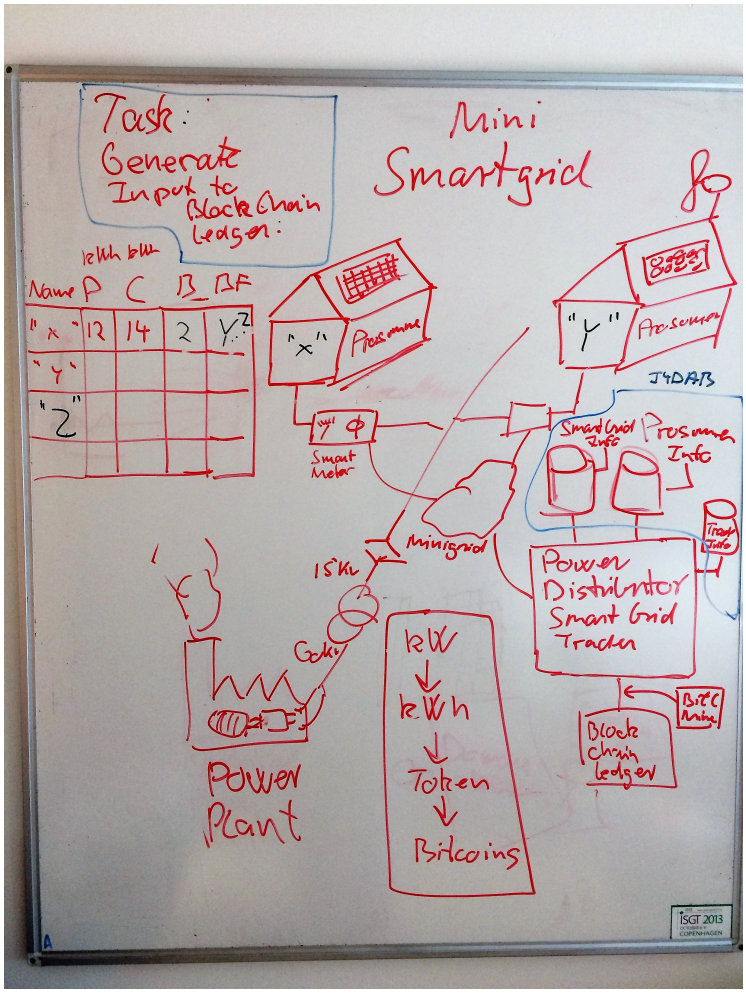
Efterår 2018
Ingeniørhøjskolen Aarhus Universitet

DAB - Handin 4

Gruppe 13

Village Smart Grid



Studienummer	Navn	Studieretning
201610450	Rasmus Brask Rolsted	IKT
201606374	Jakob Levisen Kvistgaard	IKT
20119144	Ramtin Asef-Nakhai	IKT
201408675	Parweiz Hagshenas	IKT

Dato: 14-12-2018

AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

Indholdfortegnelse

1 Indledning

1.1 Case

Der skal udarbejdes et system til at definere et såkaldt SmartGrid, de indeholder oplysninger om forbruger/producerende elkunder på elnettet. I deete grid skal det kunne defines i applikationslogik, hvor meget en forbruger har produceret eller frobrugt og en given periode.

Derudover forventes det, at der i forbindelse med el handler indbyrdes laves et logistisk system, til at define, hvilke handler der er foregået. Både tidligere, nuværende og fremtidige. Per deifiniton skal dette kunne defeineres, alt efter perioden man vælger at afregne på.

Alle informationer omkring SmartGrid'et skal kunne findes fra en specifik database. Hvorefter applikationslogikken kan finde de resterende informationer i andre tilgængelige databaser.

1.2 Krav

Kravene for opgaven er som følger.

1.2.1 Problemdomænets definitoner

- Databasen Smart Grid Info: indeholder en beskrivels af konfigurationen af det givne Mini Smart Grid.
- Databasen Prosumer Info: indeholder oplysninger og beskrivelse af de "Prosumers"som er med i den given Mini Smart Grid.
- Databasen Trader Info: indeholder oplysninger om, hvorledes der er handlet, hvorledes der handles lige her og nu og hvorledes der skal handles fremover. Tabellen i øverste højre hjørne af billedet er en skematisk beskrivelse af nogle af oplysninger i Trader Info DB.
- Smart Meter: En IOT enhed eller mere konkret en enhed, som måler og opsamler de konkrete data omkring strøm, strømproduktion, strømforbrug og status iøvrigt for den pågældende Prosumer.
- Power Distributor/Smart Grid Trader: Systemet som er forsyningsselskabet bag Mini Smart Grid bruges til at overvåge, styre og kontrollere Smart Grid. Trader delen står for at håndtere de indbyrdes salg mellem de enkelte Prosumers.
- Power Plant: symbolisere en typisk central elforsyning uden for Mini Smart Grid.

1.2.2 Kravspecifikation

- Udvikles 3 databaser - Trader Info, Prosumer Info og Smart Grid Info.
- De nævnte databaser skal udstyres med Front End REST API, hvor nødvendige metoder afhænger af interaktionen med resten af systemet.
- MiniSmartGrid i denne opgave kaldes 'Village Smart Grid', til 33 husstande og 12 virksomheder/landbrug.
- Der gælder en række karakteristika, som har relevants for MiniSmartGrid i forhold til hver overordnet type husstands-prosumer og virksomhedsprosumer. Disse skal kunne registreres.
- Afregning skal ske på basis af kWh-blokke. Smart Meter måler her og nu strøm og spænding på Prosumers elstik.
- Differenesen mellem ind- og udgående blokke aflæses i forhold til en given tidsperiode kaldte afregningsvinduet.
- Der er en "bryder"mellem "The Village Smart Grid"og resten af Danmark.
- Resten af Danmark skal betagtes som en et stort Smart Grid 'The National Smart Grid'.

- Hver Prosumer i Mini Smart Grid er kendt via sin kobberforbindelse (sit elstik), sine karakteristika og sit Smart Meter.
- Afregningsprincip et det princip som Prosumers bruger til at afregne kWh-blokke indbyrdes.
- Der kan inddrages et salgsvindue eller et købsvindue, hvor et salgsvindue er den tid og det tidsum en prosummer vil stille/stiller en mængde kWh-blokke til rådighed, og hvor et købsvindue er det er den tid og det tidsrum en prosumer vil købe kWh.blokke.
- I et givent afregninsgsvindue gælder en bestem pris for en kWh-blok. Hvis priserne for en kWh-blok ønskes dynamiske i forhold til et bestemt elmarked gøres aflæsningsvinduet kortere men ønskes en mere fast afregningsstruktur gøre afregningsvinduet længere.
- En Prosumer har mulighed for selv at bestemme, hvor mange kWh-blokke, der ønskes købt eller solgt, samt hvilke købs- og salgsvinduer der er gældende.
- Alle handler afsluttede, igangværende og kommende kendte handler skal registreres.
- Alle afregninger der er gennemført , under gennemførelse og som er kendt til at ville blive gennemført, skal registreres.

1.2.3 Tekniske Krav

- Den tekniske platform der udvikles på er Microsoft .NET. Nyeste gældende versioner.
- Enten .NET Standard eller .NET Core eller et miks af begge.
- ADO.NET Entity Framework, EF eller EF Core, benyttes mod SQL databaser. Nyeste gældende versioner.
- Azure Cosmos DB SQL API .NET benyttes Dokumentdatabaser. Nyeste gældende version.
- Der må gerne udvikles en eller flere testklienter som bruger de udviklede REST API'er.
- Tilsvarende må Swagger Swashbukle tilføjes REST API servererne for test af REST API.
- Tilsvarende med predefinerede request der sendes via REST klienter som Postman REST Client eller Advanced Rest Client må bruges.
- Der udvikles på SQL Express LocalDB og Azure CosmosDB SQL API Emulator.

1.2.4 Arbejdsplan

Efter at gruppen har kigget og undersøgt opgaven, er der besluttet at vi har valgt at dele opgaven ud således at vi har en mand på hver database udover dokumentdatabasen Trader info hvor vi har valgt at sætte 2 mand på, da vi mente at det var der som vil være mest arbejde at lave. Følgende personer har arbejdet:

- Ramtim: SmartGridDB og rapportskrivning (Design, DDD og Konklusion)
- Rasmus: ProsumerDB og rapportskrivning
- Jacob: SmartGridDB, ProsumerDB, TraderDB og oprettelse af DDD-diagrammer
- Parweiz: TraderDB og kontrol og udarbejdelse af CRUD-operationer vha. Swagger.

2 Design

Der er til denne løsning benyttet 1 NoSQL-database og 2 SQL-databaser for at overholde kravet om mindst 1 af hver jf. kravsspec.

Gruppen har valgt at lade Traderdatabasen være NoSQL, da dokumentdatabasen blev vurderet som den mest fleksible, idet den ikke er så låst fast som SQL-databaserne.

I forhold til den indledende fase og design har gruppen ikke valgt at bruge et ERD til at understøtte designet. Gruppen har i stedet i fællesskab tegnet systemet op i hånden og diskuteret, hvorledes klasserne og databaserne skulle opsættes. Det samme gjorde sig ligeledes gældende med Handin 3.2, del2, hvilket virkede fint. Skulle man dog overdrage systemet til en anden udvikler eller skulle der ske hurtig refaktorering vil et ERD selvfølgelig være en fin hjælp at have. Og som med så mange andre diagrammer, styrker det forståelsen af det system som man arbejder med.

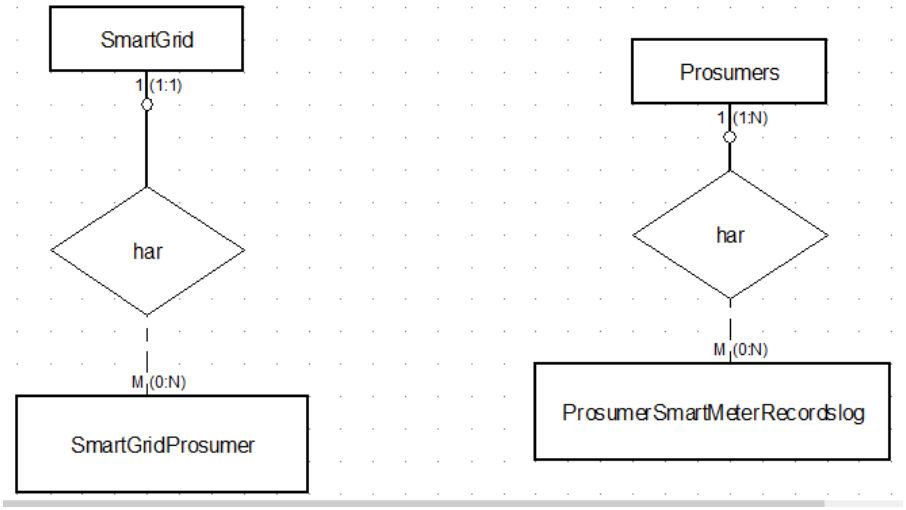
Gruppen har ligeledes hurtigt besluttet - med accept fra underviser Jesper Tørresø - at de 3 databaser ikke nødvendigvis skulle snakke sammen. Dette er derfor ikke implementeret i systemet, men vil være en oplagt del at fokusere på i det fremtidige arbejde.

Den tekniske platform er udarbejdet på .NET Core med MVC- og Repository-pattern. Der er benyttet Swagger til udarbejdelse af Rest-Api'er. Denne fremfor Postman, da gruppen har bedre erfaringer med Swagger.

Til gennemførelse af gruppens opgave er der taget stor inspiration fra medlemmernes løsning af Handin 3.2, del 1 og del 2.

2.1 Entity Relationship Diagram(ERD)

I forbindelse med uarbejdelsen af de valgte tabeller i i respektive databaser, (hvor vii forestilles os stadig 2 databaser som foreskrevet i opgaven, selvom tabellerne i projektet er implementeret i samme database) er der implementeret 2 nedstående ERD diagrammer fo ProsumerInfo og SmartGridInfo databasen.

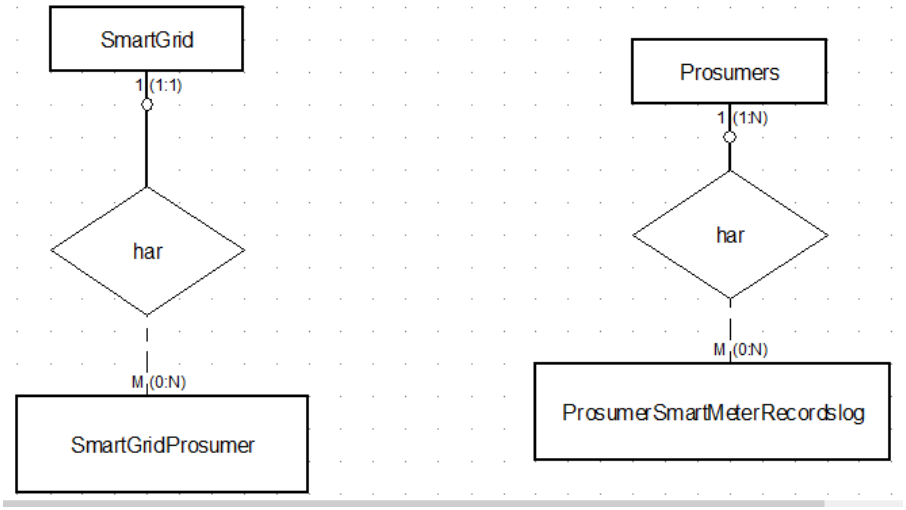


Figur 1: Opstilling af ERD-diagram for SmartGridInfo og ProsumerInfo

Overstående figur 1, viser relationen mellem tabellerne i de respektive databaser.

2.2 Database Studio Design (DSD)

I figur 2 ses både Primary og Foreign key for de 2 moddeller. Som vi har arbejdet ud fra i forbindelse med implementeringen af DDD diagrammerne.



Figur 2: Opstilling af ERD-diagram for SmartGridInfo og ProsumerInfo

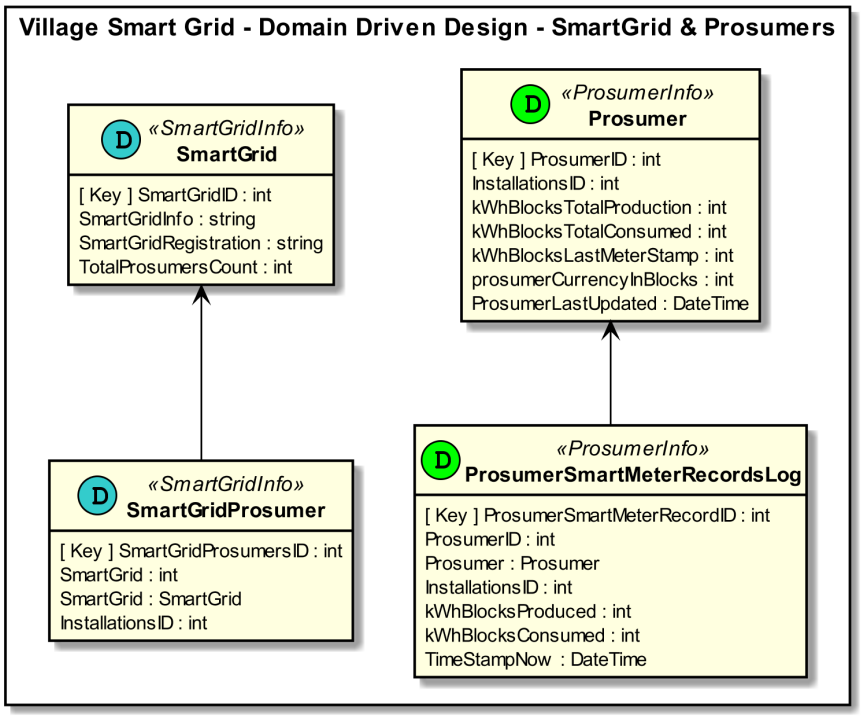
2.3 Domain Drive Design (DDD)

I vores DDD har vi lavet 2 diagrammer, hvor der i det ene er vores Smartgrid og Prosumer, mens der i den anden er vores Trader.

I det første diagram, SmartGrid er der 2 klasser, hhv. SmartGrid og SmartGridProsumer. SmartGrid indeholder udover en PK også information om den valgte smartgrid og hvor mange prosumers, der er.

SmartGridProsumers inderholder info omkring prosumers for hvem, der er i dette grid. Blandt andet deres InstallationsID og SmartMeterID.

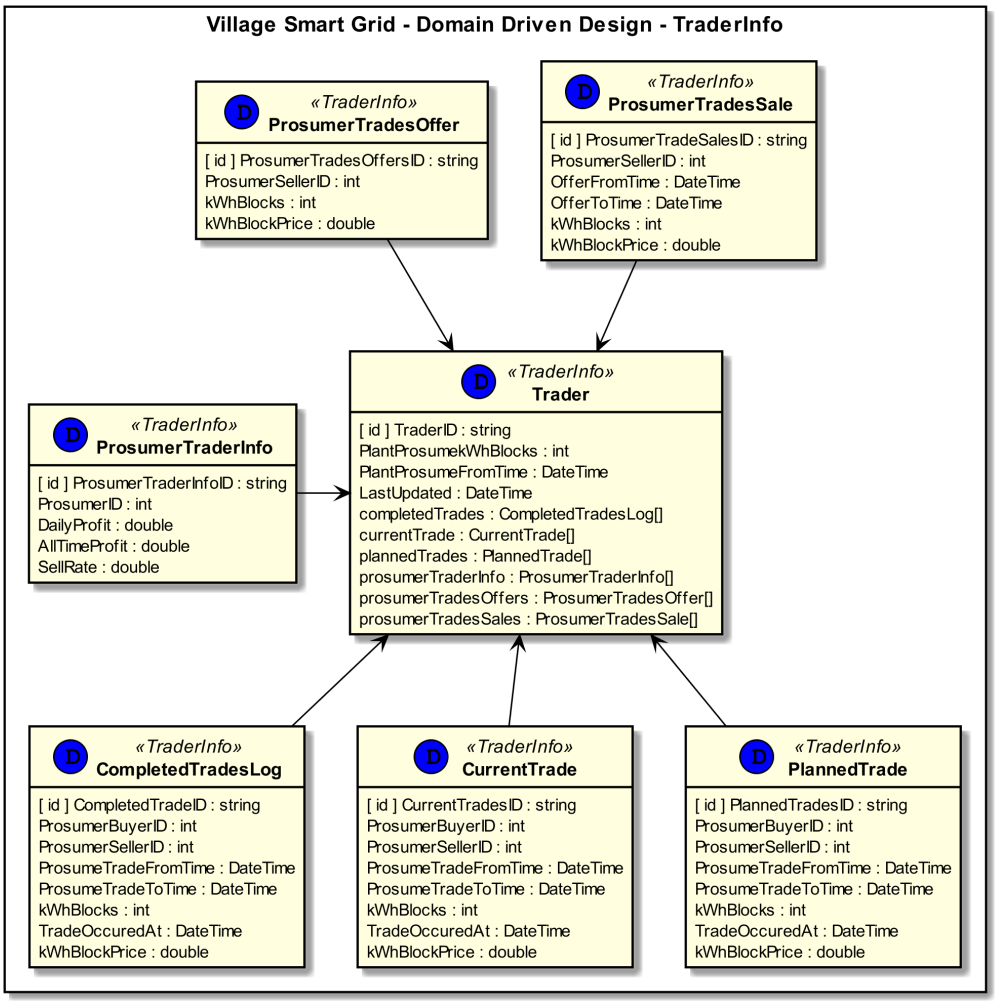
I Prosumers findes ligeledes 2 klasser, hhv. Prosumers og ProsumersSmartMeterRecoresLog. Den første indeholder udover et ProsumerID bl.a. info omkring total produktion og total forbrug af kWh-blokke. Den anden klasse indeholder tidsstemplede elementer med prosumerID for hver gang, der sendes data fra en given elmåler/smartmeter.



Figur 3: Opstilling af DDD-diagram af ProsumerInfo og SmartGridInfo.

I det andet diagram findes der nogle flere klasser, da det er en del mere kompleks. Vi vil i det følgende kort beskrive hver klasse:

- Trader: info omkring trader-delen ift. grid, total brug af kWh fra plant og hvis den er negativ viser den forbrug. Traderklassen er forbundet til de øvrige klasser, hvorfor der er et objekt eller en liste for hver enkelt af de øvrige klasser i databasen, hvor det har givet mening. Dette særligt, da der er tale om en dokumentdatabase.
- ProsumerTraderInfo: Hver prosumers info omkring totalproduktion, profit og ”sellrate”.
- CompletedTradesLog: En ny række for hver enkel handel for en given periode med køber, sælger og pris for, der er aftalt.
- CurrentTrades: Igangværende trades, køberID, sælgerID samt antal kWh blokke og pris.
- PlannedTrades: Fremtidige trades, køber- og sælgerID samt stamps fra og til og antal blokke.
- ProsumerTradesSales: Liste over specielle tilbud fra en prosumer, det pågældende tidspunkt, antal blokke, bestemt af en tidsmarkør for hvornår folk kan købe til prisen.
- ProsumerTradesOffer: Indeholder prosumerID, antal blokke og deres pris.



Figur 4: Opstilling af DDD-diagram af TraderInfo.

3 Konklusion

Som der er gjort bemærket, så er det lykkedes at lave et system bestående af tre databaser.

- En documentDB
- To relationelle databaser

Til at kontrollere og arbejde med de 3 databaser er der udarbejdet 3 REST-Api'er, hvor CRUD-operationerne er testet med success. Systemet vil derfor kunne udleveres og benyttes til at håndtere de gældende SmartGrids, Prosumers og de trades, der har været.

Der er i databaserne simuleret dataoplysinger for fiktive personer og deres tilhørende oplysninger. Dette kun for kontrollere systemet. I et virkeligt design vil der være gjort brug af rigtige data, hvilket også vil have givet et måske bedre billede af oplysningerne. Dette har dog ikke den større betydning for opgaveløsningen, hvor funktionaliteten, forståelsen og opsætningen har været i fokus.