på og bytte strøm. Hvis der ikke er nok strøm at købe for griddet og man ikke selv producere nok er det muligt at købe det der mangler fra et National Power Grid. Det samme gælder hvis der er for meget strøm i overskud vil National Power Grid også have mulighed for at købe det ekstra strøm

Design overvejelser

I dette afsnit vil der blive gennemgået hvilke design overvejelse vi har haft under vejs. Vi vil komme ind på hvorfor vi har valgt det antal databaser, som vi har valgt. Beskrive hvilke typer af databaser og hvorfor vi har overvejet disse til de forskellige funktioner. Hvor vi er kommet op med designet vedrørende klasser, beskrivelse af funktionalitet programmet skal have og om overvejelse om det skulle sættes op som en GUI eller blot en UI.

Antal databaser

Først havde vi overvejet om det var muligt at lave dette system med blot to databaser, da vi bare skulle have en måde at persisterer prosumers og transactions på. Disse to tog dog ikke højde for at prisen på strøm kan ændre sig, for eksempel ved udbud og efterspørgsel. Derfor var det nødvendigt at have en tredje database til at kunne holde styr på priserne og hvilke ombytnings værdier der skal være fra strøm til tokens. Men da vi kun har fået to forskellige databaser givet af universitet vil SmartGridInfo med priser i blot være en tabel i vores SQL database.

Type af databaser

Der skulle bruges to typer af databaser en SQL og en NoSQL, vi har fået givet to af universitetet en SQL database og en Azure Document Database. I dette afsnit skrives om hvilke overvejelser vi har haft vedrørende hvilket ansvar de forskellige databaser skal have og hvorfor at vælge den ene frem for den anden.

SQL

Det giver mest mening at persistere vores prosumer data, adresse data og info vedrørende priser i SQL databasen. Da vores prosumer og adresse er klart defineret er det mest oplagt at persistere dette data i en SQL. Der er valgt at have prosumer data og adresse data i to forskellige tabeller, hvor disse blot relaterer til hinanden. Der kan sagtens være adresser uden prosumers, men ikke prosumers uden adresser. Priserne er også klart defineret derfor har vi det SQL database. Transaktion data er valgt ikke at persisteret i en SQL da disse kan ændre sig undervejs og hvad man gerne vil have persisteret i den kan også ændres. Det hele vil blive sat op med EnitityFramework og efter Code-First, ved brug af migrations.

NoSQL

Vi har kun arbejdet med en Azure Document Database i I4DAB faget, så derfor var denne type af NoSQL mest oplagt til dette system. Her har vi valgt at persister vores transactions. Der vil blive oprette en collection for, Power, PendingTransactions og CompletedTransactions. Da disse kan ændre sig var det oplagt at persistere disse i en document database. Vores prosumer og adresse er klart defineret derfor er det ikke nødvendigt at persister dette data i en NoSQL, men det ville også være muligt at have en collection for disse.

Design af klasser – hvad skal de kunne

Der er oprettet forskellige modeller for vores system. Vores prosumers har alle et Id, så det gør det nemt at skelne den fra hinanden, et navn, en adresse, et adresse id og en prosumer type. Prosumer typen bestemmer om en prosumer er en boligejer, landmand eller virksomhed. Vores adresse indeholder blot et Id, vejnavn, vejnummer samt by og postnummer. SmartGridInfo indeholder data såsom ombytningsratioer fra strøm til tokens, prisen fra National Power Grid og et tidspunkt hvornår disse priser er oprettet.

Beskriv tankegang omkring klasser, vis klassediagram

General funktionalitet

Noget funktionalitet som blev valgt fra, var f.eks. i sammenhæng med prosumers login. Heri var der snak om at lave et password, så der skulle bruges et ”rigtigt” login, i stedet for bare at bruge et Id. Dette ville være smart hvis programmet skulle bruges af personer, så der var en privat bruger til hver husholdning/virksomhed eller person, men ift denne aflevering vurderedes dette for besværligt ift unikke passwords for hver person. Prosumers type, altså om det er en virksomhed eller en privat bolig, er også noget der er blevet overvejet hvor stor en indflydelse det skulle have på systemet. Den hovedsagelige forskel på disse to typer, blev set som værende mængden af kWh som der skulle købes og sælges. Hvor en husholdning ikke køber og sælger i stor stil, ville et firma måske skulle købe stort ind, hvis de skulle have flere store maskiner kørende hele natten, eller hvis et firma producerede strøm som erhverv. Dette blev ikke set som en større vigtighed for programmet, idet opgaven omfatter brugen af databaser mere en hvor store tal der arbejdes med.

UI, GUI eller web application

Vi har overvejet om der skulle udvikles en UI, en GUI eller en web client for den server client hvor de forskellige prosumer vil have mulighed for at logge in med deres Id. En UI er det mest simple at lave og kræver ikke noget af den maskine som systemet skal køre på, men det er sværere at gøre dataet præsentabelt med et konsol program. En GUI gør det nemt at få overskud og data er let at læse hvis den er lavet ordentligt, men det vil tage noget længere at implementere og det er ikke fokus for denne opgave. En web application vil nok være det mest optimale, da det ville være nemt for prosumers at tilgå deres smart grid fra deres telefon, pc mm. Denne løsning er dog for kompliceret. Vi har valgt at udvikle en UI, og har lavet nogle hjælpe klasse for at formatere dataet, så det bliver præsenteret pænt på konsol vinduet.

Implementeret design

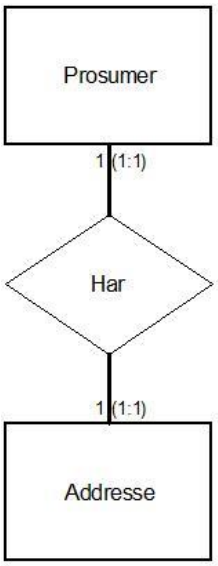
I denne del vil der blive gennemgået hvordan vi har implementeret vores design. Der vil blive gennemgået hvordan de valgte databaser er implementeret, hvordan klasserne er implementeret, hvordan programmet kører med noget essentiel kode og om hvordan vores UI er sat op.

Valgte databaser

I denne del beskrives hvordan vores SQL og NoSQL databaser er implementeret og realiseret. Der vil blive gennemgået hvad de hver især har til ansvar for.

MsSQL

Vores SQL har til ansvar for at persister vores prosumer data, adresse data og SmartGridInfo data. Vores prosumers har en adresse, men vores SmartGridInfo skulle have været en database for sig selv. Men dette er blot en tabel i den database vi har fået udleveret af universitetet. Denne er sat op med Enitity Framework og ved brug af Code-First migrations. Vores prosumer og adresse data er kodet i en configuration.cs i dens seed method. Så hvem der er i vores Smart Grid Village ligger fast. Der er oprettet 32 husstande, 10 virksomheder og 3 landbrug som vores prosumers. Der er også oprettet et SmartGridInfo som bare er basis information. Vores forskellige entities er kun prosumer og adresse i forhold til hinanden, er er lavet følgende ERD for hvordan det er sat op.



Figur 1, viser ERD for Prosumer og Adresse

Vi har valgt at sætte op så der er en prosumer pr. adresse, denne prosumer er så hvem der styrer strømmen i hhv. bolig, virksomhed og landbrug. Det er ikke nødvendigt med et ERD for SmartGridInfo, da dette skulle have været en isoleret database og ikke relateres til de andre tabeller i databasen. SmartGridInfo kan så blive opdateret hvis priserne ændre sig undervejs. Der er også lagt en REST Api henover så det er muligt at til gå data i databasen ved brug af http kald.

NoSQL - Azure Cosmos DocumentDB

Meget af den data der omhandler transaktioner med strøm i Smart Grid skal flyttes en del, og skal også samtidig være mindre dokumenter. Disse bliver derfor lagt i tre collections: PowerInventory, PendingTransactions, CompletedTransactions. PowerInventory står for at holde på det strøm der bliver solgt. PendingTransactions håndtere at lave transaktioner inden de kan blive afstemt. CompletedTransactions er de færdiggjorte transaktioner, som virker som en form for log. Der er lavet et library med generiske repositories og en factory til repositories, så dette kan genbruges. Oven på dette library er lagt ASP.NET Web Api 2. Udover de regulere funktioner i Web Api'et, er der lavet to query funktioner, et eksempel på disse er:

```
// QUERY: api/PendingTransaction/5
        [HttpGet]
        public IEnumerable<PendingTransaction> Query(string options)
        {
            return WebApiApplication.PendingTransactionRepo.WhereQuery(options);
        }
```

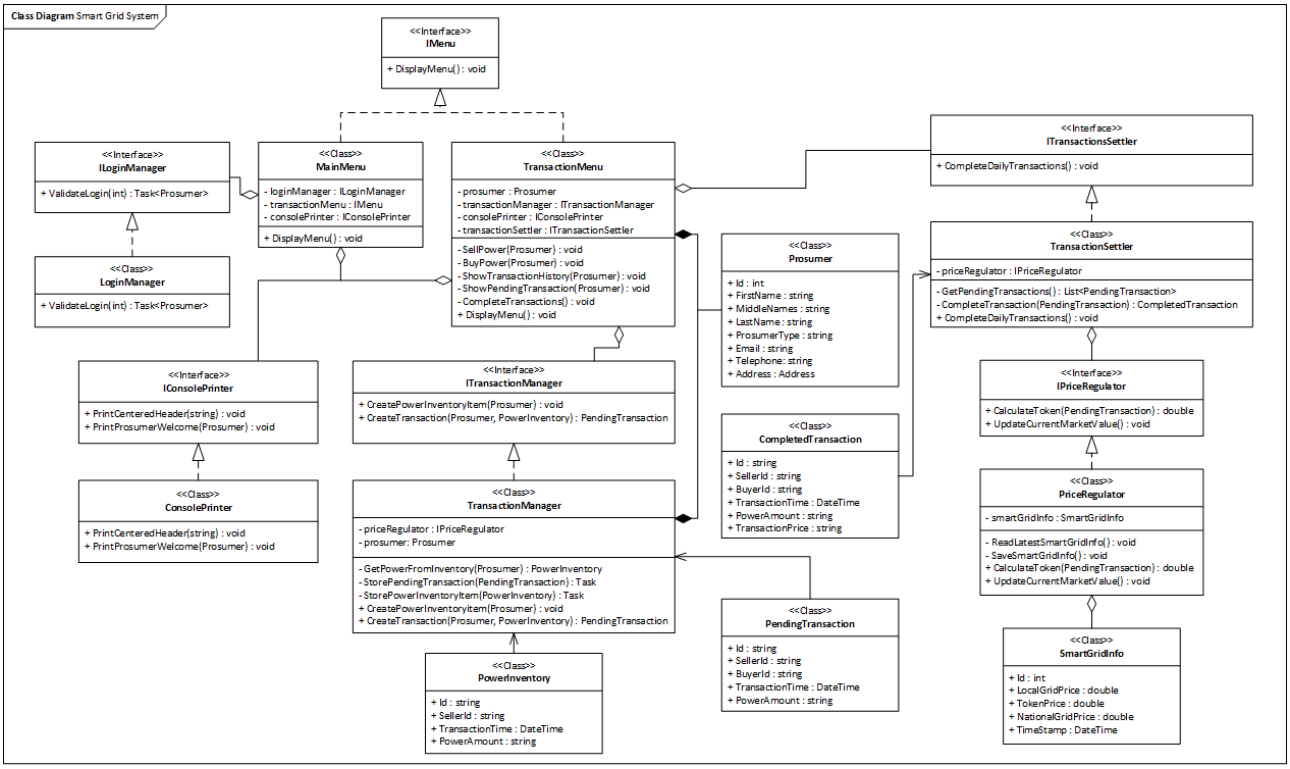
Denne funktion bruges i på modtager siden i denne funktion:

```
private void ShowPendingTransactions(Prosumer prosumer)
        {
            var getPath = AzureWebApiCaller.Client.BaseAddress +
             "/PendingTransactions?queryOptions=" + prosumer.Id;
            var receive = AzureWebApiCaller.Client.GetAsync(getPath).Result.Content.
             ReadAsAsync<List<PendingTransaction>>().Result;
            Console.Clear();
            foreach (var pendingTransaction in receive)
            {
                Console.WriteLine(pendingTransaction.ToString());
                Console.WriteLine();
            }
            Console.ReadKey();
        }
```

Forbindelsen til databasen er implementeret med Singleton, for stabilitet til Azure Cosmos.

Klasser – hvem gør hvad

Programmet består af en masse forskellige klasser, som kan ses på det overordnede klassediagram herunder.



Figur 2, Viser klassediagram for hele systemet

Nogle af klasserne repræsenterer data fra databaserne. Dette er SmartGridInfo, PendingTransaction, CompletedTransaction, PowerInventory og Prosumer.

SmartGridInfo indeholder prisen for strøm i det lokale grid samt i det nationale grit. Derudover prisen på token, et Id og et timestamp så det er muligt at se prisen på et givet tidspunkt.

PendingTransaction og **CompletedTransaction** indeholder begge en købers samt sælgers Id, en Id for handlen, tidspunktet handlen tog sted samt mængden af kWh der blev forhandlet. Derudover indeholder CompletedTransaction også en pris for handlen, hvilken en transaction først bliver tildelt når den går fra pending til completed.

PowerInventory er lavet for at opbevare kWh i det lokale grid. Den bliver lavet hver gang der sælges strøm til det lokale grid, og indeholder derfor en PowerAmount for det enkelte salg. Derudover har den en TransactionTime, for hvornår strømmen blev solgt til det lokale grid, en Id og en SellerId.

Prosumer indeholder den nødvendige info som skal bruges for at kunne benytte SmartGrid. Alle prosumere skal have et FirstName, MiddleName og LastName, til identifikation. Derudover personlige oplysninger som Email, Telefon og en adresse. Vigtigst indeholder prosumer også en prosumerType. Denne type bruges til at kende forskel på om den pågældende prosumer er en privat person eller om det er en virksomhed.

Menu klasserne, som arver fra IMenu, er brugt som UI til brugeren. I MainMenu skal der indtastes brugerens Id, hvorefter man bliver sendt videre til TransactionMenu. Der overvejedes at lave et password hertil så hver bruger havde private oplysninger, men eftersom dette ikke var en del af opgaven blev dette udeladt.

I **TransactionMenu** kan brugeren, som er logget ind, sælge og købe kWh, og se pending og completed Transactions. Dette kører igennem en konsol.

LoginManager bruges af MainMenu, og finder den prosumer, som har det indtastede Id fra MainMenu. Herefter hentes den tilsvarende prosumers informationer.

ConsolePrinter er en hjælper til vores menuer, som kan udskrive ting som konsolen ønsker skrevet flere gange.

TransactionManager står for en stor del af systemets tilgang til databaserne. Den har til ansvar at lave PowerInventory klasserne ud fra de indtastede data fra TransactionMenu. Derudover skal den, hvis der købes strøm, finde en sælger i det lokale grid og lave pendingTransaction. Hvis der ikke findes en sælger i det lokale grid, købes der fra "The National Grid" som beskrevet i opgaven. Herefter laves en PendingTransaction.

TransactionSettler sørger for at PendingTransactions bliver til CompletedTransaction. Den har derfor en PriceRegulator for at have adgang til værdierne på kWh samt tokens. Derudover har den adgang til PendingTransactions.

PriceRegulator opdaterer de informationer som gemmes i SmartGridInfo. Derudover kan den beregne token værdien af en pending transaction, hvilket TransactionSettler bruger.

Hvad gør programmet

Dette bliver gennemgået i detalje i den vedlagte video. Programmet starter med at bede om fra den prosumer som gerne vil logge ind på smart grid'et. Her indtaster prosumere blot sit ID og er nu præsenteret med en Menu, hvor prosumeren har til mulighed for at købe, sælge strøm, se Pending transactions og se completed transactions. Programmet henter prosumer data ned fra SQL databasen og når prosumere strøm vil der oprettet en Pending transaction i vores Document database

Test af design

Vores test af design kan ses i den vedlagte video. Det ville være svært at forklare præcis hvordan dette program fungere i bare rent tekst.

Diskussion

I dette afsnit vil vi komme ind på hvilke problemer vi er løbet ind i undervejs, hvad der er godt og skidt ved et smart grid, hvordan vores smart grid afviger fra et rigtigt smart grid.

Problemer undervejs

De fleste problemer vi mødte undervejs var at få data korrekt sendt rundt i mellem de forskellige klasser. Selve databaserne havde vi ikke mange problemer med, der kunne bruges kode og viden fra de andre opgaver i I4DAB til at lave disse. Vi havde et problem med at de data vi fik ind fra Azure ikke blev deserialized ordentligt, så dataet ikke kunne passes rundt i programmet. Det viste sig at version af Newtonsoft.Json var forskellige imellem projekterne og det gjorde at den ikke var enig om hvordan objekterne skulle håndteres. Da alle fik samme version var problemet løst. Da vi ikke har adgang til at se dokumenterne som bliver lagt op på universitets Azure, er det svært at vide om data nu bliver gemt korrekt, men det hele er verificeret til at virke på LocalDB, og programmet virker stadig efter hensigten selv efter vi har opkoblet til universitets Azure.

Pros and cons smart grid

Det smarte ved et smart grid er at alle prosumers har mulighed for holde øje med deres strømforbrug og selv regulere om de skal købe eller sælge strøm Dette kan gøre strøm billigere og der er undgås spild af strøm. Det kan også skaleres op og ned. Der kan være et nationalt smart grid hvor større byer og kommuner vil kunne forhandle strøm, samt et ”globalt” smart grid hvor lande imellem kan handle strøm.

Grunden til at smart grid ikke er brugt verden over endnu er at det er for dyrt for mange lande/byer at omstrukturere hele deres el net. Større institutioner vil have en konstant brug af Strøm til at opretholde varme, servers mm. Dette kan betyde at prisen for strøm i nogle perioder kan blive meget dyr for den enkle bruger. Der kan også være sikkerhedsbrister i form af det data som Smart meter gemmer og sender. Dette kunne være nemt tilgængeligt for hackere, da teknologien er meget ny og muligvis uden ordentligt sikkerhed.

True Smart Grid Vs Vores

I et True Smart Grid ville der være nogle ekstra features. F.eks. ville en køber få strøm fra den tætteste sælger i stedet for en tilfældig sælger, hvilket sker i vores smart grid. Herudover findes der jo ingen

"national smart grid" som kan tage uendelige mængder af strøm. Priserne som bruges vil være reguleret af noget mere advanceret end vores price regulator, og være baseret på rigtig valuta. Prosumers ville have private brugere til at købe og sælge kWh på, eller måske have automatiseret registrering så dette var helt unødvendigt, samt automatisk registrering af salg og køb af kWh.

Konklusion

Vi har udviklet et Smart Grid, som virker på en mindre by, med 32 husstande, 10 virksomheder og 3 landbrug, her er det muligt for de forskellige prosumers at købe og sælge strøm til hinanden. Selve databaserne står for at persistere data om hvem prosumerne er, hvilke køb der er lavet et i givet tidsvindue, og alle køb som er blevet gennemført. Det hele er koblet op på et større National Smart Grid som har til funktion at opretholde strøm hvis der skulle være dyk i produktion af strøm, eller købe strøm som er tilovers ved overproduktion. Hver prosumer har sin egen Server Application hvor de kan logge på med deres id og til går smart grid'et. Lige nu er det dog muligt for alle, at tilgå alles køb og salg af strøm, da man blot bruger Id til at logge på med, uden et password. Vores application vil mere var den som en server administrator ville have. Da den også har mulighed for at gennemføre alle, ventende transaktioner. Der er lagt et REST Api over alle databaserne, det gør det muligt at tilgå data ved brug af http kald. Så disse blot kunne benyttes i vores C# kode.

Systemet virker som et smart grid, og er koblet op til National Smart Grid, hvis prosumers prøver at købe strøm som ikke er der vil programmet ikke crashe.