Danmarks Tekniske Universitet
Lyngby 2013

Business Monitor Dashboard Migration

IT-Diplomingeniør eksamenprojekt udført hos

Author:
Javid Bahramzy

Supervisor:
Bjarne Poulsen

External supervisor:
Frederik Kiær

IMM-B.Eng-2012-46

Business Monitor Dashboard Migration

Javid Bahramzy

August 2013

Technical University of Denmark
Department of Applied Mathematics and Computer Science, DTU Compute
Matematiktorvet, building 303B, DK-2800 Kgs. Lyngby, Denmark
E-mail: compute@compute.dtu.dk
Phone +45 4525 3031, Fax +45 4588 2673
EAN-nr. 5798000428515

IMM-B.Eng-2012-46

Summary

There are many client-side technologies available on web today (including Adobe Flash, Microfsoft Silverlight), the most popular is without any doubt HTML5/JavaScript due to its widely support across all modern browsers. The biggest advantage of HTML5/JavaScript over Silverlight and other plug-in technologies is that it does not require a plug-in. The user does not have to install any other software to view a HTML5 page. This means that HTML has the farthest reach. Microsoft itself is shifting from Silverlight to HTML5 and admits at the same time that HTML is the only true cross-platform solution. This has led to some speculations about future of Silverlight.

The Business Monitor application is developed in Silverlight, which is a browser plug-in technology from Microsoft. The aim of this project is to examine and attempt, how to pave the way for the Business Monitor application to migrate to a cross-platform HTML5 alternativ that can be utilised on different sized screens. The last part is something that can be achieved throught the use of Responsive Design. Using Responsive Design you only need to maintain one web application and one design. The design will automatically adapt itself based on the screen size of the device.

One of the main issues with Silverlight is that the technology is poorly supported on IOS—and it is totally out of the question on the mobile and tablet platform. On MAC computers Silverlight performs ok using Safari and Firfox (although there are some issues with memory leaks)—but with Chrome text fields are not working.

Keywords:
HTML5, JavaScript, jQuery, AJAX, Kendo UI, Knockout, MVVM, ASP.NET WEB API

Resumè

Der findes flere browserbaserede klient-side teknologier såsom Adobe Flash, JavaFX og Silverlight, men den mest berømte for tiden er uden tvivl HTML5/JavaScript grundet dens udbrede understøttelse blandt de moderne browsere. Den seneste udvikling inden for RIA (Rich Internet Application) viser, ifølge tal fra Google Trends fra september 2012, at plug-in baseret rammeværker (frameworks) langsomt er ved at blive erstattet af HTML5/JavaScript alternativer. HTML5/JavaScript alternativer som AJAX bruger indbygget browserfunktionalitet, i modsætning til pluginbaserede løsninger som fx Silverlight, som anvender et software rammeværk.

Business Monitor -applikationen er implementeret i en sådan pluginbaseret teknologi fra Microsoft, nemlig Silverlight. HTML5 standarder har udviklet sig, og browserkompatibilitet med disse standarder er blevet noget bedre. Selv Microsoft erstatter Silverlight med HTML5 på nettet[1], og dermed erkender de, at HTML er den sande tvær-platform-løsning. Dette medfører, at der spekuleres i Silverlights fremtid. Der sættes spørgsmåltegn ved, om der kommer en version 6 af Silverlight[2].
Dette projekt har til hensigt at undersøge, forsøge og bane vejen for, hvorledes Business Monitor-applikationen kan migrere til et tværplatform alternativ ved brug af en plug-in-fri teknologi, som har en sikker fremtid. Meget tyder på, at det har Silverlight ikke.

Business Monitor A/S vil gerne nå ud til de forskellige webplatforme—også de mobile. Business Monitor har oplevet, at enkelte kunder har afvist løsningen med den begrundelse, at man ikke har kunnet køre Business Monitor på sin IPAD. På de mobile platforme er Silverlight fuldstændig udelukket. På MAC maskiner fungerer Silverlight i Safari og Firefox, men der er konstateret problemer med memory leaks. I Google Chrome virker indtastningsfelter ikke.

Med HTML5/JS kan Business Monitor nå alle platforme, og spreder ud til den mobile platform, uden at skulle vedligeholde native apps til hver af de forskellige mobile platforme (ANDROID, IOS, WINDOWS PHONE 8), hvilket vil kræve en større udviklingskompetence og mere udviklingstid. Business Monitor er en lille virksomhed med begrænsede udviklingskompencer og er derfor nødt til, i første omgang, at satse på en webapplikation, der er i stand til at tilpasse sig forskellige skærmstørrelser gennem Responsive Design, som betyder i al sin enkelhed, at websitet justerer sig til den skærmstørrelse, brugeren anvender.

[1] Ifølge artiklen udgivet på *Version2*, der er et dansk online nyhedsmedie henvendt til it-professionelle. Link til artiklen: `http://www.version2.dk/artikel/microsoft-erstatter-silverlight-med-html5-paa-nettet-16799`

[2] `http://www.computerworld.dk/art/200402/microsoft-vi-siger-ikke-et-piv-om-silverlight`

Figurer

Tabeller

Listings

Indhold

Forord

Dette dokument er resultatet af afslutning på min IT-diplomuddannelse udarbejdet i perioden 1. marts 2013 - 1. august 2013 ved Danmarks Tekniske Universitet (DTU). Projektet er udført i samarbejde med virksomheden Innologic Business Monitor a/s under vejledning af Frederik Kiær, som er partner i Innologic Business Monitor a/s, og ekstern vejleder tilknyttet projektet. Projektet har desuden været vejledt af Bjarne Poulsen, Associate Professor ved DTU COMPUTE. Jeg ønsker herved at rette en stor tak til begge for deres professionelle vejledning og kontruktive feedback.

Også en stor tak til Kristine Kiær, der har været en stor hjælp i forbindelse med korrekturlæsning af rapporten.

Sluttelig vil jeg gerne takke mine dejlige børn, Arain- og Sanja Bahramzy, for deres rummelighed og tålmodighed under udarbejdelsen af dette projekt— også selvom det til tider var dejligt solskinsvejr udenfor.

Prototypen
Prototypen udarbejdet i forbindelse med dette projekt er tilgængelig på nedenstående webadresse.
Se i øvrigt vejledningen i Appendiks A afsnit Manual & skærmbilleder

https://bmjavid.cloudapp.net/

Kapitel 1

Introduktion

Dette kapitel beskriver projektets baggrund, vision og omfang samt en beskrivelse af problemområdet. Desuden beskrives virksomheden, projektet er udført hos, og en plan for projektets løbetid. Kapitlet har til formål at give en bedre forståelse for projektets problemområde.
En beskrivelse af den anvendte udviklingmetode vil også være at finde i dette kapitel. Kapitlet vil afslutningsvis komme ind på, hvordan rapporten er struktureret, og hvordan projektets problemdefinition vil blive besvaret gennem de efterfølgende kapitler.

1.1 Virksomheden

Innologic a/s[2] er et partnerdrevet konsulenthus - etableret i 2005 af Lars Millgaard og Keld Pilegaard – med primær fokus på Business Intelligence. Innologic[1], med hovedkontor i København på Fruebjergvej 3, er i dag blandt Danmarks førende konsulenthuse indenfor Business Intelligence. Innologic besidder særlig ekspertise indenfor SAP Business Warehouse og SAP Business Objects.

Forretningsgrundlag Virksomheden leverer BI[2] løsninger til store- og mellemstore virksomheder på det danske og nordiske marked. Innologic har både forretningsmæssige kompetencer og tekniske kompetencer indenfor datamodellering, programmering og test. Med kombination af den forretningsmæssige- og tekniske kompetence er Innologic i stand til at skabe overblik ved at transformere data til information. Dette er også, hvad deres seneste produkt Business Monitor gør, nemlig at skabe overblik.

Innologic Business Monitor a/s
Innologic Business Monitor a/s[2] (herefter betegnet som Business Monitor a/s)

[1] `http://innologic.dk/`
[2] Business Intelligence

er i dag et søsterselskab til Innologic med kontor på samme adresse. Business Monitor er en cloud-applikation[3], der afvikles i en browserbaseret Silverlight-klient. Det er en Dashboard, Budgettering og Rapporterings løsning, som virksomheden lancerede på markedet i september 2012[4].

Med Business Monitor får man mulighed for at monitorere sin virksomhed vha. markedsdefinerede nøgletal som omsætning, dækningsbidrag, omkostninger og mange flere. Man har adgang til en række Dashboards, der er let tilpasselige efter behov. Man har mulighed for at analysere på de enkelte nøgletal både grafisk og i tabeller. For nøgletallene er der defineret grænseværdier, der kan justeres, som sørger for, at man blive advaret, hvis en nøgletalsværdi ikke er acceptabel.

1.2 Baggrund

Business Monitor a/s[2] er en Business-to-Business (B2B) virksomhed. Applikationen Business Monitor er udviklet i Silverlight[5], og er i første omgang henvendt til de virksomheder, som bruger regnskabsprogrammet e-conomic[5]. På sigt er planen at udvide Business Monitor til også at understøtte andre kildesystemer. Business Monitor er udviklet med denne udvidelse i tankerne, således at man let vil kunne integre mod andre regnskabssystemer og derved nå ud til en langt bredere målgruppe. Business Monitor henvender sig, ligesom e-conomic, til små og mellemstore virksomheder. Business Monitor er baseret på standard datastruktur, og det kræver ingen konfiguration af slutkunden.

På baggrund af sin Business Intelligence ekspertise, og det faktum at Innologic selv bruger e-conomic, har virksomheden investeret i Business Monitorprojektet, idet man mener at have de bedste forudsætninger for at lave løsningen, der skaber overblik for mindre virksomheder.

e-conomic a/s er virksomheden[6] bag et online regnskabsprogram ved samme navn. Regnskabsprogrammet er særligt udviklet til små og mellemstore virksomheder. e-conomic regnskabsprogrammet indeholder finans- debitor- kreditor- og fakturamoduler og kan udvides med ekstra funktionalitet. Man kan som en virksomhed blive e-conomics app-partner, hvilket vil sige, at man som virksomhed har udviklet et system, der kan forbindes med e-conomic regnskabsprogrammet.

Business Monitor er udviklet som en standard-app, der skaber forbindelse til e-conomic regnskabsprogrammet. Standard i den forstand, at programmet kan bruges af alle e-conomic kunder, og at løsningen ikke er kundetilpasset. Altså, at Business Monitor ikke er en kundespecifik app-løsning til e-conomic, men en

[3] Hostet i Windows Azure.

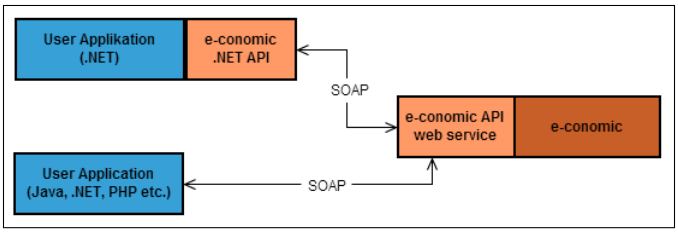
[4] `https://businessmonitor.dk/`

[5] Silverlight er Microsofts browserplugin, der giver mulighed for udvikling af rige Internetapplikationer (RIA).

[6] e-conomic er både navnet på et online regnskabsprogram og virksomheden, der står bag det - `http://e-conomic.dk/`

generel løsning, der kan anvendes af flere e-conomic kunder. Innologic Business Monitor er e-conomics app-partner.

Integration `Business Monitor` integreres med `e-conomic` via e-conomic API, som er et programmerings interface e-conomic stiller til rådighed for udviklere/virksomheder der gerne vil udvikle programmer til at kummunikere med e-conomic. Dette API er baseret på web services ved brug af SOAP 1.2, som de fleste moderne programmeringssprog understøtter. Desuden stiller e-conomic et SDK til rådighed til .NET udviklere som en alternativ til SOAP. `e-conomic` .NET API er et .NET `assembly`, som indkapsler al kommunikation med `e-conomic web service`. Der findes entitet klasser for alle `e-conomic` entiteter i .NET API. Således er der to måder at kommunikere med `e-conomic` API på: ved hjælp af `e-conomic` .NET API eller ved at kalde `web service` metoderne direkte via `e-conomic` API `web services`. Dette er illustreret via et flowchart diagram vist i figur 1.1 nedenfor.

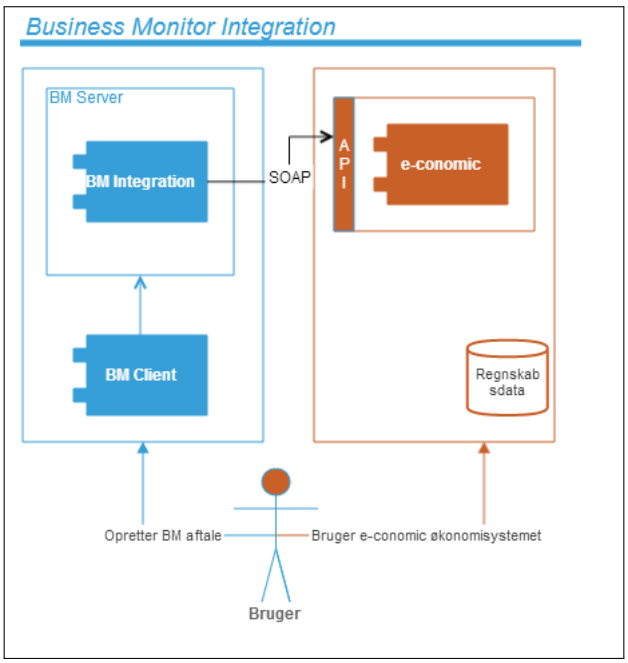


Figur 1.1: e-conomic API kommunikation

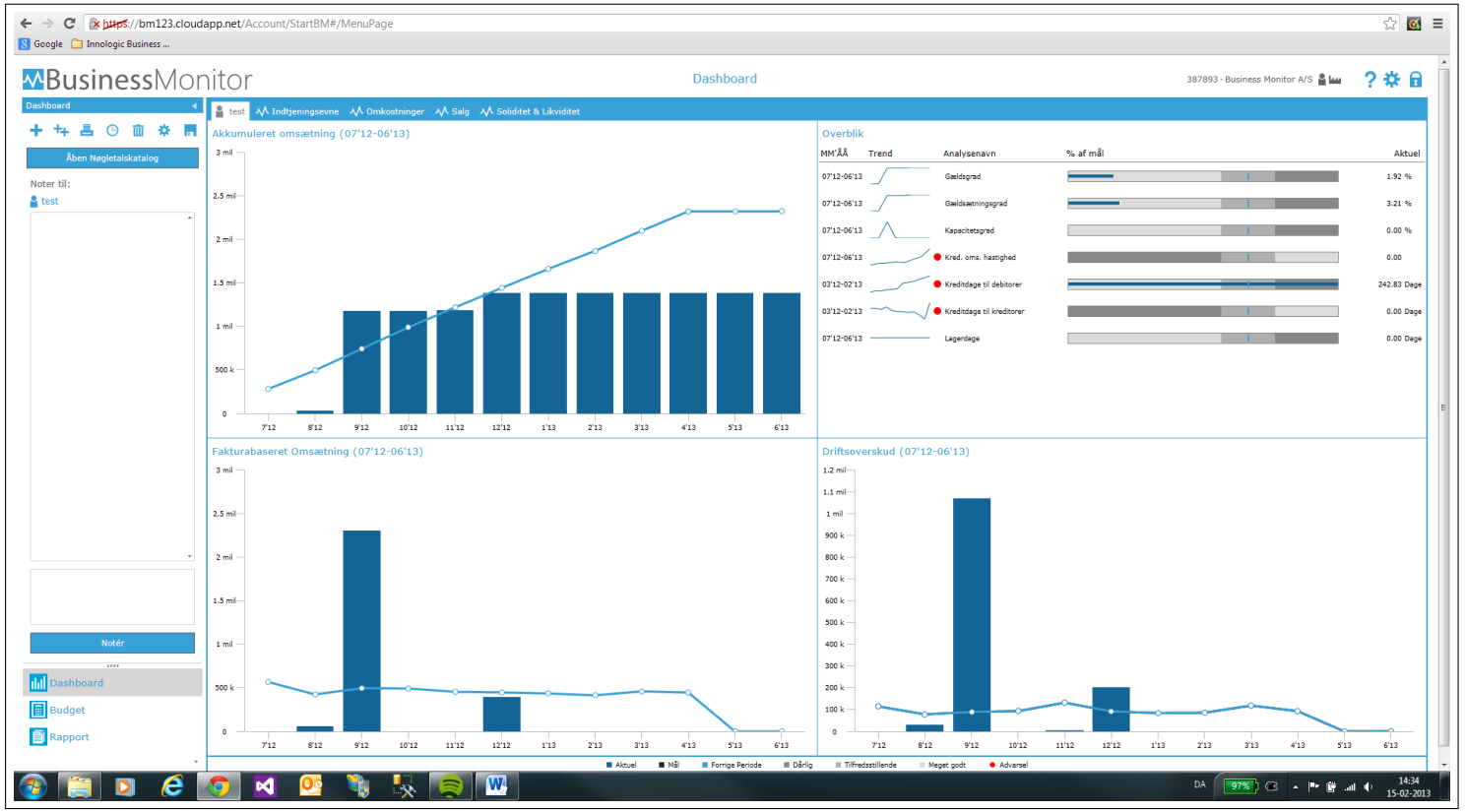
Hvordan virker det? Business Monitor får data fra e-conomic ved brug af en e-conomic brugeraftale. Dvs. man vil som en e-conomic bruger kunne få adgang til Business Monitor ved at oprette en aftale hos Business Monitor—se figur 1.2. Dette gør man ved at indtaste sine e-conomic oplysninger på Business Monitor websiden og trykke OPRET. Business Monitor vil herefter verificere de indtastede oplysninger hos e-conomic gennem deres API og hente data for denne e-conomic brugeraftale. Brugeren får efterfølgende en e-mail om den første integration med et link til Business Monitor websiden, hvor brugeren kan logge på.

Under integrationsprocessen beregner Business Monitor nogle finansielle nøgletal[7] på baggrund af kundens/virksomhedens data og udformer dashboards i Silverlight. Det kan tage nogle minutter afhængig af kundens datamængde. Figur 1.3 viser et eksempel på et Dashboard for en Business Monitor brugeraftale.

[7] Et finansielt/økonomisk nøgletal belyser et område af virksomhedens aktuelle økonomiske situation.



Figur 1.2: Business Monitor integration. Som e-conomic kunde har man mulighed for at oprette en Business Monitor aftale. Man bruger sine e-conomic oplysninger, når man opretter aftalen. Business Monitor henter herefter brugerens data hos e-conomic, og udformer Dashboards.



Figur 1.3: Regnskabsdata i form af Dashboard i `Silverlight`

1.3 Motivation

Personlig Der er ingen tvivl om at HTML5, og de potentialer den giver som platform for at udvikle cross-platform[8] applikationer, får stigende popularitet og fremdrift. Det er klart for enhver, at denne popularitet bag HTML5 er voksende. Jeg synes derfor, projektet giver nogle fremtidssikrede kompemtencer inden for området, som jeg i høj grad får brug for, både i mit videre studieforløb og i et fremtidigt arbejdsliv.

Virksomhed `Silverlight 5` kan være den sidste version af `Silverlight`, som `Microsoft` frigiver, idet `Microsoft` selv nu støtter op om HTML5 og CSS3. Det er derfor i virksomhedens interesse at satse på og søge mod fremtidssikrede teknologier. Med dette projekt håber virksomheden på at få svaret på spørgsmålet, om hvorledes `Business Monitor` applikationen kan genskabes baseret på en mere fremtidssikret teknologi.

Virksomheden vil gerne nå ud til flere platforme. Virksomheden har haft potentielle kunder, der har sagt 'nej tak' til løsningen, fordi de ikke kan bruge applikationen på deres IPAD. Man har ligeledes oplevet at kunder, der virkelig har været interesseret i løsningen, ikke kunne bruge det på deres MAC maskiner, og måtte gå på en PC for at kunne bruge det. En løsning kunne være at lave forskellige `native apps` og have `Silverlight` løsningen kørende. Men det kræver et API og en masse udviklingskompetencer-/ressourcer, som virksomheden ikke er stor nok til have på nuværende tidspunkt. Flere og flere bruger mobiltelefoner og tablets, så det er også virksomhedens ønske at kunne benyttes på disse apparatter.Virksomhedens udviklingskompetencer er begrænsede, og derfor giver det god mening at kaste tid og energi i en webapplikation med de muligheder, der ligger i HTML5/JS. Heri ligger virksomhedens motivation for projektet— at bane vejen for en sådan webapplikation.

Man vil meget gerne have, at tingene er så adskilt som muligt, fordi det gør det meget nemmere at vedligeholde. Man vil have en klar adskillelse af serverlogik og klient-side logik. Dette kan man opnå ved at have et WEB API, og et præsentationslogik lag på klientside, der spørger API'et.
`Silverlight` er rigtig tung nu og al logik ligger i klient applikationen. Virksomheden er derfor interesseret i et API, hvor alt forretningslogikken kan placeres.

1.4 Vision

Projektets vision er at udvikle en prototype webapplikation, som drager fordele af HTML5, `jQuery` og ASP.NET MVC 4, og derved kan afvikles i en browser på de ønskede apparater. Der ønskes udviklet en prototype web-applikation indeholdende `dashboard` løsning, med faciliteter som i den nuværende `Silverlight` applikation.

[8] På tværs af platforme

1.5 Problem definition

`Business Monitor` er en cloud-applikation, der afvikles i en browserbaseret `Silverlight`-klient. Virksomheden ønsker, at der undersøges muligheden for at genskabe `Silverlight`-klienten i en mere tilgængelig præsentationsteknologi (HTML5/JavaScript). Projektet vil omfatte analyse af teknologier, planlægning af udviklingsstruktur/mønstre og prototyping af dashboard-løsning med samme faciliteter som den nuværende `Silverlight` løsning. Med baggrund heri svarer projektet på følgende problemstillinger:

1. Anvendelse/bestemmelse af en udviklingmetode til styring af projeket.
2. Hvilke muligheder er der for udvikling af en dynamisk, interaktiv og responsiv web appliktion?
3. En analyse af hvilke teknologier der kan bruges og hvordan de skal bruges (`JavaScript, jQuey`, AJAX etc.).
4. Beskrive de funktionelle krav samt identificere og skrive `use cases`.
5. Med udgangspunkt i den eksisterende `Dashboard` løsning, at komme med et bud på hvordan en tilsvarende HTML5/JavaScript udgave kan se ud - hvordan et `dashboard` kan opbygges og hvordan de finansielle nøgletal kan repræsenteres i dashboardet.
6. Et nøgletalskatalog, hvor de finansielle nøgletal er dokumenteret med mulighed for at ændre indstillinger af nøgletal.
7. Få repræsenteret et nøgletal med mulighed for forskellige visninger (grafisk/tabelvisning).
8. Mulighed for tilføjelse af nøgletal til dashboard, hvor nøgletal vælges i nøgletalskatalog.
9. Udvikle en grænseflade (WEB API) til server kommunikation.
10. Persistence— mulighed for at gemme ændringer.
11. Test af prototypen og verificering af kravspecifikationen.

1.6 Afgrænsning

Hvor jeg under afsnittet 1.5 præsenterede problemstillingen, vil jeg i dette afsnit præcisere problemstillingens omfang, og hvad der primært er projektets fokus.

Med den nuværende `Silverlight` løsning bliver alt data og logik hentet ned på Silverlight-klienten. DVS. når klienten ændrer nogle indstillinger, der resulterer i, at tingene bliver reberegnet, ligger den nødvendige forretningslogik og data allerede til rådighed på klieneten i `Silverlight`. Og hvis tingene skal persisteres, laves der servicekald til serveren.
Projektets fokusområde inkluderer ikke programmering af dette forretningslogiklag. Dette lag placeres bag et API, som præsentationslaget (der er projektets fokusområde) kommunikerer med. Projektets fokus omfatter således også at bygge en kommunikationsstruktur mellem server (den nævnte API) og klienten (HTML/JS). Det er virksomheds ansvar at placere den nødvendige forretningslogik bag API'et, idet det inddrager dyb forretningsforståelse og udførelse af `Business Intelligence` på data, hvilket er uden for projektets scope.

1.7 Metodologi

Metodologi i denne sammenhæng er en kombination af nogle metoder, som hjælper mig med at gennemføre og styre mit projekt. Det er nogle metoder, som kan anvendes i softwareudvikling med henblik på at levere et softwareprodukt. Software udvikling er en iterativ aktivitet/process. Typisk bygger man software gennem en udviklingprocess, som for det meste er iterativ, og slutresultatet fås ved at akkumulere delresultaterne. Hvert delresultat har til formål at opfylde et eller flere delkrav, men ikke hele kravspecikationen.

Dette afsnit beskriver den anvendte udviklingsmetode, og dens anvendelse. Desuden indeholder afsnittet en plan for selve projektforløbet.

1.7.1 Metodevalg

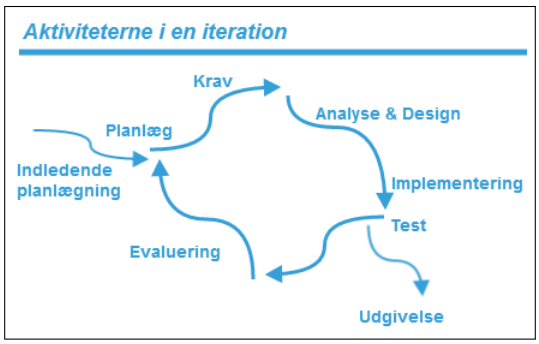
Jeg har valgt at anvende Unified Process (UP) fremgangsmåden til analyse, design og implementering af dette projekt. Jeg vil anvende den agile fremgangsmåde til UP, som beskrevet i Craig Larmans bog [4]. Bogen beskriver generelle ideer og artefakter relateret til `Objekt Orienteret Analyse/Design` (OOA/D) og kravspecifikations analyse.

UML `Unified Modeling Language` er et visuelt modelleringssprog til at specificere, konstruere og dokumentere et systems artefakter. UML er en udbredt standard for diagramnotation, og bruges til at tegne og repræsentere softwarerelaterede billeder[4]. Notationen bliver hovedsageligt brugt til at beskrive OO[9] softwaresystemer. Jeg vil benytte UML til at dokumentere og visualisere systemet under udvikling.

Iterativ og gradvis udvikling `Unified Process` er en iterativ udviklingsprocess, hvorigennem objektorienterede systemer bygges. Iterativ udvikling er

[9] Objekt Orienteret.

essensen i OOA/D. Metoden passer godt til projektet, da den lægger op til at udvikling starter, før alle kravene er defineret. Der lægges op til en gentagen proces af programmering og test af dele af systemet, og denne cyklus starter allerede tidligt i forløbet. Figur 1.4 illustrerer aktiviteterne gennem en iteration.



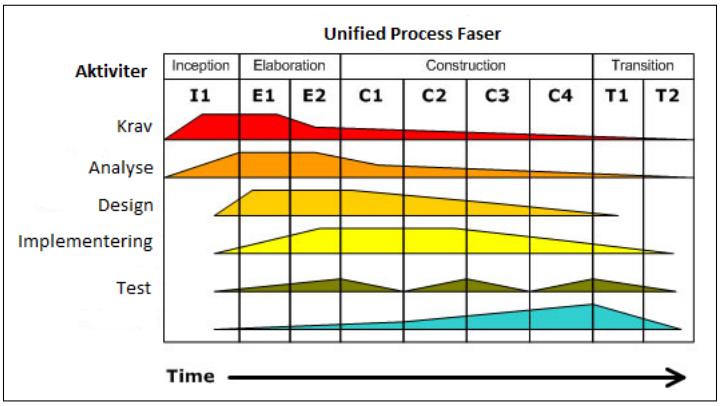
Figur 1.4: En UP iteration

Iterativ udvikling er meget centralt i UP. Projektet organiseres i en række mini-projekter kaldet **iterationer**, som hver iteration opfylder et eller flere delkrav og afsluttes med test - (gradvis udvikling). Arbejdet og iterartioner organiseres i følgende fire faser. Se i øvrigt figur 1.5.

UP faser

1. **Inception** [Forberedelse]— i denne fase etableres projektes fundament.
2. **Elaboration** [Etablering]— iterativt implementering af kernearkitektur og idetificering af de fleste højrisiko krav.
3. **Construction** [Konstruktion]— iterativt implementering af de resterende lavrisikokrav.
4. **Transition** [Overdragelse]— betatest og overdragelse.

Inception Projektet befinder sig i skrivende stund i forberedelsesfasen, hvor problemområdet og baggrund for projektet bliver defineret. Der bliver etableret en fælles vision for projektet, da det er vigtigt, at både jeg, min vejleder på DTU og virksomheden har den samme vision. Der er derfor planlagt et møde med min vejleder i slutningen af denne fase, som det også fremgår af projektplanen i figur 1.6 i næste afsnit.
I inception fasen bliver de mest forventede brugerscenarier identificeret, men kun nogle af brugerscenarierne (herefter kaldet `Use cases`) bliver beskrevet i detaljer[4, s. 33].



Figur 1.5: UP projekt organisering i faser og iterationer

Elaboration I denne UP fase, etableres og implementeres gradvist kernearkitekture for systemet, ved at inddrage dele, der har størst risiko for systemet. Størstedelen af kravene identificeres og stabiliseres. De største risici begrænses[4, s. 127].

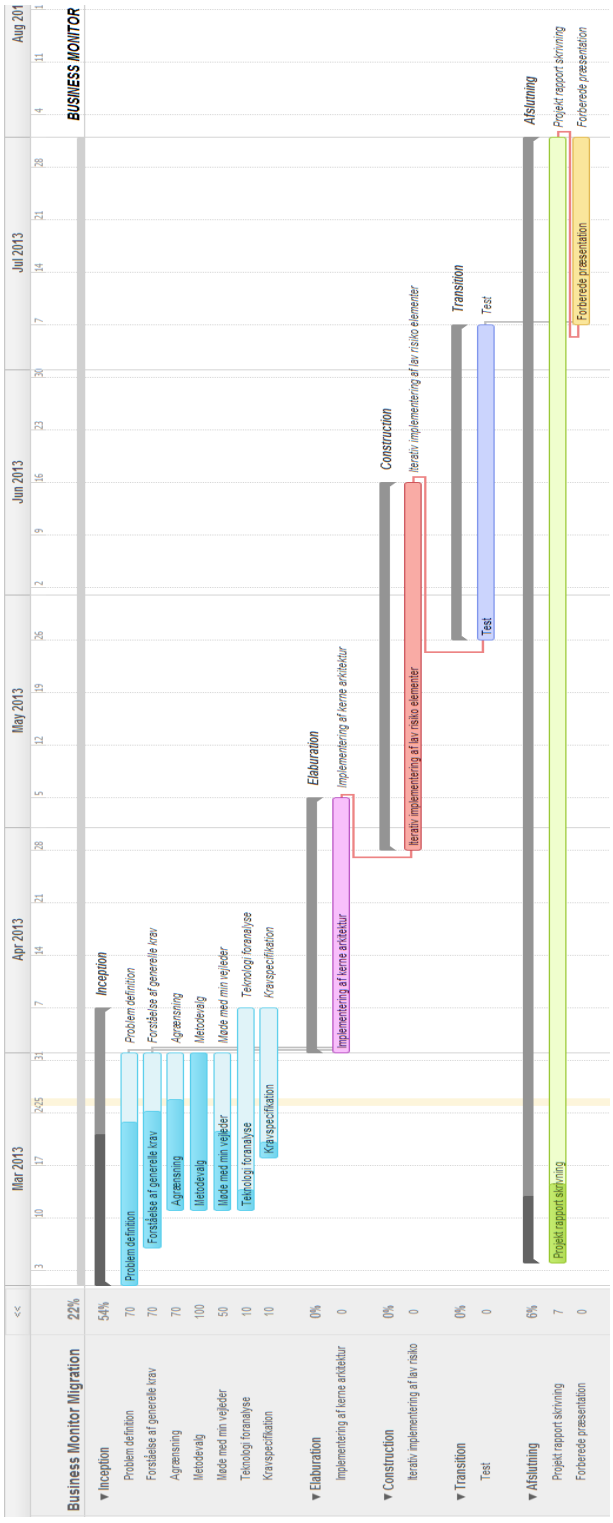
Construction Under denne fase implementeres iterativt de resterende elementer af systemet, og det gøres klart til udgivelse. Det er her de fleste iterationer ligger, som resulterer i nogle små udgivelser og tilføjer funktionalitet til det samlede system.

Transition Denne fase har til formål at overdrage systemet til slutbrugeren, og håndtering af de problemer der opstår i forbindelse med overdragelsen.

Arbejdsbyrden i mit projekt vil langsomt flyttes fra `Krav` mod `Test`, hvor indsatsen vil være størst mod `Krav` og `Analyse` i starten. Over tid vil min største indsats være rettet mod `Design, Implementering` og `Test` senere i forløbet, som vist i figur 1.5.

1.7.2 Projektplan

Projektet udføres inden for et semester med startdato den 1. marts 2013 og slutdato den 1. august 2013. Perioden svarer til 154 dage— i alt 22 uger. Jeg har lavet en tidsplan, som er vist i figur 1.6, og det er min hensigt at gennemføre projektet baseret på denne plan.



Figur 1.6: Projektplan

1.8 Rapport struktur

De i afsnit `Problem definition` opstillede problempunkter søges besvaret gennem de efterfølgende kapitler. Projektet er bygget op således, at der efter denne indledende gennemgang følger en række kapitler, der behandler de opstillede punkter under problemdefinitionen således:

Teknologi Analyse Kapitlet indeholder en foranalyse af teknologier og frameworks (rammeværker), der vil hjælpe mig med at eliminere de sorte huller, således at jeg ved, hvad der skal bruges, og hvordan de skal bindes sammen.
PUNKT (3) under `Problem definition`.

Kravspecifikation Kapitlet indeholder en kravspecifikation til prototypen. Derudover fremsættes og beskrives specifikke UP krav artefakter, hvilket inkluderer en **Use-Case Model**— Et sæt scenarier, der beskriver brugen af systemet og identificerer de funktionelle krav. Modellen indeholder ligeledes et UML use case diagram.
PUNKT (4) under `Problem definition`.

Analyse Kapitlet ser på de identificerede funktionelle krav nærmere detaljeret, og beskriver de enkelte use cases der blev udpeget i kapitlet om `Kravspecifikation`. Dette kapitel fokuserer på brugerens interaktion med systemet.
PUNKT (2, 4) under `Problem definition`.

Design Kapitlet beskriver de forskellige elementer i systemet, samt hvordan de er bundet sammen. Kapitlet fokuserer ligeledes på systemets arkitektur og præsenterer et UI design for systemet.
PUNKT (5, 6, 7, 8,10, 9) under `Problem definition`.

Implementering Kapitlet omhandler implementering af prototypen i henhold til den arkitektur, jeg er kommet frem til i kapitlet om `Design`, og den fuktionalitet der blev præsenteret i kapitlet `Analyse`, ved brug af de teknologier analyseret i kapitlet `Teknologi Analyse`.
PUNKT (5, 10) under `Problem definition`.

Test Kapitlet viser, hvordan testprocessen er blevet implementeret og verificerer, hvorvidt kravspecifikationen er opfyldt.
PUNKT (11) under `Problem definition`.

Kapitel 2

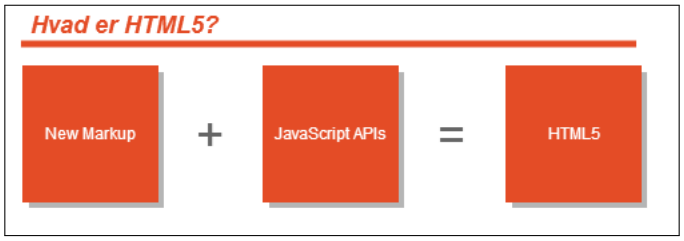
Teknologi Analyse

Jeg vil i dette kapitel lave en foranalyse af teknisk karakter, da jeg er nødt til at vide noget om de forskellige teknologier, før jeg kan komme til en realisering af projektet. Foranalysen kommer til at omhandle valg af relevante frameworks, forstå hvilke muligheder de giver mig og hvorledes jeg kan få dem bundet sammen. Jeg vil ligeledes i dette kapitel beslutte, hvilke værktøjer jeg vil bruge til udvikling af prototypen.

2.1 HTML5

HTML5 er en paraplybetegnelse. Det handler ikke kun om markup, men mere om en række teknologier. Med HTML5 er browsere nu i standt til at udføre meget mere end hvad den traditionelt har været i stand til. Normalt forbinder vi browser med et program, der kan sende forespørgsler til en webserver og får svaret tilbage, som den præsenterer for brugeren. Hvis der skal udføres noget vigtigt, sendes en ny forespørgsel og der modtages et nyt svar.
Med HTML5 er browsere blevet i standt til at byde på meget. Man har nogle moduler til rådighed til at tilgå data, mulighed for kommunikation i realtid, kommunikere over SOCKETS og en mere avanceret grænseflade. Man taler om, at med HTML5 er browsere blevet mere som et operativsystem, fordi disse er alle de dele, man kan finde i et operativsystem. Browsere er blevet en mere kompetent platform, som vi kan afvikle rige applikationer i [7, s. 1].

Så hvad er HTML5 egentlig? Hvis man skal definere, hvad HTML5 er (se figur 2.1), kan man sige, at det er tilføjelsen af nye markup og nye JavaScript APIs. Nogle af disse nye markup er en række nye elementer, der er blevet tilgængelige, men der er også tilføjet nye attributter, som er blevet tilgængelige for eksisternede elementer. Men i virkeligheden, når man taler om HTML5, er det de nye JavaScript APIs og JavaScript biblioteker, der definerer, hvad HTML5 er samt den funktionalitet, der nu er til rådighed i browser. Det overordnede håb er at en dag vil man med HTML5 få en konsistent weboplevelse på tværs



Figur 2.1: Tilføjelse af nye Markup og nye JavaScript APIs definerer HTML5.

af forskellige enheder og browsere. HTML5 er stadig under udvikling, og selvom specifikationerne ikke er 100% færdiggjort, er browsere begyndt at implementere nogle af dens features.

Bedre JavaScript Over de seneste år, er JavaScript ydeevne i de fleste moderne browser blevet meget bedre og hurtigere. Kombineret med HTML5s forbedrede ydeevne, er der ikke længere forskel på, hvor hurtig en HTML applikation er i forhold til kompileret binær kode. Dette betyder, at man ikke længere kan drage fordel ved plugin-teknologier som Silverlight og Flash [7, s. 6].

2.2 JavaScript, biblioteker og frameworks

JavaScript er det primære client-side programmeringssprog man bruger, når man skriver HTML5 applikationer. Ved hjælp af JavaScript kan man fx vælge et hvilket som helst element på en side, arbejde med data i hukommelsen eller kommunikere asynkront med serveren. Ved hjælp af framework som jQuery kan man både gøre udviklingsprocessen hurtigere og reducere mængden af den leverede kode.

Open source frameworks Udvikling i JavaScript drager fordel af de utallige Open Source projekter og gratis værktøjer, som man som udvikler har til rådighed, og som man kan bruge for at gøre udviklingsprocessen mere effektiv. Som eksempel kan jeg nævne `Knockoutjs`, som er et `JavaScript` framework baseret på udviklingsmønstret MVVM (`Model-View-ViewModel`). `Knockoutjs` giver mulighed for, på klient siden, at kode efter MVVM paradigmet, når man vil implementere en vedligeholdelsesvenlig HTML applikation. Et andet eksempel er `JavaScript` biblioteket `jQuery`, som jeg nævnte tidligere. Og der er mange flere.

Dette kapitel indeholder mine overvejelser omkring valg af teknologier og de tredjeparts JavaScript bibliotker, som jeg vil bruge i forbindelse med implementering af prototypen. De fleste af disse JavaScript biblioteker er open source, men der er også nogle kommercielle produkter. Hvad skal man vælge? Dette

er netop spørgsmålet jeg gerne vil have besvaret gennem dette kapitel og fastlægge derved den tekniske ramme for projektet. Nogle valg er ikke så svære at træffe og kræver ikke yderligere undersøgelse. Andre er præget af mine egene personlige præferencer. Fx vil jeg bruge JavaScript biblioteket jQuery, da det er en af de mest berømte frameworks for udvikling af HTML applikationer, og er understøttet af nogle af de store virksomheder. Microsoft har endda valgt at distribuere jQuery via sit udviklingsværktøj Visual Studio, som er det IDE (**Integrated Development Environment**) jeg vil bruge i forbindelse med udvikling af prototypen.

JavaScript er et Loosely Typed funktionsorienteret sprog og indeholder objektorienteret egenskaber. Der er flere måder at konstruerer objekter på. Man skal som udvikler lægger et fast mønster at udvikle efter, hvis koden skal være vedligeholdelsesvenlig. Knockoutjs simplificerer dynamisk JavaScript UI ved at anvende MVVM designmønstret. Der findes andre mønster-baserede JavaScript biblioteker, som er baseret på MVC (Model View Controller) paradigmet, men jeg vælger Knockoujs, da jeg kan drage fordel af min erfaring med MVVM, både i forbindelse med et kursus jeg har haft på DTU, men også mit arbejde under praktikperioden hos virksomheden, hvor jeg arbejdede med Silverlight, hvor vi også anvendte MVVM.

2.2.1 jQuery

Grundet øget fokus på rige brugeroplevelser på klient siden, bliver brug af JavaScript i web applikationer mere og mere vigtigt. Desværre er det en mere krævende process at arbejde med rå JavaScript. Forskellige browsere har forskellige features og begrænsninger, der kan gøre kodning i JavaScript mere komplekst. Her kommer JavaScript biblioteker ind i billedet. En af de meste berømte er **jQuery**. jQuery JavaScript biblioteket sigter mod og hjælper med, at man lettere kan arbejde med JavaScript og normalisere JavaScript funktionalitet på tværs af browsere [3, s. 5].

jQuery's hovedfokus er at hente elementer fra html sider, og udfører operationer på dem. Dette gør jQuery, mens det gør sit bedste og sørger for at koden altid virker på tværs af de forskellige browsere. jQuery løser problemer omkring et inkonsistent DOM API blandt forskellige browsere.

2.2.2 AJAX

AJAX er teknik til at udføre en **XMLHttpRequest** (et API til at sende HTTP forespørgsler direkte til en webserver.)[1] forespørgsel fra en webside til server, og sende og modtage data der skal bruges på websiden. AJAX står for `Asynchronous Javascript And XML` og bruger JavaScript til at konstruere en **XMLHttpRequest**.

[1] `http://en.wikipedia.org/wiki/XMLHttpRequest`

jQuery implementerer et interface til AJAX forespørgsler og skjuler derved også kompleksiteten forbundet med forskellige browseres understøttelse til udførelse af AJAX anmodninger.

2.2.3 Kendo UI

Der er JavaScript biblioteker til at udføre bestemte opgaver, biblioteker der agerer som et udviklings framework eller biblioteker til unit test. Kendo UI er et HTML5 og JavaScript framework baseret på jQuery. Kendo UI er et eksempel på et kommercielt JavaScript bibliotek fra Telerik. Business Monitor Dashboard har til formål at visualisere data, hvilket er noget man kan bruge Kendo UI til. Kendo UI er en pakker indeholdende 3 forskellige produkter: Kendo UI Web, Kendo UI Data Viz og Kendo UI Mobile. Det er primært Kendo UI DataViz jeg er interesseret i, da det har data visualiseringskomponenter, som jeg vil kunne bruge i mit HTML Dashboard. Der er flere virksomhedere (både nyere og ældre), der leverer HTML5 og JavaScript baserede data visualiserings komponenter. Jeg valgte Kendo UI ud fra kriterier såsom hjælpeforum, community omkring produktet, og hurtig hjælp osv. Desuden samarbejder Business Monitor i forvejen med Telerik i forbindelse med den eksisterende Silverlight applikation, hvor der anvendes Teleriks UI komponenter til Silverlight og har positive erfaringer med det.

Hvad er Kendo UI? Kendo UI er et JavaScript bibliotek til at bygge moderne interaktive web applikationer. I dag forventer man at websider er rige, interaktive og responsiv. Det opnår man ved at drage fordel af client-side teknologier og toolset. Dette er hvad Kendo UI giver os. Kendo UI består af en samling af script filer og andre ressourcer som CSS styles, billeder osv. Kendo UI leverer client-side teknologier, som anvendes til at implementere rige web applikationer, som inkluderer:

JavaScript
: JavaScript har som tidligere nævnt, gjort et stort comeback i forbindelse med den seneste tids udvikling inden for web, og giver mulighed for at bygge spændende web applikationer.

HTML5
: HTML5 er den senetes udgave af HTML specifikation, som har til formål at sætte en standard for hvordan næste generation af web applikationer skal implementeres.

CSS3
: CSS3 er den senestes version af CSS specifikation og giver yderligere funktionalitet til at style responsiv web applikationer. Specifikationerne for både HTML5 og CSS3 er stadig under udarbejdelse, men mange af funktionaliteterne er allerde understøttet af de største browsere.

jQuery

jQuery som tidligere nævnt, er en meget populære JavaScript bibliotek, som gør DOM manipulation meget nemmere ved brug af **selectors**.

2.2.3.1 Hvad får vi med Kendo UI?

Efter at have set på hvad Kendo UI er, ser jeg nu på, hvilke features Kendo UI har.

UI Widgets

Med Kendo UI får vi en stor samling af rige UI Widgets, som er nogle HTML5 controls, baseret på jQuery og er understøttet af alle nye og ældre browsere. Der er 3 katagorier af UI Widgets:

- Web
- DataViz
- Mobile

Jeg kommer primært til at bruge dens DataViz og nogle af dens Web Widgets. Web widgets inkluderer Grid med funktionaliteter som paging, sortering og filtrering. Desuden indeholder den Widgets som menuer, tekst editor, treeview og meget mere. DataViz eller data visulaliseringswidgets bruges til interaktiv data visualiseringer, og inkluderer forskellige typer af grafer.

Client-side DataSource

Kendo UI inkluderer også data source komponent på klientsiden. **DataSource** komponenten er en abstraktion for de data vi bruger, som både kan være lokal data (array af JavaScript objekter), eller remote data (XML, JSON[2]). Den simplificerer data binding og andre data operationer.

MVVM Framework

Kendo UI har også et MVVM framework, men jeg vil hellere bruge Knockout, da efter hvad jeg har læst mig frem til, så er Knockoutjs binding framework mere modent end Kendo UIs, som i skrivende stund stadig er under udarbejdelse. Jeg er spændt på at finde ud af hvorvidt Kendo UI understøtter Knockoutjs, da jeg kommer til at bruge Knockout bindings på Kendo UI widgets. Jeg har læst i en blog på Kendo UI forum, at de har lagt meget energi i at understøtte Knockoujs, men at denne undestøttelse ikke er 100%, og at i nogle tilfælde vil man vha. lidt kode kunne få bindings til at virke.

[2] JavaScript Object Notation eller JSON formattet bruges for serialisering og transmitering af strukturede data over en netværksforbindelse.

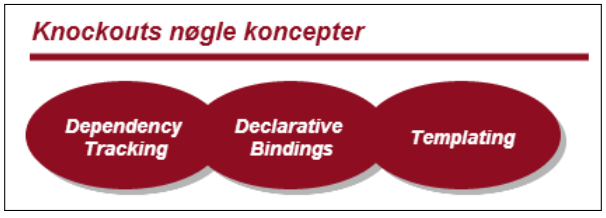
Hvorfor Kendo UI? Som jeg skrev i starten af dette afsnit, valgte jeg Kendo UI ud fra nogle kriterier. Jeg nævnte kriterier som udvalget af widgets, hjælpeforum og kontaktmuligheder. Men der er også andre aspekter der er vigtige, når man vælger at satse på en teknologi.
Der er en brede vifte af andre værktøjer på markedet. Så hvorfor vælge Kendo UI? Først og fremmest, Kendo UI giver alle de værktøjer, jeg skal bruge i en samlet pakke. Det gør at man nemmere kan opsætte udviklingsmijøet, men også at de forskellige værktøjssæt arbejder mere effektivt sammen. En anden ting er support. Kendo UI er et produkt af en meget populær leverandør, der tilbyder god support og hurtig svartider ved henvendelse.

2.2.4 Knockoutjs

Hvad kan vi bruge Knockout til? Hvad nytte gør det? Dette prøver jeg at får en afklaring på i dette afsnit. Hvad vi gerne vil opnå, er en klar adskillelse mellem de forskellige 'ting'. Adskillelsesprincippet bruges også i mange andre teknologier, men i forhold til HTML og JavaScript, menes der en adskillelse mellem strukturen, som er HTML, fra præsentationen (CSS) og fra adfærd (JavaScript). Det kan vi også godt opnå i JavaScript udvikling, og det er vigtigt, fordi vi vil gerne gør tingene mere vedligeholdelsesvenlige og nemmere at bruge.

Så hvordan håndterer vi UI mønstre i JavaScript udvikling? Hvordan kan vi strukturere vores JavaScript? Og hvad med data og bindings. Alle applikationer bruger og håndterer data. I JavaScript loader man fx data via Ajax service, og får noget JSON data tilbage. Hvordan skal dataen så indlæses i View'et? Hvordan skal ændringerne skubbes frem og tilbage mellem source (JavaScript objekt) og target (HTML elementer)?
Med Knockoutjs kan vi løser mange af disse forskellige scenarier og sætte nogle strukturer på plads.

Hvad er Knockout? Knockout giver mulighed for client-side interaktivitet ved at følge MVVM principper og give en god adskillelse mellem adfærd og struktur. Knockout bruger det man kalder **Declarative bindings**, for at binde UI elementer (HTML) eller endda også CSS klasser, til source objekter. Knockout understøtter følgende koncepter (figur 2.2):



Figur 2.2: Knockouts nøgle koncepter.

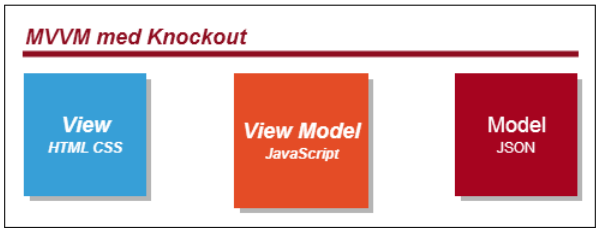
Dependency tracking
Dependency tracking betyder, at Knockout automatisk opdaterer de relevante dele af UI, når den 'underliggende' datamodel ændres. Det gælder også den anden vej, når UI værdier ændres, at ændringerne vil automatisk reflektere i source objektet. Man kan sammenligne det, i XAML tekonologier som WPF eller Silverlight, med INotifyPropertyChanged interfacet.

Declarative bindings
Med Declarative binding menes at man kan binde dele af UI elementer med sin data model på en meget simpel måde - dvs. via selve HTML elementet. Så i stedet for i JavaScript via kode at finde et element via dets id og sige at det skal bindes med et JavaScript objekt, kan man gør det via Declarative binding i selve HTML elementet.

Templating
Templating er brugbart, når man har en struktur i websiden, der gentager sig. Fx `<li>` elementer inde i `<ul>` struktur, eller rækker i et `<table>` element.

2.2.4.1 MVVM med Knockout

For at få en bedre forståelse af Knockout, er det vigtigt at forstår hvad MVVM er. MVVM er blot en adskillelsesmønster, og en måde at organisere og strukturere sin kode for at gøre det nemmere at vedligeholde. Den adskiller ansvarsområder for Model (data), View, som i dette tilfælde er HTML og ViewModel, som vi skriver i JavaScript. Så hvad betyder MVVM i forhold til JavaScript og HTML?



Figur 2.3: MVVM med Knockout.

Figur 2.3 illustrerer MVVM med Knockout. HTML og CSS repræsenterer et View—JSON data repræsenterer en Model og imellem de to er der noget logik, der angiver, hvordan vi præsenterer Model (JSON data) i View'et—denne logik er repræsenteret ved ViewModel. Så vi har View, ViewModel og Model. Model repræsenterer data. I dette tilfælde er det et JavaScript objekt, der har nogle properties som fx `name`. Viewet er HTML siden. Det er den information vi gerne vil vise på skærmen gennem forskellige HTML tags. Og så er der ViewModel, som binder de to sammen. Den indeholder al den adfærd, som View'et skal bruge, og den samler en eller flere Model objekter, der skal vises i Viewet.

2.3 Responsive Design

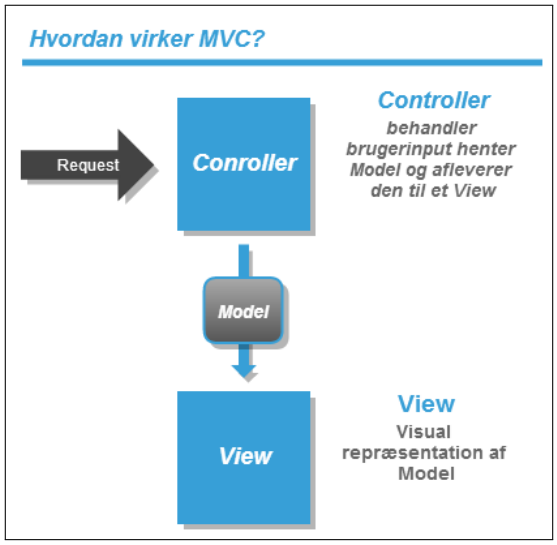
Nutidens internetbrugere benytter i mindre grad kun computere, når de browser. I løbet af den seneste tid er antallet af tablet- og smartphonebrugere ekploderet. Dette har sat krav til design og visning af websites, da disse gerne skulle vises optimalt på flere forskellige platfome. Så hvad betyder udtrykket **Responsive Design**? Når en side vises på en skærm eller et visningsområde, vil det være meget rart, hvis det ændres og tilpasses i overensstemmelse med skærmopløsningen og størrelse. Det er, hvad der menes med Responsive Design— evnen til at kunne gøre dette. Dette kan opnås med CSS3; at lave en responsiv web design, der giver en god brugeroplevelse på tværs af forskellige enheder og skærmstørrelser. Ved brug af CSS3 **media queries** er vi i stand til at udpege specifikke CSS regler til at træde i kraft ved specifikke *viewports*.

Media queries er en af CSS3s mange moduler. Vha. media queries kan vi anvende nogle bestemte CSS styles afhængig af enhedens skærmstørrelse. Vi kan fx ændre den måde indhold på en side bliver vist på— baseret på ting som såsom *viewport* brede, skærmformat, orientering osv., ved blot nogle få CSS regler.

2.4 ASP.NET MVC 4

Jeg vil implementere prototypen som en ASP.NET MVC 4 applikation. ASP.NET MVC 4 er en applikation framework fra Microsoft med fokus på vedligeholdelse af kode, adskillelse af UI og logik og testbarhed, baseret på MVC designmønstret. ASP.NET MVC 4 får sit navn fra MVC designmønstret, som er et designmønster man følger, når man vil adskille beskrivelserne af brugergrænseflade, logik og model. Dette er illustreret i figur 2.4 nedenfor.

Bogstavet C i MVC står for **Controller**. Controller er den software komponent, som tager imod indkommende HTTP forespørgsel. Når en Controller modtager forespørgslen, har den ansvaret for at bygge en **Model** - bogstavet M i MVC. Det er Model, der indeholder de nødvendige information, som vi skal præsenterer for brugeren for at opfylde den indkommende forespørgsel [8, s. 7]. Controller vælger herefter et **View** for at præsentere denne Model. View i ASP.NET MVC er nogle meget simple objekter. View objekter tager information fra Model og placerer dem, der hvor de hører til på en html side. Resultatet bliver, at man isolerer adfærd i sin grænseflade i en af de tre katagorier. Et View objekt vil aldrig få brug for at vide, hvordan det tilgår datalaget, da Model objektet alerede indeholder de data, View'et skal bruge. På samme vis vil en Controller aldrig have bruge for at vide, hvor fx en fejlbesked skal vises, eller hvad farve den skal have, idet det er View'ets ansvar.



Figur 2.4: Hvordan virker MVC?.

2.5 Udviklingsmiljø og værktøjer

2.5.1 Visual Studio 2012

Visual Studio er det IDE, hvor jeg vil skrive, debugge og teste min kode. Dette værktøj indeholder kode editor, som kan håndtere C#, og også JavaScript, HTML og CSS. Visual Studio har genveje til alle de teknologier, som vi har brug for at binde sammen i en webapplikation. Visual Studio giver en meget funktionsrig debugging og redigerings miljø til JavaScript CSS, HTML og .NET selvfølgelig.

2.5.2 Team Foundation Server tfs

TFS er et Microsoftprodukt udviklet til kollaborativ software udvikling. Jeg vil bruge TFS til versionsstyring af den kode jeg udvikler.

2.5.3 Web browser debugging værktøjer

De fleste webbrowsere har debuggingsværktøjer indbygget, som er meget robuste og velegenede til at anvende under HTML og JavaScript udvikling. Jeg kommer primært til at bruge Chromes debuggingsværktøj, fordi jeg synes, det indeholder nogle meget brugbare funktioner og er dejligt overskueligt.

2.5.4 (Azure) SQL Database

Business Monitor applikationen kører i Windows `Azure` infrstrukturer og bruger de tjenester, som Windows Azure stiller til rådighed. Det gælder også datatje-

nester. Windows Azure tilbyder SQL Databasetjenester, der stiller samme data interface til SQL-baseret database adgang til rådighed som ens egen lokale instans af SQL server. I forbindelse med udvikling af protoypen kommer jeg til at bruge den eksisterende database, men jeg vil også tilføje en database til Dashboard persistens ved brug af Entity Frameworks Code First feature.

2.5.5 IIS og Azure deployment

Som webserver til at hoste og eksekvere applikationen under udvikling, vil jeg bruge ISS (Internet Information services) lokal på min maskine. Jeg vil løbende under udvikling også deploye løsningen til Azure, hver gang jeg når en milestone, så jeg nemmere kan vise det til mine vejledere. Jeg vil selvfølgelig også deploye den endelige løsning til Azure.

2.5.6 ShareLaTeX

Til rapportskrivning benytter jeg den online LaTeX editor ShareLaTeX (`https://www.sharelatex.com/`).

2.5.7 UML værktøj - Gliffy

Til tegning af UML- og andre diagrammer, har jeg brugt Gliffy, som er et HTML5 baseret online diagramværktøj (`http://www.gliffy.com/`).

2.5.8 Mockups værktøj - Balsamiq

Jeg vil lave mockups til prototypen, der viser hvordan siden skal struktureres, inden jeg laver strukturen i HTML. Til det vil jeg bruge værktøjet Balsamiq.

Kapitel 3

Kravspecifikation

Dette kapitel specificerer de betingelser og funktioner, som prototypen og projektet i det hele taget, skal opfylde. Disse vil, ifølge UP disciplin, blive katagoriseret som **funktionelle** (beskriver systemets adfærd) og **ikke-funktionelle** (øvrige) krav. Kravene vil blive organiseret i en UP artefakt til formålet, nemlig Use-Case Model. Det er hovedsageligt de funktionelle krav, der vil blive organiseret som use cases, da de angiver, hvad systemet vil udføre— de adfærdsmæssige krav. De ikke-funktionelle krav dækker over ting som brugeroplevelse, udseende og pålidelighed.

3.1 Problemområdet

Business Monitor applikationen er udviklet i Silverlight, som er en pluginteknologi fra Microsoft. Ulempen ved brug af pluginteknologier er deres manglede tilgængelighed på tværs af forskellige platforme og enheder. Ulempen er blevet større i kraft af den seneste tids udvikling inden for mobile enheder, som mange internetbrugere i dag benytter. De fordele, man forbandt med pluginteknologier for nogle år siden, er blevet af mindre betydning i takt med den seneste tids udvikling inden for webteknologier— navnlig HTML og JavaScript. Fx var/er en af de største fordele ved pluginteknologier, at plugins giver en forudsigelig runtimemiljø inde i browseren, hvor man som udvikler ikke behøver at bekymre sig om problemer omkring inkompatibilitet på tværs af browsere. Men evolutionen i HTML og JavaScript har gjort, at dette ikke længere er et problem, takket være de mange JavaScript frameworks og værktøjssæt. Disse udstyrer udviklere med øget produktivitet og tilføjer et abstraktionslag, som normaliserer inkompatibilitet.

Business Monitor A/S ønsker på sigt at komme ud af Silverlight løsningen. Med HTML5 løsningen kan Business Monitor nå ud til mange brugere og platforme— også de mobile, da HTML5 er understøttet på alle enheder/platforme. Selve HTML teknologien er langt mere sikker end nogen af pluginteknologierne. En webside

skrevet i HTML er langt mere sandsynlig stadig at være brugbar efter et årti, end en der er skrevet i Silverlight.

3.2 Business Monitor Dashboard

Inden jeg begynder at specificere en kravmodel for prototyen, vil jeg først definere, hvad et Dashboard er, og hvordan man vil bruge det samt hvilke funktioner det skal tilbyde brugeren, på et højere abstraktionsniveau end det kravmodellen angiver i det næste afsnit.

Business Monitor ønsker et HTML Dashboard værktøj, hvormed man kan overvåge sin virksomheds økonomiske situation. Dashboardet skal kunne give et hurtigt overblik over de vigtigste finansielle nøgletal, der viser hvordan ens virksomhed præsterer. Dashboardet skal ligeledes give mulighed for at grave sig ned i de data, der ligger til grund for nøgletalsberegningen, og man skal kunne se resultater grafisk eller i tabelform.

Grafisk visualisering af en virksomheds finansielle nøgletal, er en vigtig funktion, som Business Monitor ønsker i Dashboardet, idet den kommunikerer resultater på en meget sigende og overskuelig måde. HTML Dashboardet skal endvidere give mulighed for at oprette nye dashboards, et katalog af nogle finansielle nøgletal (KPI[1] katalog), hvor man kan vælge og tilføje nøgletal på sit dashboard. Nøgletalene skal så, på baggrund af virksomhedens regenskabsdata, kunne præsenteres på dashboardet i form af en graf eller tabel. Man skal desuden have mulighed for at justere grafen, så den viser, hvad man har behov for. Fx skal man kunne vælge at se sin omsætning over tidsdimension eller afdelingsdiminsion for indeværende regnskabsår— eller vælge at se en sammenligning med forrige regnskabsår.

Disse er nogle af de funktioner, jeg gerne vil have prototypen i HTML skal kunne udføre. I næste afsnit vil jeg definere nogle funktionelle krav til prototypen.

3.3 Funktionelle krav

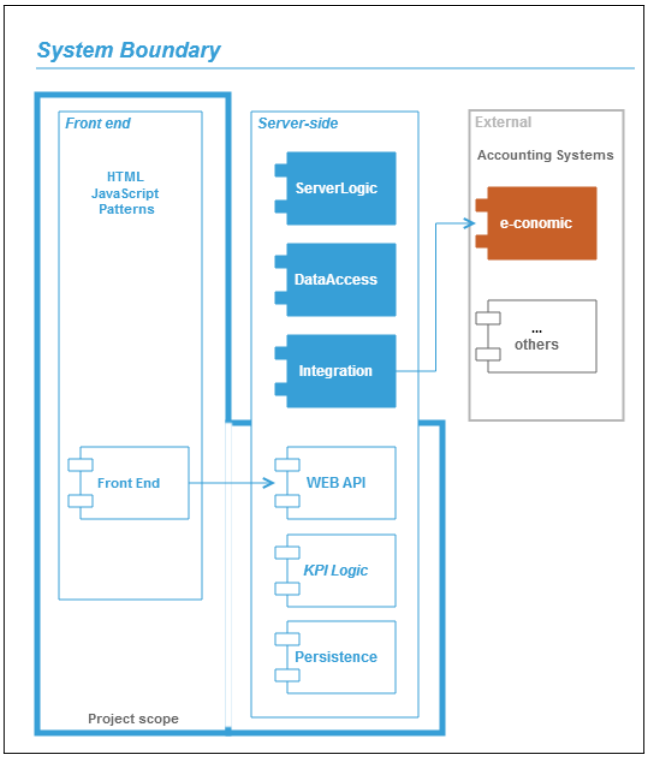
For bedre at kunne identificere de funktionelle krav systemet skal opfylde, vælger jeg at følge proceduren foreslået her [4, s. 82]. Jeg vil starte med at definere **systemgrænserne** (System boundary), så det derved kan blive fastsat, hvad der er eksternt og internt i systemet under opførelse. At fastsætte systemgrænserne her er især vigtigt for mig, da prototypen kommer til at indgår i resten af Business Monitor systemet for at kunne udføre sine funktioner. Business Monitor trækker data ud af regnskabssystemet e-conomic gennem sit integrationsmodul. Prototypen kommer til at kommunikere med Business Monitor serveren gennem

[1] KPI = *Key Performance Indicator* = Nøgletal.

et WEB API (PUNKT 9 under `Problem definition`) og bruger de data. En udspecificering af systemgrænserne vil samtidig også hjælpe mig med at forstå hvilke aktører, der er i spil i systemet.

3.3.1 Udspecificering af systemgrænserne

Som beskrevet ovenfor, vil jeg her prøve at fastsætte systemgrænserne for prototypen. Dette vil samtidig gør det nemmere at identificere de aktører, der optræder i systemet for at protoypens funktionalitet kan opfyldes. En aktør er en person, virksomhed eller et eksternt system, der spiller en rolle i form af interaktion med systemet. Figur 3.1 nedenfor illustrerer, hvad der er projektets scope. Projektets scope er markeret med indramning.



Figur 3.1: Systemgrænserne for prototypen.

3.3.2 Projektets scope

Figur 3.1 viser systemelementerne `Front end`, `Server-side` og `External`. Udviklingen af `Front end` og nogle server-side komponenter, er hvad der definerer rammen for projektet. Server-siden består af nogle moduler, hvoraf de relevante er vist på figuren (komponenterne med baggrundsfarve øverst i server

elementet). Den nederste del af serverelementet viser nogle komponenter, jeg i forbindelse med projektet skal tilføje og implementere, som beskrevet i det følgende.

WEB API
: WEB API komponenten, som det fremgår af figur 3.1, definerer grænsefladen til `Front End`. Al kommunikation med Business Monitor server vil ske via denne komponent. Dette giver en klar adskillelse af klient og server, og gør tingene mere vedligeholdelsesvenlige.

KPI Logic
: Dataanalyse og beregning af KPI er i den nuværende løsning, noget der sker i Silverlight klienten. Dvs. når en bruger logger på, er denne logik en del af den klientkode, som bliver hentet ned på brugerens maskine.
Jeg er derfor nødt til at implementere en server komponent, der indeholder KPI beregningslogik for nogle udvalgte KPI'er, således at prototypens funktionalitet kan opfyldes.

Persistence
: Det ønskes, at prototypen har funktionalitet, der sikrer, at ændringerne bliver persisteret, således at en bruger, der besøger sit Dashboard, kan vende tilbage og opleve sit Dashboard, som brugeren efterlod det sidst. Serveren skal derfor have tilføjet denne mulighed.

Figur 3.1 illustrerer yderst til højre et systemelement (`External`), som viser Business Monitors forbindelse til økonomisystemet e-conomic. Business Monitor `Integration` har pt. kun grænseflade til e-conomic, men den kunne potentielt også have til andre økonomisystemer, som det fremgår af figuren.

3.3.3 Identifikation af use cases

Dette bringer mig frem til at kunne identificere følgende primære use cases for prototypen. Tabel 3.1 viser de identificerede use cases for prototypen. Detaljegraden er holdt på et relativt overordnet niveau på nuværende tidspunkt. I videre analyse af kravene vil nogle af use cases blive splittet op i en række sub use cases og vise flere detaljer. Aktøren er en person, der repræsenterer en virksomhed. Da det typisk er en virksomheds ledelse, der har interesse i- og adgang til informationer om hvordan virksomheden præsterer, vil aktøren være en person med en ledelsesstilling i virksomheden.

3.3.3.1 Use case oversigt

Følgende afsnit opstiller alle de overordnede use cases for systemet, der er blevet identificeret i forbindelse med kravanalysen. Use cases opstillet i tabel 3.1 er angivet i kortfattet udtryksform, hvilket er i overensstemmelse med, hvad LARMAN[4, s. 66] foreslår, at man bruger tidlig i kravanalysen for at få en hurtig fornemmelse systemets scope.

Use case oversigt	
Use case navn	**Beskrivelse**
Administrer Dashboard	Dashboards er visualisering af de financielle nøgletal. En bruger skal have mulighed for at sætte det op for at monitorere data. En bruger kan have flere Dashboards. Hvert Dashboard med en række visningsområder, der viser nøgletallene i enten graf- eller tabelform.
Administrer KPI	KPI står for `Key Performance Indicator` og bruges fx til at vurdere en virksomheds økonomiske formåen. En KPI er et stykke datavisualisering med en række indstillinger. En bruger skal have mulighed for, på en visual og klar måde, at vælge og administere KPI'er på dashboardet.
Konfigurer KPI	KPI konfiguration dækker over visual dataanalyse. En KPI kan have en række indstillinger, som resulerer i en ændret visualisering af data. En bruger skal have mulighed for at lave KPI analyse på en visual og klar måde.
Persister KPI	Det skal være muligt for en bruger at kunne gemme sine dataanalyse, således at brugeren kan vende tilbage på et senere tidspunkt og se sine datavisualiseringer med samme indstillinger, som de blev indstillet med sidst.
Autentificer Bruger	Systemet skal kunne autentificere en bruger, der prøver at bruge systemet. Kun bruger med en Business Monitor aftale kan blive autentificeret og bruge systemet.

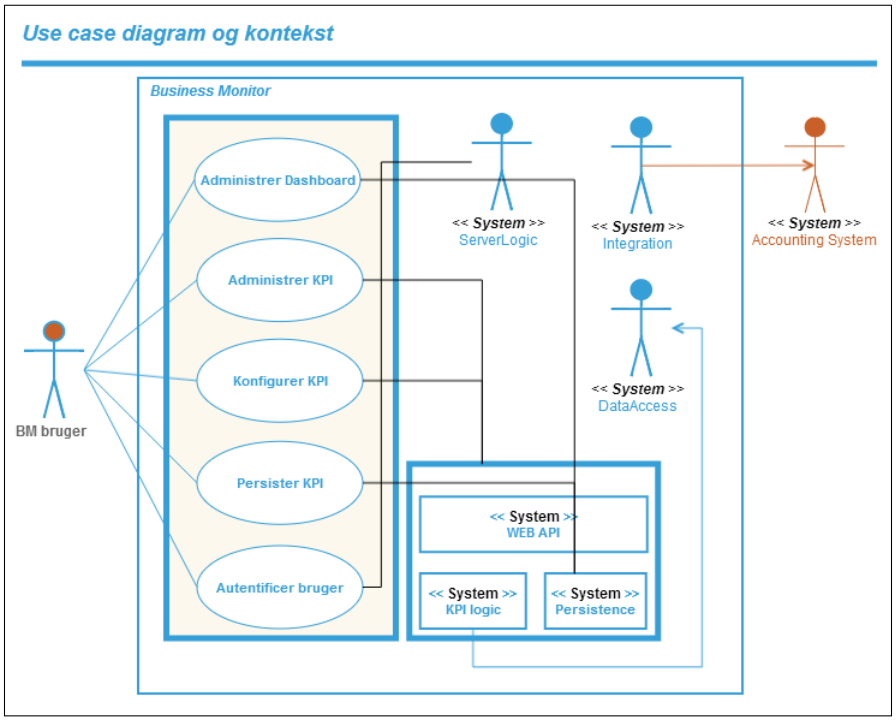
Tabel 3.1: Use case oversigt - identifikation af use cases

Det giver anledning til følgende use case diagram vist i figur 3.2. Dette use case diagram illustrerer samtidig et billede af systemets kontekst. Det viser, hvad der er uden for systemet, og hvordan systemet bliver brugt. Det fremgår ligeledes af diagrammet, hvordan systemet interagerer med sine aktører, (både den primære aktør og andre sekundære), i forbindelse med udførelse af use cases. Derved er diagrammet et godt værktøj til at opsummere systemets adfærd og aktører. Dette er i overensstemmelse med, hvad Craig LARMAN beskriver i bogen [4, s. 90] om anvendelse af UML use case diagram.

Nogle af use cases vist i diagrammet kan deles op i flere use cases. Dette vil ske i `Analyse` kapitlet, hvor jeg beskriver de identificerede use cases i nærmere detaljer. Det drejer sig om følgende use cases:

- **Administrer Dashboard**
- **Administrer KPI**

Som vi kan se på figur 3.2, forbindes use casen Administrer Dashboard med server side komponenterne. Og det samme gælder de andre use cases vist i use case diagrammet. Forbindelsen angiver kommunikation med server side komponenterne i forbindelse med udførelse af den pågældende use case. Al kommunikation med server siden sker via WEB API komponenten, så alle use casene bruger/kommunikerer med den. Dette er i overensstemmelse med PUNKT (9) under `Problem definition`. Derudover bruger nogle af use casene KPI logic- og andre komponenten Persistence i forbindelse med udførelse.



Figur 3.2: Use case diagrammet med kontekst.

3.3.4 Prioritering af use cases

For at bestemme omfang og prioritet af de use cases, som jeg vil arbejde med i løbet af den 1. iteration, laver jeg en analyse af de fundne use cases for at bestemme, hvilke use cases der er arkitektonisk vigtige for systemet. Disse use cases vil have høj prioritet og vil blive implementeret i løbet af Elaboration fasen. Af tabel 3.2 nedenfor fremgår en prioritering af de fundne use cases. Prioritering er sket på baggrund af use casenes vitalitet for systemets funktionalitet. De use cases der er mest kritiske og dækker prototypens vitale funktioner, er prioriteret højere end andre mindre kritiske use cases. Prioriteringen af use cases sker på en skala fra `høj` til `lav`, hvor `høj` angiver den højeste prioritet og vitalitet. `mellem` angiver en grad af vigtighed, der dækker en del af prototypens nødvendige funktionalitet. Og `lav` angiver vigtige funktionalitet af prototypen, som er af mindre kritiske karakterer.

Prioritering af use cases			
ID	**Navn**	**Prioritet**	**Kommentar**
UC1	Administrer Dashboard	`høj`	Denne use case tildeler jeg en `høj` prioritering, da den indikerer et væsentligt brug af systemet og er derved en arkitektonisk vigtig use case. Denne use case kan, som tidligere nævnt, splittes op i nogle sub use cases, som vi vil se i videre analyse af kravene i næste kapitel. Der vil vil være nogle use cases blandt disse sub use cases, som muligvis vil have en lavere prioritering.
UC2	Administrer KPI	`høj`	Denne use case beskriver ligeledes et vigtigt brug af systemet og får af samme grund en `høj` prioritering. Implementering af denne use case og UC1, som begge består af en række sub use cases, vil tilsammen kræve, at der tages nogle vigtige beslutninger, omkring hvordan tingene skal repræsenteres på klienten (brug af klientside teknologier), og hvad der kræves på serversiden for at understøtte udførslen.
UC3	Konfigurer KPI	`mellem`	*Konfigurer KPI* markerer jeg som en `mellem` prioriteret use case, da de use cases som den dækker over, angiver nogle vigtige brugsscenarier af systemet, men er ikke arkitektur bestemmende for systemet. Den består af en gruppe af use cases, der har med en KPIs indstillinger at gøre. Disse indstillinger vil ændre i dataets granularitet, der ligger til grund for nøgletallet, og resultet vil blive afspejlet i dens graf/tabel. Jeg vil senere i analysen tage stilling til, hvilke konfigurations indstillinger jeg vil have med.
UC4	Persister KPI	`lav`	Denne use case får en `lav` prioritering, da denne ikke har signifikans i forhold til systemets arkitektur. Dog angiver use casen et vigtigt brug af systemet, idet man gerne skulle kunne vende tilbage og finde sine ting i samme tilstand, som ved sidste anvendelse.
UC5	Autentificer Bruger	`mellem`	I Business Monitor er der oprettet nogle aftaler for demo- e-conomic kunder (som har nogle demodata hos e-conomic). De blev brugt under Silverlight udvikling. Til udvikling af prototypen skal jeg bruge en af disse demobruger. Derfor er det nærliggende at implementere noget brugerautentifikation for prototypen.

Tabel 3.2: Prioritering af use cases

3.4 Ikke-Funktionelle krav

Som nævnt i starten af kapitlet dækker de ikke-funktionelle krav over krav, der ikke bliver organiseret i use cases.

3.4.1 Funktionalitet

Sikkerhed For at kunne bruge systemet påkræves brugerautentificering. Desuden er det påkrævet, at al overførsel af data mellem webbrowser og Business Monitor server anvender SSL kryptering. Dette er med til at beskytte brugernes virksomhedsdata.

3.4.2 Brugbarhed

Det ønskes, at systemet er let, overskueligt og intuitivt. Systemet skal kunne visualisere nøgletallene på en nem og brugervenlig måde. Graferne skal være store, og ikke kun være pæne, men også kunne kommunikere indhold og have fokus på det. Systemet skal informere brugeren om opståede fejl.

3.4.3 Pålidelighed

Beregningen af de financielle nøgletal, der ligger til grund for datavisualisering, skal give et retvisende billede af virksomheds præstation.

3.5 Iterationsplan

Herunder er præsenteret en iterationsplan for systemet, hvorigennem jeg forventer at opføre systemet.

Iterationsplan		
Iteration	**Beskrivelse**	**Uger**
1	Der bestemmes en overordnet client-server arkitektur for prototypen. Der vil desuden arbejdes på de tredjeparts client side bibliotker (primært Kendo UI), i forbindelse med realisering af et meget basisscenarium af en af *Administrer KPI* sub use cases, som vil være med til at afsløre de udfordringer, der måtte være i forbindelse med bruge af tredjeparts biblioteker.	14-16
2	Jeg vil komme med et foreslag til, hvordan dashboard brugergrænsefladen skal se ud - både udseende og struktur. Dvs hvordan en brugers dashboards skal kunne håndteres og struktureres, og hvordan graferne og deres indstillinger skal vises.	17-18
3	Den arkitektoniske ramme for prototypen vil på det tidspunkt være fastsat. En client-server miljø vil allerede være sat op. Fokus vil derfor være rettet mod implementering af de use cases, der har prioritetet `høj`.	19-22
4	Implementering af use cases med prioritet `mellem`.	23-25
5	Implementering af use cases med priotetet `lav`.	26-27

Tabel 3.3: Iterationsplan for systemet under opførsel

Kapitel 4

Analyse

Dashboardapplikationen, der skal udvikles igennem dette projekt, involverer en række af spændende webteknologier, der skal arbejde sammen om at opfylde systemets funktionelle krav. Inden jeg beskriver de identificerede use cases i detaljer, vil jeg præsentere en overordnet arkitektur for systemet og definere dets datalag, webservice lag og præsentationslag. Jeg vil desuden angive, hvilke teknologier jeg vil anvende og relatere dem til de forskellige lag i den foreslåede arkitektur.

De i kapitlet `Kravspecifikation` identificerede use cases definerer den måde en bruger interagerer med systemet på. Denne interaktion har mange af de egenskaber, som man populært kalder SPA (**Single Page App**). Jeg vil starte med at definere, hvad SPA er, og hvordan prototypen relaterer sig til SPA egenskaber.

4.1 Hvad er spa?

SPA (Single Page Application) er en webapplikation, hvor alle nødvendige ressourcer (HTML, JavaScript og CSS) bliver indlæst første gang siden forespørges. Andre dele bliver indlæst dynamisk efter behov, som svar på brugerinteraktion med siden. Interaktion med siden involverer dynamisk og asynkron kommunikation med webserveren[1].

Brugeroplevelse er vigtig i en applikation, og drejer sig ikke kun om, hvordan den ser ud, men mere om hvordan den føles og opleves af brugeren. Brugeroplevelse er et stort emne, og noget der bliver forsket meget i. Den dækker over faktorer som pålidelig, responsiv, mobilitet og mange flere. Overordnet set er disse faktorer en indikator for, at applikationen virker efter hensigten - at brugeren kan stole på, at applikationen svarer hurtigt, og dens evne til at kunne bruges på forskellige enheder, som bliver mere og mere vigtig disse dage. SPA fokuserer

[1] `http://en.wikipedia.org/wiki/Single-page_application`

på disse egenskaber og har til formål at levere en lige så rig brugeroplevelse som en desktop applikation, men ved brug af webteknologierne HTML5, CSS og JavaScript.

Det, der kendetegner SPA er, at man navigerer rundt på siden ved dynamisk at skifte/opdatere dele af siden, så det forekommer, at applikationen navigerer frem og tilbage mellem forskellige sider, men siden bliver ikke genindlæst på noget tidspunkt. Indholdet af forskellige dele af siden bliver skiftet ud og opdateret via asynkrone server interaktion. Det medfører meget applikationslogik på klientsiden, som man nu skal håndtere ved brug af følgende:

Asynkrone tjenester
Man har brug for asynkrone tjenester, for nu skal der foretages mange kald fra klienten ved bruge af jQuery og AJAX.

Klientside logik
Der er meget logik, der nu skal håndteres på klienten ved brug af JavaScript, så designmønstre på klienten bliver vigtigere.

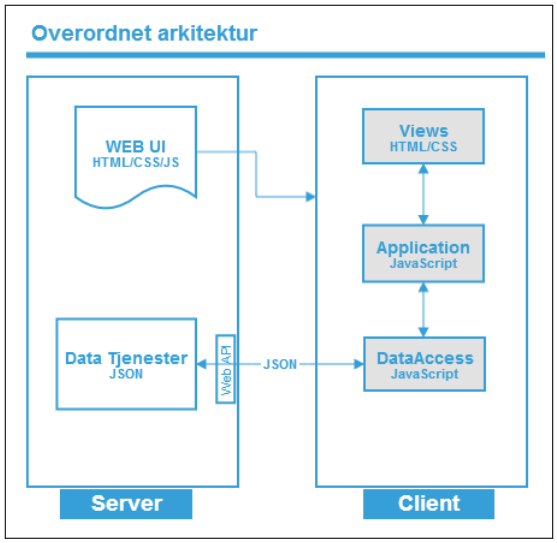
Tilstand på klient
Browser er jo stateless, så det er vigtigt at finde ud af, hvilke tilstandsinformationer der skal gemmes, og hvordan de skal gemmes på klienten.

Data binding
Data binding er meget brugbar i den slags applikationer - fx ved hjælp af (MVVM).

HTML dashboardet vil, som beskrevet i det forrige kapitel, kunne vise en brugers dashboards, når brugeren besøger siden. Brugeren vil herefter kunne interagere med programmet, og bruge dets funktioner. Brugeren vil fx kunne ændre indstillingerne på sine nøgletal, repræsenteret i dashboardet, eller vælge at tilføje nye. Enhver interaktion med systemet vil resultere i asynkron kommunikation med serveren gennem jQuery's Ajax metoder. En bruger vil under sin interaktion med systemet, i forbindelse med udførelse af en af de identificerede use cases, ikke behøve at forespørge serveren om en navigering til en anden side og derved foretage en full-page-refresh. Sagt med andre ord, al kommunikation med serveren vil ske i baggrunden, vha. asynkron JavaScript kald og opdatere dele af HTML dashboardet. Man kan derfor sige, at HTML dashboardet, der udvikles i løbet af projektet, er en SPA.

4.2 Overordnet arkitektur

Jeg vil i dette afsnit se på arkitekturen for systemet på et overordnet plan. Der er brug for en server, der kan servere websiden til klienten. Dvs. sender alle de HTML, JavaScript og CSS filer jeg måtte have i applikationen. Der vil være et WEB API, der eksponerer nogle data tjenester i JSON format.



Figur 4.1: Overordnet arkitektur for systemet.

Dette forhold er afbilledet i figur 4.1. Som man kan se, er det et client-server miljø bestående af en server, som vil servere systemets HTML, JavaScript og CSS filer. På klientsiden kan vi se, at HTML og CSS filerne konstruerer brugergrænsefladen/View for systemet. Applikation elementet på klientsiden angiver adfærd for brugergrænsefladen. Når brugergrænsefladen er indlæst, skal den bruge noget data. Data hentes via DataAccess elementet ved at kalde WEB API, som returnerer JSON data. Dette er den overordnede arkitektur, som jeg vil bestræbe mig på designe systemet efter.

4.3 Klientside teknologier

I dette afsnit vil jeg opsummere de klientside teknologier, jeg vil bruge i forbindelse med udvikling af prototypen.

4.3.1 HTML og CSS

Prototypen vil drage fordel af HTML5s mange features, som fx de nye tags der angiver semantik i HTML5, men også ting som `Placeholders` og validerings attribut. Prototypen kommer desuden til at bruge HTML5s mange andre specifikationer i kraft af de tredjeparts biblioteker, jeg kommer til at benytte fx Kendo UI. Derudover, kan vi på klientsiden drager fordel af ting som Media Queries der vil sørger for at forskellige CSS reglset bliver anvendt for at style/præsentere HTML siden, tilpasset afhængig af skærmstørrelse eller typen af enheden. I Design kapitlet vil der være et afsnit, der behandler emnet Responsive Design, hvor denne teknik vil blive gennemgået. Jeg vil dog nævne, at implementering af

prototypens funktionalitet kommer i første række. Jeg vil anvende Media Queries teknikken til at gøre prototypen responsiv, når alle de funktionelle krav er implementeret, og hvis der er tid til overs.

4.3.2 JavaScript Patterns

JavaScript kode kan skrives uden, at man behøver at følge en bestemt format for at strukturere sin kode. Det gør JavaScript nemt at arbejde med, men kan føre til spaghettikode. Spaghettikode introducerer problemer som fx global variables og navnekonflikter, som bliver uhåndterlige i stor mængde kode. Som nævnt tidligere inkluderes nu om dage mere og mere JavaScript kode i HTML5 applikationer, som fører til behovet for at strukturere JavaScript kode. Her kommer JavaScript patterns ind i billedet. Der findes flere JavaScript patterns for at strukturere JavaScript kode, gøre koden vedligeholdelsesvenlige og undgå navnekonflikter. Formålet med disse mønstre er at modularisere (indkapsle JavaScript funktioner og variable) JavaScript kode. I næste kapitel (Design) vil jeg gennemgå dette emne i nærmere detaljer.

MVVM er et andet eksempel på et designmønster, der kan anvendes på JavaScript kode. Dette handler ikke om strukturering af selve JavaScript koden, men om adskillelse af de forskellige elementer i brugergrænsefladen. MVVM kan bruges til at adskille Model, som er vores data og View, som er HTML brugergrænsefladen, fra ViewModel, som er JavaScript. Jeg vil bruge Knockout JavaScript biblioteket til at anvende MVVM i prototypens præsentationslag.

4.3.3 Biblioteker

jQuery er en stor hjælp i forbindelse med udførelse af AJAX anmodninger og DOM[2] interaktion på en måde, der skjuler inkompatibilitet blandt forskellige browser API, og derved forbedre produktiviteten. Jeg vil i høj grad gøre bruge af jQuery (især dens AJAX funktioner) under implementering af prototypen.

Knockoutjs vil hjælpe mig med at anvende MVVM mønstret. Den gør det muligt at binde HTML elementer med JavaScript kode vha. Knockout `Observables`.

toast.js er en lille JavaScript bibliotek, der kan bruges til at printe beskeder på grænsefladen. Jeg vil bruge den i forbindelse med AJAX anmodninger og informere brugeren om ting, der sker i baggrunden. F.eks. at give brugeren besked om, at en anmodning var vellykket, eller at den fejlede.

QUnit er en JavaScript unit test framework[3], som jeg vil bruge i forbindelse med test af prototypens funktioner.

[2] DOM står for `Document Object Model`, og er en oversigt over hvilke objekter man kan tilgå i HTML-dokumenter. JavaScript/jQuery bruger det til at tilgå forskellige HTML objekter.

[3] http://qunitjs.com/

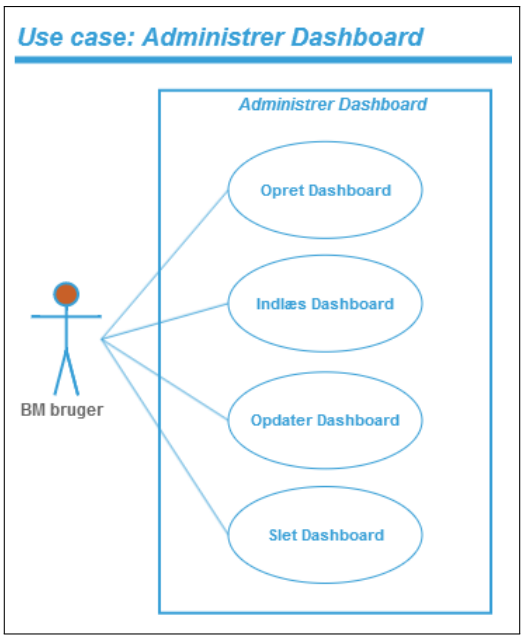
4.4 Use Cases i detaljer

Dette afsnit beskriver de i kapitel `Kravspecifikation`, afsnit `Funktionelle krav` fundne overordnede use cases til prototypen i nærmere detaljer. I det følgende vil vi se, at nogle af de identificerede use case bliver splittet op i en række sub use cases og yderligere detaljeret.

I use case beskrivelserne er begreberne Bruger og System brugt. Bruger er en aktør, der interagerer med systemet, og System er den måde dashboard løsningen skal agere og reagere.

4.4.1 UC1 "Administrer Dashboard "

Dashboards er visualisering af en virksomheds financielle nøgletal. En bruger vil kunne sætte det op med formålet at kunne monitorere data, der ligger til grund for virksomhedens financielle nøgletal. Use casen *Administrer Dashboard* dækker over følgende sub use cases vist i use case diagrammet i figur 4.2.



Figur 4.2: UC1 - Administrer Dashboards.

4.4.1.1 Opret Dashboard

Navn	Opret Dashboard
Prioritet	`høj`
Beskrivelse	Brugeren skal have mulighed for oprette og navngive et Dashboard, hvor der kan organiseres en samling af relaterede nøgletal.
Udløser [Triggers]	Systemet persisterer ændringerne.
Indgangsbetingelser	Brugeren er autentificeret
Udgangstilstand	Et Dashbaord er oprettet og klar til at tilføje indhold
Normal flow	1. Brugeren indikerer at ville se sine Dashboards. 2. Systemet viser en liste af Dashboards. 3. Brugeren indikerer at ville oprette et Dashboard. 4. Brugeren bliver bedt om at indtaste et navn. 5. Brugeren indtaster et navn. 6. Brugeren indikerer opret. 7. Systemet opretter og gemmer Dashboardet. 8. Systemet instantierer funktioner for opdatering og sletning af Dashboardet, samt tilføjelse af KPI indhold.
Alternativt flow	5a. Brugeren vælger at afbryde. 1. Brugeren indikerer at ville afbryde opretelse af nyt Dashboard. 2. Systemet afbryder processen. 3. Systemet vender tilbage til tilstanden før use case startede.

Tabel 4.1: UC1 - Opret Dashboard

4.4.1.2 Indlæs Dashboard

Navn	Indlæs Dashboard
Prioritet	`høj`
Beskrivelse	Brugeren skal have mulighed for vælge et Dashboard mellem sine oprettede Dashboards, og få vist dets indhold.
Udløser [Triggers]	Systemet initialiserer indhold.
Indgangsbetingelser	• Brugeren er autentificeret • Et Dashboard eksisterer
Udgangstilstand	Et Dashboard er vist
Normal flow	1. Brugeren indikerer at ville se sine Dashboards. 2. Systemet viser en liste af Dashboards. 3. Brugeren vælger et Dashboard. 4. Systemet viser Dashboardets detaljer /indhold.
Alternativt flow	

Tabel 4.2: UC1 - Indlæs Dashboard

4.4.1.3 Opdater Dashboard

Navn	Opdater Dashboard
Prioritet	`mellem`
Beskrivelse	Opdatering af et Dashboard er en mulighed for brugeren at kunne persistere sine ændringer foretaget på Dashboard detaljerne. Dvs. at systemet er i stand til kunne gemme ændringerne med henblik på senere at kunne genskabe Dashboardet i denne tilstand.
Udløser [Triggers]	[*Jeg vil muligvis implementerer nogle af opdate funktionerne som auto-save. I hvilket tilfælde vil enhver opdatering udløse et* `save` *event, som vil persistere opdateringen øjeblikkeligt.*]
Indgangsbetingelser	• Brugeren er autentificeret • Et Dashboard eksisterer
Udgangstilstand	Et Dashbaord er opdateret og persisteret
Normal flow	1. Brugeren indikerer at ville se sine Dashboards. 2. Systemet viser en liste af Dashboards. 3. Brugeren vælger Dashboardet der skal opdaters. 4. Systemet viser Dashboardet og dets detaljer. 5. Brugeren foretager de ønskede ændringer i Dashboardets indhold. 6. Brugeren indikerer at ville gemme ændringerne. 7. Systemet persistere det opdaterede Dashboard.

Fortsætter på næste side

Tabel 4.3 – *Fortsættelse fra forrige side*

Alternativt flow	

Tabel 4.3: UC1 - Opdater Dashboard

4.4.1.4 Slet Dashboard

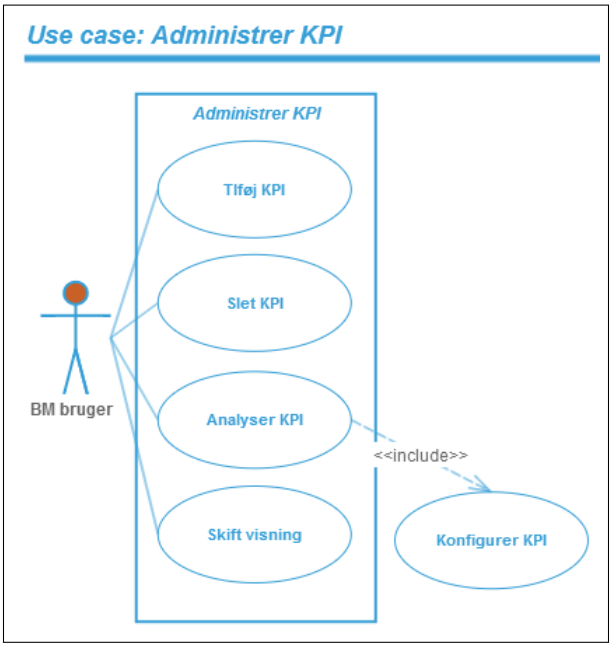
Navn	Slet Dashboard
Prioritet	`lav`
Beskrivelse	Brugeren skal have mulighed for at slette et Dashboard og dermed alt dets indhold fra systemet.
Udløser [Triggers]	Systemet persisterer sletningen.
Indgangsbetingelser	• Brugeren er autentificeret • Et Dashboard eksisterer
Udgangstilstand	Et Dashbaord og dets indhold er blevet slettet fra systmet.
Normal flow	1. Brugeren indikerer at ville se sine Dashboards. 2. Systemet viser en liste af Dashboards. 3. Brugeren vælger et Dashboard. 4. Systemet viser Dashboardet og dets detaljer. 5. Brugeren indikerer at ville slette. 6. Brugeren bekræfter handlingen. 7. Systemet persistere sletningen.
Alternativt flow	5a. Brugeren vælger at afbryder. 1. Brugeren indikerer at ville afbryde sletning af Dashboardet. 2. Systemet afbryder processen. 3. Systemet viser Dashboardet og dets detaljer.

Tabel 4.4: UC1 - Slet Dashboard

4.4.2 UC2 "Administrer KPI "

Hver datavisualisering er på et Dashboard repræsenteret ved et KPI (Key Performance Indicator). En bruger skal have mulighed for at kunne vælge et KPI og tilføje det til et Dashboard. Brugeren skal ligeledes kunne udføre KPI analyse, sletning og andre relevante operationer på et KPI. Figur 4.3 viser et use case diagram, der illustrerer de use cases, som use casen *Administrer KPI* er delt op i. Figuren viser også at use casen *Analyser KPI* har en *include* relation til use case *Konfigurer KPI* (beskrevet i næste afsnit). Dette forhold betyder, at udførelse af *Analyser KPI* use casen inkluderer udførelse *Konfigurer KPI* use case. *Konfigurer KPI* dækker over funktionalitet, der relaterer sig til en KPIs analyse indstillinger, som vil blive dækket i næste afsnit. Jeg har valgt at udar-

bejde et Systemsekvensdiagram (SSD) (figur 4.4) for *Tilføj KPI* use casen. SSD illustrerer input- og output events til systemet under opførelse. Det beskriver use casens normal flow og interaktion med systemet [4, s. 175].

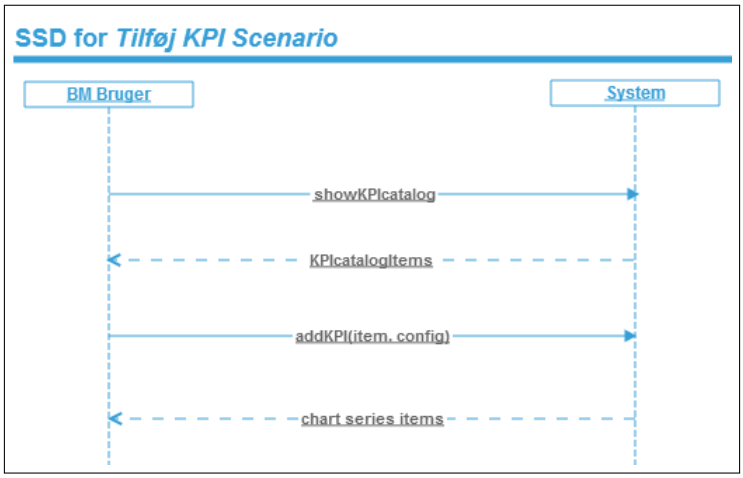


Figur 4.3: UC2 - Administrer KPI.

4.4.2.1 Tilføj KPI

Navn	Tilføj KPI
Prioritet	`høj`
Beskrivelse	Det skal være muligt for en bruger at tilføje et KPI til Dashboard. KPI'et skal kunne vælges i et nøgletalskatalog (`Kpi Catalog`), hvor de KPI'er, der findes i systemet er dokumenteret. Det skal være muligt for brugeren at vælge om det nye KPI skal repræsenteres som et detalje-Kpi eller et overbliks Kpi. Og ligeledes skal det være muligt i systemet at kunne repræsentere og visualisere et KPI som enten et detalje-Kpi eller overbliks Kpi.
Udløser [Triggers]	Dashboard instans funktioner visualiserer det tilføjede KPI med standard indstillinger. Handlingen bliver persisteret.
Indgangsbetingelser	• Brugeren er autentificeret • Et Dashboard er valgt, hvor KPI'et ønskes tilføjet
Udgangstilstand	Et KPI er tilføjet og visualiseret i et Dashboard
Normal flow	1. Brugeren indikerer at ville se KPI Katalog 2. Brugeren indikerer at ville tilføje et KPI 3. Brugeren vælger et KPI i kataloget 4. Brugeren vælger en repræsentation af KPI'et 5. Brugeren indikerer opret 6. Systemet opretter og gemmer KPI'et. 7. systemet initialiserer og visualiserer det tilføjede KPI 8. Systemet instantierer funktioner for opdatering og sletning samt konfiguration af KPI'et
Alternativt flow	5a. Brugeren vælger at afbryde. 1. Brugeren indikerer at ville afbryde tilføjelse af KPI. 2. Systemet afbryder processen. 3. Systemet vender tilbage til tilstanden før use casen startede.

Tabel 4.5: UC2 - Tilføj KPI



Figur 4.4: SSD for Tilføj KPI scenario.

4.4.2.2 Slet KPI

Navn	Slet KPI
Prioritet	`lav`
Beskrivelse	Et tidligere tilføjet KPI skal kunne slettes af brugeren.
Udløser [Triggers]	Handlingen bliver persisteret.
Indgangsbetingelser	• Brugeren er autentificeret • KPI'et eksisterer (er repræsenter et på et Dashboard)
Udgangstilstand	Et KPI er blevet slettet
Normal flow	1. Brugeren indikerer at ville se et Dashboard og dets indhold af KPI'er 2. Systemet viser Dashboardet og dets indhold af KPI'er 3. Brugeren vælger et KPI 4. Systemet viser detaljer for KPI'et 5. Brugeren indikerer slet 6. Systemet sletter KPI'et. 7. systemet persisterer sletning
Alternativt flow	5a. Brugeren vælger at annullere. 1. Brugeren indikerer at ville annullere sletningen af KPI. 2. Systemet afbryder processen. 3. Systemet viser Dashoard og dets indhold af KPI'er

Fortsætter på næste side

Tabel 4.6 – *Fortsættelse fra forrige side*

Tabel 4.6: UC2 - Slet KPI

4.4.2.3 Analyser KPI

Navn	Analyser KPI
Prioritet	`mellem`
Beskrivelse	Analyser KPI indebærer udførelse af KPI analyse funktioner repræsenteret ved *Konfigurer KPI* use case (se næste afsnit). I en af trinnene under normal flowet inkluderer jeg udførelse af *Konfigurer KPI* use casen markeret med understregning.
Udløser [Triggers]	Handlingen bliver persisteret.
Indgangsbetingelser	• Brugeren er autentificeret • KPI'et eksisterer (er repræsenter et på et Dashboard)
Udgangstilstand	Et KPI er blevet opdateret/indstillet
Normal flow	1. Brugeren indikerer at ville se KPI'er repræsenteret på et Dashboard 2. Systemet viser Dashboardet og dets indhold af KPI'er 3. Brugeren vælger et KPI 4. Systemet viser detaljer for KPI'et 5. Brugeren indikerer at ville foretage en analyse 6. Systemet præsenterer brugeren for konfigurations muligheder 7. Brugeren udfører: *KonfigurerKPI* 8. Brugeren indikerer at ville persister 9. Systemet persisterer KPI indstillingerne
Alternativt flow	8a. Brugeren indikerer ikke at ville persistere de ændrede indstillinger . 1. Brugeren undlader at indikere persister KPI indstillingerne. 2. Systemet viser det indstillede KPI 3. Systemet persisterer ikke KPI opdateringer

Tabel 4.7: UC2 - Analyser KPI

4.4.2.4 Skift KPI visning

Navn	Skift visning
Prioritet	`mellem`
Beskrivelse	Nogle brugere kan lide at se på datavisualiseringen i form af grafer, hvor andre hellere vil se det på tabelform. En bruger skal have mulighed for at kunne skifte visningen af et KPI på et Dashboard. Udførelse af denne use case skal resultere i at det pågældende KPI's visningsform skifter fra graf til tabel og omvendt.
Udløser [Triggers]	
Indgangsbetingelser	• Brugeren er autentificeret • KPI'et eksisterer (er repræsenter et på et Dashboard)
Udgangstilstand	Et KPI visning er ændret
Normal flow	1. Brugeren indikerer at ville se KPI'er repræsenteret på et Dashboard 2. Systemet viser Dashboardet og dets indhold af KPI'er 3. Brugeren vælger et KPI 4. Systemet viser detaljer for KPI'et 5. Brugeren indikerer at ville skifte visning 6. Systemet skifter KPI-visningen mellem graf og tabelform 7. Brugeren indikerer at ville persister ændringen 8. Systemet persisterer KPI ændringen
Alternativt flow	8a. Brugeren indikerer ikke at ville persistere ændringen. 1. Brugeren undlader at indikere persister KPI ændringen. 2. Systemet viser KPI ændringen 3. Systemet persisterer ikke KPI opdateringer

Tabel 4.8: UC2 - Skift KPI visning

4.4.3 UC3 Konfigurer KPI

Med KPI'er overvåger man en virksomheds finacielle nøgletal og får derved et indtryk af virksomhedens formåen. En bruger skal have mulighed for at kunne analysere på de data, der ligger til grund for det pågældene KPI. Denne use case dækker over disse analyse funktioner, en bruger skal kunne udføre på et KPI. Følgende analyseindstillinger vil jeg bestræbe mig på som minimum at implementere:

Sammenligning med forrige regnskabsperiode

En mulighed for at tilvælge en sammenlignings kurve i sin graf, således

at man kan se en sammenligning af nøgletallet med samme nøgletal fra forrige periode.

Tidsperiode

En bruger skal kunne vælge at se data for følgende tidsperioder:

- Indeværende regnskabsår
- Forrige regnskabsår (Business Monitor synkronisere data for de seneste to regnskabsår)
- De sidste 12 måneder
- Et interval hvor brugeren vælger fra og til måned/dato

Dimensioner for data

Jeg vil som minimum i prototypen give mulighed for at kunne vælge mellem enten tidsdimension eller afdelingsdimension, således at brugeren har mulighed for at vælge at se udviklingen over tid, eller over afdelinger i virksomheden. Man skal kunne vælge at se sine data over tidsdimension eller afdelingsdimension. For tidsdimensionen skal brugeren kunne vælge enten *måned* eller *kvartal* som repræsention for dimensionen.

4.4.3.1 Konfigurer KPI

Navn	Konfigurer KPI
Prioritet	`mellem`
Beskrivelse	Denne use case dækker over nøgletalsanalyse. En bruger skal have mulighed for at kunne indstille det valgte nøgletal. En bruger skal kunne indstille nøgletallet så det er muligt at sammenligne med tal fra forrige regnskabsår, vælge tidsperioden for regnskabsåret og/eller vælge den dimension brugeren ønsker at se tallene over.
Udløser [Triggers]	Instans funktioner visualiserer det valgte KPI med ændrede indstillinger.
Indgangsbetingelser	• Brugeren er autentificeret • Et KPI er valgt
Udgangstilstand	KPI analyse er foretaget, og beregningen er visualiseret på et Dashboard.

Fortsætter på næste side

Tabel 4.9 – *Fortsættelse fra forrige side*

Normal flow	1. Brugeren indikerer at ville se KPI'er repræsenteret på et Dashboard 2. Systemet viser Dashboardet og dets indhold af KPI'er 3. Brugeren vælger et KPI 4. Systemet viser detaljer for KPI'et 5. Brugeren indikerer at ville konfigurere et KPI'et 6. Systemet præsenterer brugeren for konfigurations muligheder 7. Brugeren instiller et KPI ved brug af de tilgængelige konfigurations muligheder 8. Brugeren indikerer at ville beregne det indstillede KPI 9. Systemet beregner det indstillede KPI på baggrund af de nye indstillinger 10. Systemet opdatere visualisering af det pågældende KPI med resultet af beregningen *Brugeren gentager trin 7-10 for hver ønskede indstilling* 11. Brugeren indikere at ville persister ændringerne 12. systemet initialiserer og visualiserer det tilføjede KPI 13. Systemet persisterer KPI ændringen
Alternativt flow	11a. Brugeren indikerer ikke at ville persistere ændringen . 1. Brugeren undlader at indikere persister KPI ændringen. 2. Systemet viser KPI ændringen 3. Systemet persisterer ikke KPI opdateringer

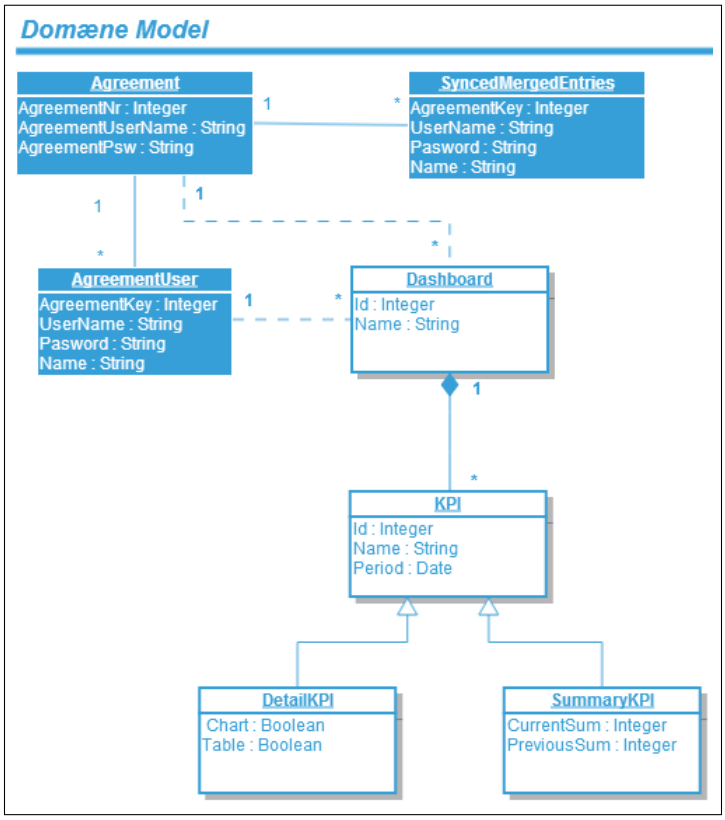
Tabel 4.9: UC3 - Konfigurer KPI

4.5 Domænemodel

I dette afsnit vil jeg udforme endnu et analyseartefakt, nemlig en Domænemodel. Med en domænemodel illustreres de bemærkelsesværdige koncepter i et forretningsdomæne. Flere af domænemodellens koncepter vil senere i design kapitlet blive til nogle software objekter. Figur 4.5 illustrerer en domænemodel for systemet. Modellen medtager nogle relevante entiteter, der allerede eksisterer i Business Monitors arkitektur. Disse er vist i figuren med farvet baggrund, og ligeledes fremgår det af figuren, hvordan prototypens konceptuelle elementer optræder i forhold de eksisterende entiteter i systemet.

Agreement

Agreement et en entitet i Business Monitor, der vedligeholder information om de oprettede Business Monitor aftaler.



Figur 4.5: Domænemodel for prototypen.

AgreementUser

På en Business Monitor aftale kan der være tilknyttet en eller flere bruger med hver deres rettigheder. Det kan f.eks. være, at en BrugerAftale kun har læserettigheder til Dashboardet, mens en anden med rettigheder til at oprette og redigere.

SyncedMergedEntries

Dette element repræsentere de synkroniserede data, som Business Monitor henter fra kildesystemet for en aftale.

Dashboard

For prototypen har jeg identificeret en række konceptuelle klasser, som illustrerer hvordan domæneobjekterne forholder sig til hinanden. Dashboards vil være tilknyttet en AgreementUser- og/eller Agreement entitet. Jeg er på nuværende tidspunkt usikker på, om jeg vil nå at implementere den detalje, om at give brugeren mulighed for at tilknytte et Dashboard enten en Agreement eller AgreementUser. Forskellen vil være at et Dashboard tilknyttet en Agreement vil være tilgængelig for alle BrugerAftaler. Jeg vil

dog som minimum implementere løsningen således, at Dashboards knyttes til en BrugerAftale (AgreementUser). Det er vist, at en bruger kan have flere Dashboards, men et Dashboard kun kan være tilknyttet en enkel bruger. Et Dashboards indhold består af nogle KPI'er, som er vist i figuren med et kompositionsforhold. Et KPI på et Dashboard kan optræde som et Detalje-Kpi eller et Overbliks-Kpi.

4.6 Opsummering

Der skal udvikles en HTML5 version af Busiess Monitor Dashboard applikationen ved anvendelse af teknologierne analyseret i kapitlet Teknologi Analyse. Problemområdet for projektet blev defineret i Introduktion kapitlet og krav for prototypen specificeret i Kravspecifikation kapitlet.

I dette kapitel blev en overordnet arkitektur for prototypen faslagt. De forskellige lag i arkitekturen blev identificeret, og de blev relateret til de teknologier, der vil blive anvendt for at opføre disse lag. Der blev desuden i dette kapitel beskrevet i detaljer nogle af de vigtigste use cases for systemet. Grundstenene for systemet blev derved lagt, og i det næste kapitel vil hovedvægten blive lagt på at designe en løsning for prototypen.

Kapitel 5

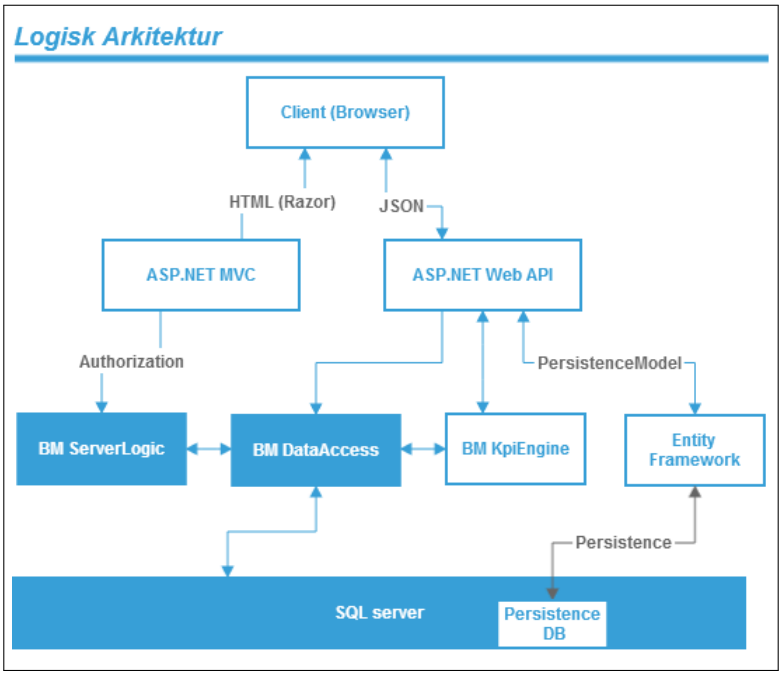
Design

Fokus hidtil i forløbet har været rettet mod analyse af krav til systemet. I dette kapitel flyttes fokus mod design af en løsning til prototypen, i form af software komponenter, objekter og samarbejdet imellem dem. I dette kapitel ser jeg på den logiske arkitektur af prototypen og dens organisering i lag. Jeg vil også komme med et foreslag til brugergrænsefladen og dens struktur, der angiver hvorledes indhold skal organiseres i brugergrænsefladen. Dernæst har jeg et afsnit dedikeret til emnet Responsive Design, der omhandler en applikations evner til at kunne tilpasse og optimere måden, hvorpå indhold i brugergrænsefladen er præsenteret, afhængig af skærmstørrelse og typen af enheden. Jeg vil behandle de forskellige softwarekomponenter i nærmere detaljer og udarbejde klassediagrammer for hver af dem. Jeg vil ligeledes se på nogle designmønstre til bedre organisering af software komponenter, men også specifikt i forhold til sproget JavaScript, da klientlaget vil blive implementeret ved brug af JavaScript.

5.1 Logisk Arkitektur

Figur 5.1 viser en lagdelt, logisk arkitektur for prototypen. De elementer i figuren, der har en baggrundsfarve repræsenterer nogle eksisterende elementer i Business Monitor arkitektur, der har nogle tjenester, som de andre elementer, der opbygges i forbindelse med projektet, kalder på og bruger for at fuldføre deres funktioner. Lad os se nærmere på de forskellige elementer af figur 5.1.

UI Det øverste lag består af UI (brugergrænseflade) og andet klientsidekode, der repræsenterer adfærd på klienten. Det er f.eks. JavaScript kode, jeg kommer til at skrive for at opfylde prototypens funktioner, men det består også af andre JavaScript biblioteker, jeg kommer til at anvende på klientsiden. Som vist i figuren, får klientelementet serveret HTML, JavaScript, CSS og andet klientsidekode gennem serverelementet ASP.NET MVC 4. Med andre ord bruger jeg ASP.NET MVC 4 til at generere HTML sider til klienten.



Figur 5.1: Lagdelt logisk arkitektur.

Web API ASP.NET WEB API er et serverside-element, som muliggør at eksponere data og tjenester over HTTP. At eksponerer data og tjenester på den måde gør det nemmere at integrere funktionalitet med en bred vifte af device- og klientplatforme. Prototypen vil have adgang til dette WEB API element og vil kune udføre sine funktioner ved at interagere med denne Web API.

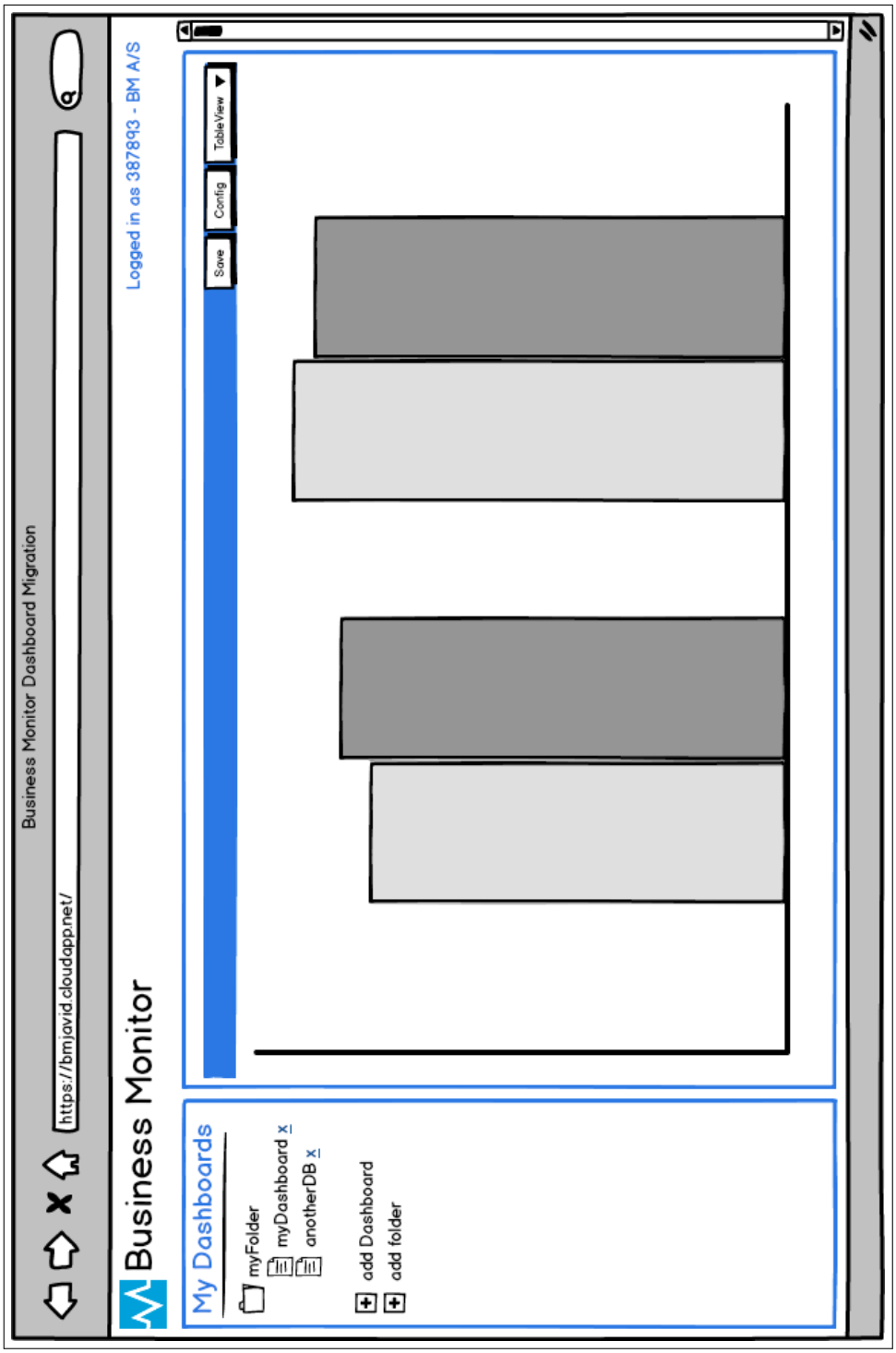
BM KpiEngine BM KpiEngine repræsenterer det element på serversiden, som er ansvarlige for beregning af nøgletal. Som jeg tidligere har nævnt, er i den eksisterende Silverlight løsning, den nødvendige logik til beregning og analyse af nøgletal allerede tilstede i Silverlight klienten. Efter samråd med virksomheden kom jeg frem til at implementere en begrænset del af KPI beregningslogik som et element på serveren. Man vil senere kunne videreudvikle på det og implementere alle nødvendige nøgletals beregninger og analyse indstillinger, som man gerne vil understøtte i systemet. I forbindelse med udvikling af prototypen, vil jeg i dette element give mulighed for beregning af en række simple KPI'er, der i Business Monitor er katagoriseret som `Finance KPI'er`, da jeg helst ikke skal bruge for meget tid på det, idet det ikke er projektets fokusområde. BM KpiEngine elementet kommunikerer med Business Monitors ORM[1] lag, BM DataAccess, når en brugers nøgletal skal beregnes.

[1] Object-relational mapping eller ORM er en teknik til at konvertere data mellem fx et objekt-orienteret sprog og en relationel database

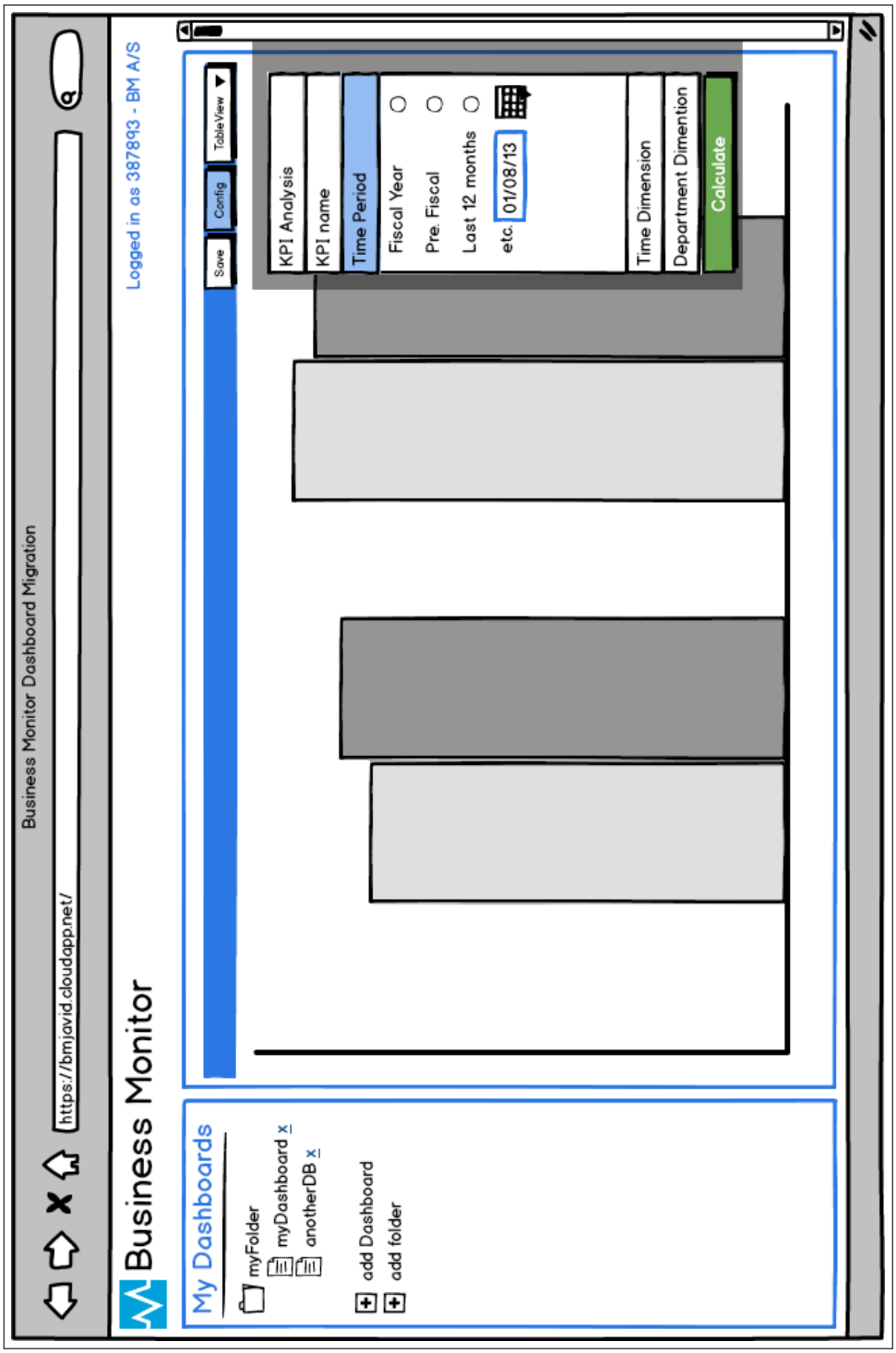
Persistens Det element i figur 5.1, som jeg har identificeret som Entity Framework, repræsenterer den logik på serversiden, der skal til for at understøtte persistens. Som det kan ses, bruger jeg Entity Framework, som er et ORM framework fra Microsoft, til at kommunikere med en SQL database, som jeg opretter til formålet for at oversætte C# Model objekter til database entiteter. Disse entiteter bruges i WEB API.

5.2 Brugergrænseflade og struktur

Der har været et ønske om, at visningsområdet, hvor graferne repræsenteres, er så stort som muligt. Derfor skal det rigtige indhold organiseres i en logisk struktur, hvor brugeren igennem brugergrænsefladen nemt kan udføre sine opgaver. Jeg har efter samråd med virksomheden kommet med dette foreslag til brugrænsefladen som vist i figur 5.2. Jeg har i forbindelse med tegning af denne mockup brugt værktøjet Balsamiq Mockups, der er et værktøj til at lave mockups med. Det blev besluttet at repræsentere en brugers Dashboards som et træstruktur i brugergrænsefladen. Strukturen består af et hierarki, hvor en brugers Dashboards er organiseret i nogle mapper (**Folders**). En bruger har en række mapper. Hver mappe indeholder en rækker Dashboards. Et Dashboard indeholder en række nøgletal repræsenteret som grafer eller tabeller. Som vist i figur 5.2 består brugergrænsefladeindretningen af 3 områder; en `header`, et navigationspanel til venstre og et indholdsområde. Jeg vil prøve at indrette brugergrænsefladen således, at menuer eller paneler med knapper og andre brugergrænseflade elementer ikke hele tiden er synlige og optager plads på brugergrænsefladen. Det er for det meste grafer eller tabeller, man er interesseret i at se, og derfor er det bedre at skjule andre elementer, som man vil kunne folde ud, når man ønsker at interagere med systemet. Dette har jeg forsøgt at illustrere i figur 5.3, hvor man ved at trykke på knappen `Config` folder et konfigurationspanel ud, hvor man kan arbejde med indstillinger af den valgte graf.



Figur 5.2: Mockup - brugergrænseflade og struktur.



Figur 5.3: Mockup - KPI analyse menu.

5.3 Server side Teknologier

Jeg vil i dette afsnit se på de server-side-teknologier, jeg vil bruge i forbindelse med udvikling af prototypen. Business Monitor applikationen kører i skyplatformen Window Azure. så lad os starte der.

5.3.1 Window Azure

Azure er Microsofts skyplatform, der giver mulighed for at opbygge, installere og adminstrere programmer. Azure kører på et globalt netværk af datacentre. Dette giver fleksibilitet omkring valget om, hvor i Azure infrastrukturer man vil installere og kører sine programmer, afhængig af hvor ens kunder er. Azure har også SQL server tjenester, som Business Monitor naturlivis bruger. I forbindelse med udvikling af prototypen kommer jeg ikke til at bruge Windows Azure på anden vis, end til at oprette en ny database (i den logiske SQL Database Server Business Monitor har på Windows Azure) til persistens, og at jeg vil deploy løsningen til Azure.

5.3.2 Database

Databasen er som nævnt en Azure SQL Database, som er en relationel database tjenester på Azure platformen. Her ligger alt data, som Business Monitor vedligeholder og synkronisere med kildesystemet. BM DataAccess som vist i figur 5.1 er en slags ORM, der mapper database entiteter til C# objkter. Men i forbindelse med persistens vil jeg bruge Entity Framework (EF) som min ORM.

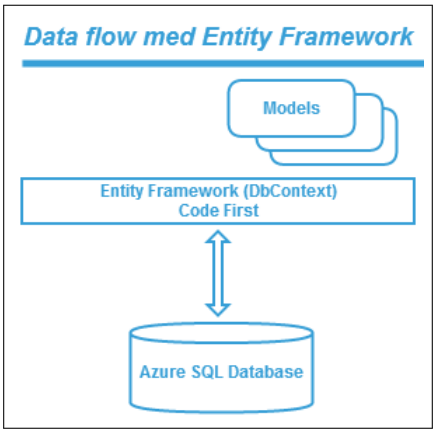
5.3.2.1 Entity Framework

Entity Framework er et Object Relational Mapper framework og har til formål at øge en udviklers produktivitet ved at generere de nødvendig database instruktioner for læsning og skrivning af data og eksekvere dem i databasen. Entity Framework har en Code First feature, som jeg vil bruge til at generere en database til Dashboard persistens. Code First vil generere databasen baseret på nogle C# Model objekter (POCO)[2] og oprette tabeller for hver klasse. Klassernes attributter bliver mappet til kolonner i den pågældende tabel.

Vha. Entity Framework kan jeg nemt tilgå data. Ved at oprette en DbContext klasse kan jeg håndtere forbindelse mellem mine model objekter (PersistensModel) og database tabeller. Den giver mig også mulighed for at bruge LINQ, og derved generere forespørgsler på mine models, så de kan tilgås og manipuleres.

Som vist i figur 5.4 kan jeg interagere med databasen vha. Entity Framework. Den har en klasse kaldet DbContext, som bruger nogle Model objekter og definerer relationen mellem disse og databasen. Så EF vil mappe min model, lad os sige Dashboard objekt, ned til f.eks. en Dashboard table i databasen. Model

[2]Plain Old CLR Object



Figur 5.4: Data flow med Entity Framework.

objekterne er nogle almindelige klasseobjekter (POCO), som EF bruger og fylder for os.

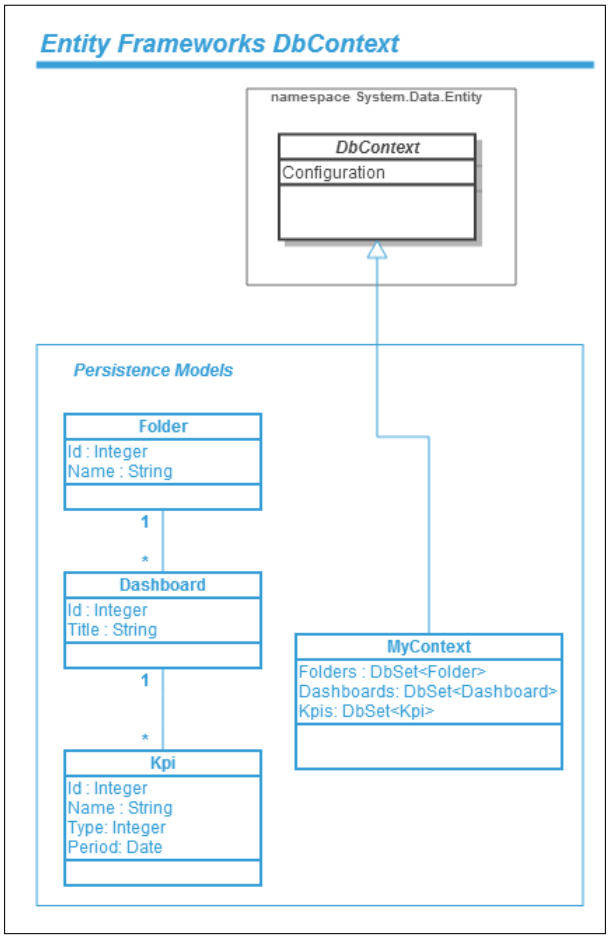
Models og DbContext Model objekter er en slags beholder for data. Når vi henter data fra databasen, er vi nødt til have dem et sted, så vi kan bruge dem og transportere dem på tværs af applikationen. Vi kender dem også som entiteter eller domæneobjekter. DbContext[3] definerer sammenhængen mellem modelobjekter og databasen og mapper dem frem og tilbage. DbContext administrer tilstanden af mine modelobjekter i sin context i hukommelsen. Jeg viser i figur 5.5 en model for persistens, som viser hvordan EF og dens DbContext klasse bliver brugt til at hente og persistere data i databasen.

Figur 5.5 viser mine modelklasser med nogle properties (det vil nok være nødvendigt at tilføje mange flere properties for at være i stand til at kunne genskabe applikationens tilstand på klienten). Her er det også vist hvorledes EFs DbContext klassen er nedarvet for at definere og relatere model klasserne til databasen vha. typen `DbSet<T>`, hvor `T` angiver min model type (fx `DbSet<Dashboard>`). I den nedarvede DbContext klasse defineres, hvilke data, der skal kunne håndteres. Hver definition af `DbSet<T>` sætter en relation mellem en databasetabel og et modelobjekt.

5.3.3 Web API

En web applikation som denne involverer en masse client-server kommunikation, som indebærer udveksling af data. Her tænker jeg specifikt på JSON data, som bruges i denne applikation. Business Monitor har et datalag (BM DataAccess), som jeg har beskrevet tidligere. I forrige afsnit beskrev jeg et andet datalag (Persistence), understøttet med Entity Framework, som jeg vil tilføje for at implementere persistens i prototypen.

[3] `http://msdn.microsoft.com/en-us/data/jj729737.aspx`



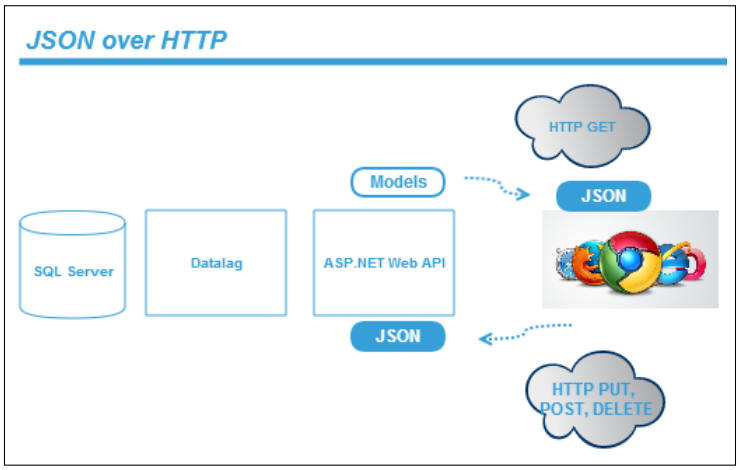
Figur 5.5: Persistens Model for Dashboard prototypen.

I dette afsnit ser jeg på, hvorledes jeg kan sætte en grænseflade op mod, på den ene side Business Moniters server side tjenester og klient kode. Det kan jeg gøre med ASP.NET WEB API. Så WEB API vil være det lag, der (for)binder vores server-side-datalag, og klientlaget sammen.

5.3.3.1 Hvorfor Web API?

Flere forskellige enheder og platforme, skal kunne sende forespørgsler til vores webserver, og vi skal kunne transportere data til- og fra server og klienten over HTTP. Vi vil gerne være i stand til at konvertere vores modelobjekter bygget på serveren til JSON, så vi kan sende dem til en webbrowser. ASP.NET WEB API passer godt i den forbindelse, fordi det virker godt med ASP.NET MVC, og fordi det er idealt til håndtering af JSON data, da det er bygget ind i ASP.NET

WEB API. Desuden simplificerer den webtjenester for os, hvilket vil effektivisere implementering.



Figur 5.6: ASP.NET WEB API - servere JSON over HTTP.

Vi har nogle modelobjekter på serveren, som skal serveres gennem WEB API, når vi får en GET request. Disse modelobjekter bliver konverteret til JSON form, når der modtages en forespørgsel. Og ligeledes når der laves en HTTP PUT, POST eller DELETE forespørgsel, får vi JSON data, som på serveren bliver konverteret tilbage til modelobjekter. Dette forhold er vist i figur 5.6.

5.4 Responsive Design

Når man bygger en web applikation i dag, tager man stilling til, om man vil have applikationen til at virke på forskellige enheder og platforme. Det er f.eks. enheder som smartphones, tablets, laptops og store skrivebordsskærme. Dertil kommer de forskellige platformteknologier (iOS, Android, Windows Phone), som skal tages med i overvejelserne, om ens applikation skal understøtte. Der er stadig stigende antal Internet brugere, der browser på Internettet ved at bruge de nævnte mobile enheder, og forretninger bliver nødt til at følge med; man vil gerne være der, hvor forbrugerne er. Dette giver anledning til at man som beslutningstager skal lægge en mobilstrategi.

5.4.1 Mobil Strategi

Native App eller Responsiv Webdesign Det mobile Internet har taget fart, og flere og flere hopper med på vognen. Der er derfor ingen tvivl om, at virksomheder som følge af denne udvikling, gerne vil repræsentere sig selv og sit brand på det mobile marked. Der skal derfor lægges en strategi for, hvordan

det skal gøres, da der er flere måder, hvorpå man kan introducere sig selv på det mobile marked [1].

Når der skal træffes et valg om en mobilstrategi, er der flere faktorer, der spiller ind. Hvad er vigtigt for virksomheden? Er det vigtigt, at man når ud til mange brugere, og være repræsenteret på tværs af enheder og på tværs af platforme. Eller er man interesseret i specialiserede apps på de enkelte platforme til den rigtige målgruppe. Noget der er vigtigt i den sammenhæng, og i forbindelse med native apps, er, at med native apps har man adgang til alle features på enheden og integrerer dybere end et mobiloptimeret website, ved at have adgang til funktioner på mobilen såsom GPS, adressebog, kamera, push-beskeder osv. [6, s. 10].

5.4.1.1 Native App

Native app dækker over udvikling af en app specifikt til en platfrom. På den måde får man adgang til alle enhedens features via dens API.

Fordele
En mere integreret app grundet adgang til enhedens API, giver mere personlig brugeroplevelse. Man får adgang til funktioner såsom notifikationer, GPS, Kalender, SMS. Man opnår en responsiv brugergrænseflade, da appen er optimeret til den enkelte platform. Der er desuden mulighed for at placere appen på de mobile appmarkeder.

Ulemper
Native apps kræver selvfølgelig større udviklingskompetence, større budget og mere vedligeholdelsesarbejde hos virksomheden, hvilket er en pris, man er villig til at betale, hvis virksomheden er stor. En native app kræver at brugeren opdatere appen, når der er sket ændringer i appen.

5.4.1.2 Responsiv Design

Det er helt klart den billigste løsning at udvikle et website, der virker optimalt på alle platforme og enheder. Her bygger man en standard webpplikation men med evnen til at kunne tilpasse udseende og, i nogle tilfælde også lidt adfærd, i forhold til skærmstørrelse. I forhold til mindre skærme gælder det om at omstrukturere brugergrænsefladen og skære ned i mængden af information, så kun det mest nødvendige vises, og derved give brugeren en bedre brugeroplevelse. Med Responsive Design skal man kun vedligeholde sit website et sted, og alle ændringer vil umiddelbart være synlige på alle platforme. En virksomhed behøver ikke at have forskellige udviklingskompetencer, som er noget, der betyder meget, når virksomheden ikke er så stor.

Fordele
Som nævnt ovenfor er det en billig løsning, når virksomheden ikke er så stor. Man er repræsenteret på alle platforme uden meromkosninger.

Ulemper
Ingen adgang til enhedens API og features såsom GPS, Kontaklist, Kamera, notifikation beskeder osv. Man har desuden, i modsætning til native app, ikke mulighed for annoncering på app markeder.

Ud fra det ovenstående er Business Monitor nødt til i første omgang, at satse på Responsive Design, og udnytter de muligheder det giver, for at blive introduceret på det mobile marked. På en smartphone kunne man f.eks. give en bruger mulighed for at overvåge sin virksomheds økonomiske nøgletal i et Dashboard, uden mulighed for at lave nøgletalsanalyse. Brugeren kunne f.eks. også have mulighed for at browse mellem forskellige Dashboards.

5.4.2 Responsive Design vha. Media Queries

Responsive Design dækker over begreberne Adaptive Design og Fluid Design. Responsive er det overornede begreb, og det angiver, at sidens layout ændres og tilpasses. Adaptive og Fluid angiver, hvordan ændringen sker. Med Adaptive design er sidens layout stadig baseret på statiske pixel-værdier og ved bestemte opløsninger angiver man vha. CSS3 **Media Queries**, hvad disse pixel-værdier skal ændres til. Med Fluid design er layout baseret på procentværdier, og ændring sker derfor mere flydende [6, s. 59].

Media Queries Vha. media queries kan vi specificere nogle CSS styles til at blive anvendt, afhængig af skærmstørelsen af det device, der viser siden. Media queries er ligesom et spørgsmål til browseren. Hvis browseren kan svare bekræftende på spørgsmålet, vil den efterfølgende blok af styles blive anvendt, ellers ikke. fx

Listing 5.1: Media Queries eksempel

```
1 @media screen and (max-device-width: 400px) {
2     h1 { color: green }
3 }
```

Kode oversigt 5.1 vil skifte farven af alle `h1` elementer på siden til grøn, hvis enheden har en skærmbrede på 400 pixels eller derunder.

5.5 Design Oversigt

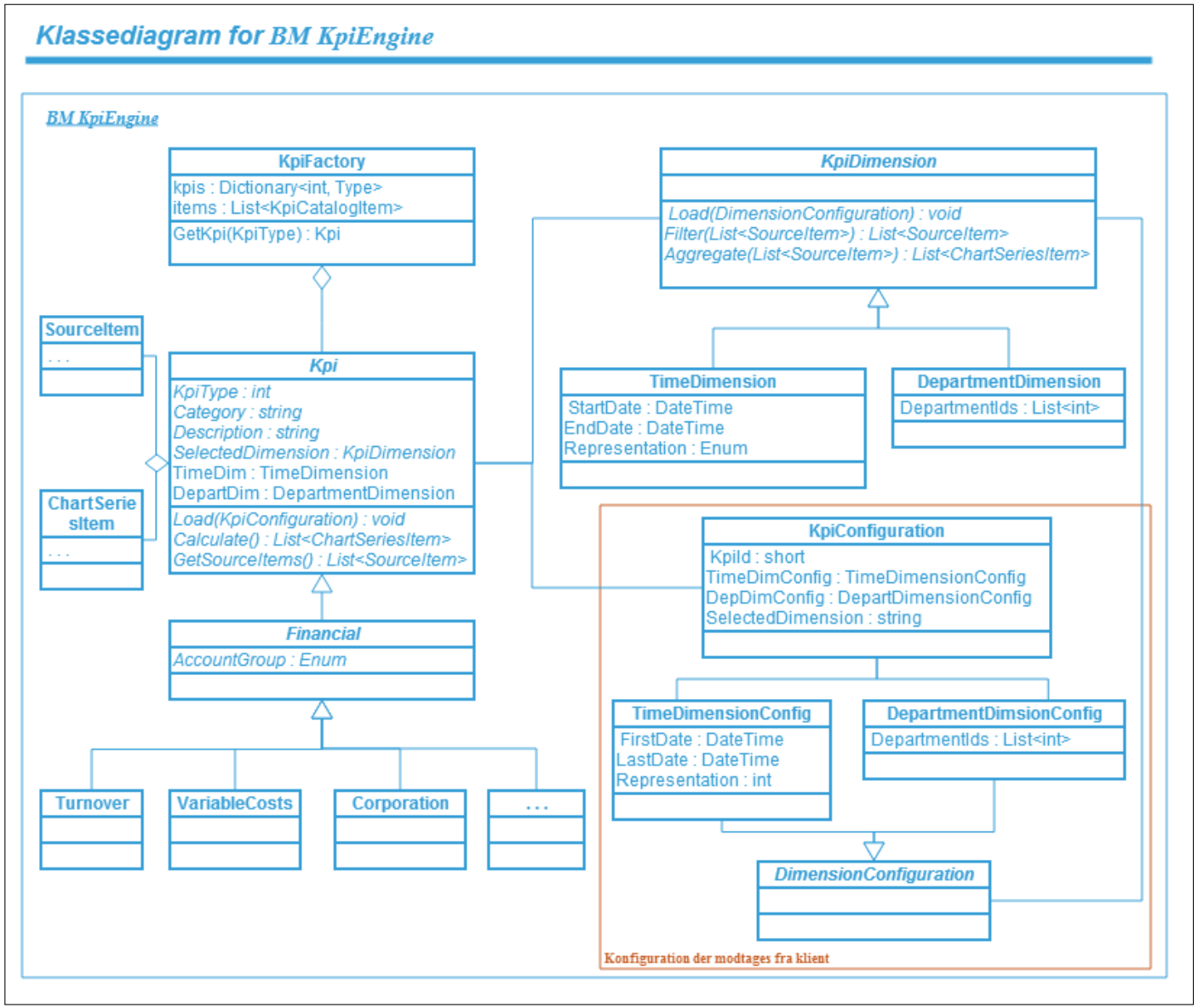
I dette afsnit vil jeg se nærmere på de forskellige komponenter, som er vist i figur 5.1 under afsnittet Logisk Arkitektur. Jeg vil beskrive, hvordan komponenterne er bygget op, og hvilke elementer de består af.

5.5.1 BM KpiEngine

Som jeg tidligere har været inde på, skal jeg implementere en server-komponent der har ansvaret for beregning af de KPI'er, der indgår i prototypen. Idet komponenten bliver implementeret som et modul, og har en grænseflade mod resten af systemet, vil man senere kunne udvide modulet med den nødvendige logik for at understøtte beregning af alle de KPI'er, der findes i Business Monitor Silverlight applikationen, uden at det har nogen betydning for prototypen. Prototypen vil automatisk kunne håndtere de nye KPI'er. Dette er i overensstemmelse med SRP (**Single responsibility principle**) konceptet i objekt orienteret programmering.

Et KPI i det følgende skal betragtes som en abstrakt struktur på serveren, som repræsenterer et økonomisk nøgletal. I prototypen, når der skal vises en oversigt over de tilgængelige KPI'er i systemet, kan jeg via `Reflection`[4] oprette intanser af de klassestrukturer, som arver fra KPI klassen, og får derved adgang til informationer om deres navne og typer, så jeg på klienten kan lave en oversigt over de tilgængelige KPI'er. Når et KPI skal beregnes, skal jeg på klienten sørge for, at der bliver sendt en konfiguration indeholdende oplysninger om, hvilken KPI der ønskes beregnet, samt andre informationer som er nødvendige for beregning af det valgte KPI - se klassediagrammet i figur 5.7. Det kan være informationer om regnskabsåret, dimensioner på KPI'et, og andre indstillinger. Der vil være adgang til brugerens kildedata via modulet BM DataAccess, som er de data, der ligger til grund for et KPI. Så i BM KpiEngine modulet har jeg nogle KPI'er - de har kildedata til rådighed, og kan på baggrund af den modtagne konfiguration, beregne og returnere data, der på klienten skal bruges som input til datavisualiseringskomponenten. Den modtagne konfiguration vil blandt andet indeholde information om, hvilken dimension er den valgte. Et KPI har et antal dimensioner. Hver dimension har nogle settings, og det er disse settings på aktive dimensioner, der afgør hvordan nøgletallet bliver beregnet. Jeg har valgt som minimum at implementere tidsdimension og afdelingsdimension for de KPI'er, der indgår i prototypen.

[4]Inden for programmering er `Reflection` en process, der giver mulighed for at modificere et objekts struktur of adfærd ved runtime.



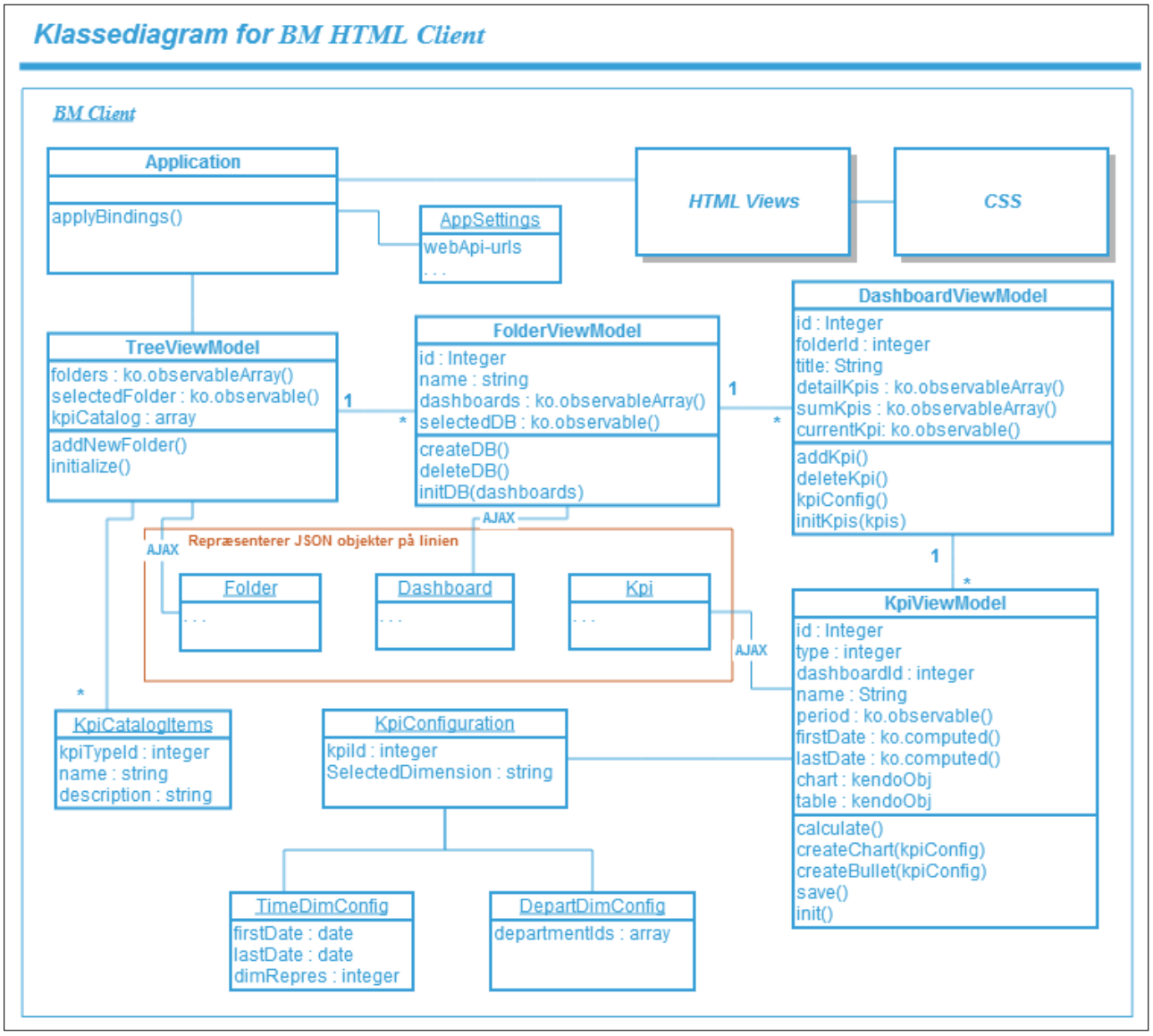
Figur 5.7: Klassediagram for `KpiEngine` modulet.

Figur 5.7 viser et klassediagram for KpiEngine modulet på serveren. Overordnet kan man sige, at modulet består af nogle klasser, der angiver KPI-struktur. Nogle klasser til implementering af KPI dimensionerne. Og så er der klasser til at repræsentere KPI konfigurationerne. Konfigurationerne som nævnt vil komme fra klienten, og vil være repræsenteret i disse klasser. Desuden er der nogle klasser til at repræsentere input data og resultater af beregninger. Figuren giver et overblik over KpiEngine modulet med de vigtigste klasser, metoder og attributter.

`KpiFactory` er klassen Web API vil kontakte, når der bliver spurgt efter et KPI. Forespørgslen vil indeholde information om det efterspurgte KPI i et konfigurationsobjekt. Funktioner i `KpiFactory` vil via Reflection oprette en instans af det ønskede KPI, som vil modtage det indkommende konfigurationsobjekt. KPI'et vil nu opsætte dimensionerne på baggrund af information i det modtagne konfigurationsobjekt. Herefter vil KPI'et, via BM DataAccess modulet, hente den relevante mængde kildedata, som bliver filtreret og aggregeret i forhold til egenskaber defineret i dimensionerne. Resultatet i form af en liste af `ChartSeiresItem` objekter, returneres til WEB API Controlleren, som havde startet procesen.

5.5.2 Client - oversigt og opbygning

Klientlaget består- og bygges ved brug af HTML, CSS, JacaScript (biblioteker og frameworks). Desuden vil der på klientsiden blive anvendt nogle designmønstre til bedre håndtering af JavaScript kode men også til at opnå bedre adskillelse af de forskellige elementer i laget - adskillelsesprincippet (på Eng. Separation of Concerns). Dette opnår jeg ved at anvende designmønstret MVVM ved brug af JavaScript biblioteket Knockoutjs. Figur 5.8 giver et overblik over elementerne på klienten. I forhold til MVVM kan man sige, at HTML og CSS elementerne, som vi ser øverst til højre i figuren, angiver Views. Det er tydeligt at se i figuren, hvad der er ViewModels. Models er i denne sammenhæng defineret ved de JSON objekter, der er på "linjen", og transmitteres frem og tilbage mellem klientkode og serveren (WEB API) via AJAX kald. Disse har jeg i figuren markeret med indramning. Jeg ved på nuværende tidspunkt, at det i hvert fald er disse tre type objekter, der bliver transmitteret over linjen. Men som jeg viste i forrige afsnit om KpiEngine modulet, vil en beregning af KPI resultere i en list af objekter af typen ChartSerieslem, som på klientsiden vil blive brugt som input til Kendo UIs Datavisualiserings komponenten i forbindelse med rendering af en graf. Så listen af ChartseriesItem objekter, der på klienten modtages som et array af JSON objekter, indgår også som en del af mine Models. Jeg gemmer dog ikke denne liste, som er resultatet af beregning af et KPI. Jeg gemmer indstillinger på et KPI, hvis beregningsresultat vil give den samme graf.



Figur 5.8: Klassediagram for `HTML Klient` laget.

Klassediagrammet viser systemets klasser, strukturer og deres sammenhæng. Da klientlaget skal bygges med JavaScript, er det vigtigt at forstå, hvad objekter i JavaScript er. Objekt-orienteret sprog er kendetegnet ved deres brug af klasser, til at lave flere instanser af objektet med samme attributter og metoder. JavaScript har ikke et *class* koncept, og objekter er derfor forskellige end dem i objekt-orienteret sprog. I det følgende afsnit vil jeg se nærmere på objektkonstruktion i JavaScript.

5.6 JavaScript Objekt Konstruktion & Pattern

JavaScript har ikke et *class* begreb. JavaScript har dog objekter, men JavaScript objekter er ikke ligesom objekter vi kender fra objekt-orienteret sprog som C# og Java, hvor objekter er noget der indkapsler tilstand og adfærd. JavaScript har en mere svag definition af, hvad et objekt er. Objekter i JavaScript er mere ligesom en samling af key/value par, hvor key repræsenterer attribut- og metode navne og values repræsenterer attributværdier og metode kroppe. Så objekter i JavaScript er som et dictionary eller hashtable. Måden hvorpå man tilgår et objekts attributter og metoder kan minde meget om den i C# og Java. Denne pointe er illustreret i kodeoversigt 5.2 nedenfor. Her kan vi se, at både attributnavn og metodenavn er som en key i et dictionary. Og deres værdier er hhv. attributværdien og metodes krop. Vi kan også bruge punktum notationen som vi kender fra objekt-orienteret sprog for at tilgår attributter og metoder.

Listing 5.2: Objekter i JavaScript

```
var dashboard = new Object();
// new keyword i JavaScript er ikke som i C# eller Java.
// Det er bare en af de maader man kan initialisere objekter paa.

dashboard['title'] = 'Finances';     // definerer key/value par
dashboard['addKpi'] = function() {
    var kpi = ...;
    kpis.push(kpi);
}

// punktum notationen
dashboard.title = 'Finances';
```

5.6.1 Constructor Pattern

Jeg vil bruge JavaScripts objektmodel til at repræsentere noget, der ligner klassekonstruktion i C# og Java, så jeg kan får en bedre mapping mellem serversidekode og klientsidekode. Så spørgsmålet er, hvordan kan jeg definere konceptet *class* i JavaScript, som den vi kender fra objekt-orienteret sprog. I kodeoversigt 5.2 ovenfor laves direkte en objektinstans. Der er ikke en *class* difinition. Hvad hvis jeg vil lave flere `dashboard` objekter. Skal der så laves en ny instans og alle attributterne redifineres med nye værdier? Der er brug for en model, der ligner en *class*. Her kommer design pattern ind i billedet. Man laver en `function`,

der indkapsler konstruktionen af objekter med specifikke interfaces. En sådan JavaScript funktion kaldes for en `constructor function`[9, s. 182]. Den giver en slags template af et objekt, som kan bruges til at lave flere instanser af. Der defineres en `function`, som fungerer både som klassedefinition og `constructor` som vist i kodeoversigt 5.3.

Listing 5.3: Objekter i JavaScript - `constructor function`

```
function Dashboard(id, title) {
    this.id = id;                // definer id key/value par
    this.title = title;          // definer title key/value par

    this.addKpi = function() {  // definer addKpi key/value par
        var kpi = ...;
        kpis.push(kpi);
    };
}
```

Her har jeg defineret en `constructor function`, som jeg så kan bruge til at lave instanser af. Og hvordan gør man så det? Det kan man gør på flere måder. Man kan bruge `call(...)` metoden, som er tilgængelig på alle objekter[9, s. 183]. `call(...)` metoden tager som sin første parameter en værdi, som repræsenterer det objekt, `this` skal refererer til. Så jeg kan sige:

Listing 5.4: Objekter i JavaScript - objekt instanser

```
var dashboard;
dashboard = {}; // {} laver et tomt objekt
Dashboard.call(dashboard, 1, 'Finances'); // dashboard obj er
    ejeren

// en anden maade, som vi godt kender
var dashboard;
dashboard = new Dashboard(1, 'Finances');
```

En nemmere og kortere måde at lave instanser, baseret på `constructor function`, er ved at bruge `new` operator, også vist i kodeoversigt 5.4. som gør det samme som `call(...)` metoden, og er forskellig fra `new` keyword i C# eller Java. Når en `constructor function` bliver kaldt på den måde ved brug af `new` operator, sker der følgende:
Der skabes et nyt objekt, og `constructors this` værdi tildeles det nye objekt, således at referencen `this` peger på det nye objekt. Koden inde i `constructor`'en eksekveres, og attributter og metoder tilføjes derved til det nye objekt, som så returneres.

Det er vigtigt at nævne, at `constructor function` ikke er en decideret funktion i forhold til andre JavaScript funktioner. Forskellen ligger bare i den måde, funktionen bliver kaldt på. Vi kalder den en `constructor function`, når funktionen bliver kaldt som en konstruktør, fx vha. `new` operatoren. Det er en konvention at `constructor function` skrives med stort begyndelsesbogstav, lige-

som klassenavne i C#. På den måde kan man skelne mellem en `constructor function` og andre funktioner[9, s. 181], der skrives med lille startbogstave.

Ideen med dette afsnit er, at jeg i figur 5.8 viser et klassediagram over klientlaget, der illustrerer de elementer, klientlaget består af. Figuren beskriver systemets klasser og strukturen mellem dem. Idet klientlaget bliver implementeret vha. JavaScript, synes jeg, det er på sin plads at komme ind på objekter i JavaScript, og endnu vigtigere, objekt konstruktion i JavaScript. Den måde at konstruere JavaScript objekter på, som jeg har beskrevet i dette afsnit, kaldes Constructor Pattern[9, s. 181]. Der findes flere objektkonstruktions patterns, som er opstået i takt med udvikling af sproget. Jeg vil bruge den omtalte Constructor Pattern i udvikling af klientlaget til at strukturere og konstruere JavaScript objekter. Mere herom i næste kapitel, Implementering.

5.7 Klientstruktur

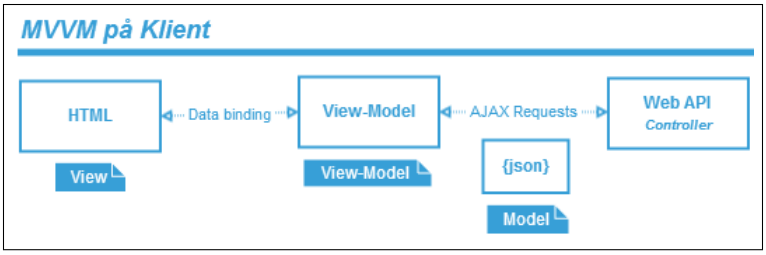
Strukturen i klientlaget er delt op i elementer, der repræsenterer hhv. brugergrænsefladen, systemets funktionalitet/adfærd og data. Dette er i overensstemmelse med design mønstret MVVM, som jeg beskrev i Teknologi Analyse kapitlet i forbindelse med JavaScript biblioteket Knockoutjs. Jeg har i Brugergrænseflade og struktursektion i dette kapitel præsenteret et foreslag til brugergrænsefladestruktur. Strukturen består af et navigationsområde i vestreside, hvor jeg gerne vil præsentere en brugers Dashboards i en træstruktur, hvor Dashboards vil blive organiseret i nogle foldere. Det valgte Dashboard vil blive vist i indholdsområdet til højre med alt dets indhold.

Designklassediagram vist i figur 5.8 fra forrige afsnit illustrerer elementerne i klientlaget. HTML og CSS elementerne fra figuren repræsenterer den foreslåede brugergrænsefladestruktur. Klassen TreeViewModel indkapsler funktionalitet til håndtering af brugerinteraktion i navigationsområdet i brugergrænsefladen. Som det kan ses, har klassen et array af folder objekter, repræsenteret ved klassen FolderViewModel i figuren. Denne har igen et array af dashboards objekter repræsenteret ved DashboardViewModel i figur 5.8. Som det kan ses, er det en hierarkisk struktur som elementerne i klientlaget er organiseret i. DashboardViewModel vedligeholder et array af KPI objekter, hvor et KPI, repræsenteret ved KpiViewModel, indkapsler funktionalitet omkring egenskaber ved et KPI. Således er alt adfærd indkapslet i ViewModel instanser og adskilt fra præsentationen (HTML). Disse ViewModel har ingen afhængighed til præsentationen (HTML). ViewModels repræsenterer i stedet for de abstrakte egenskaber af præsentationen, som fx "en liste af dashboards ".

5.7.1 At Bringe MVVM til Klienten

MVVM, som beskrevet i Teknologi Analyse kapitlet, er en adskillelsesmønster (Eng. separation pattern), der adskiller ansvarsområder for Model (data), View,

som i dette tilfælde er HTML, og ViewModel, som vi skriver i JavaScript. Jeg kan anvende MVVM på klienten ved brug af JavaScript biblioteket Knockoutjs, hvorved jeg kan binde HTML elementer med data. MVVM i form af Knockout har følgende elementer vist i figur 5.9



Figur 5.9: MVVM på klienten.

Model
: Model er server-side repræsentation af domænets data.

View
: View er præsentation af information på en brugervenlige måde. Det er HTML.

ViewModel
: ViewModel binder Model og View sammen. Det er et JavaScript objekt, indeholdende den nødvendige adfærd for et View og kan samle en eller Models for at vise data i Viewet.

Jeg vil i det følgende illustrere, hvorledes MVVM principperne kan anvendes i HTML og JavaScript ved brug af Knockout. Ved hjælp af et simpelt eksempel, vil jeg demonstrere, hvordan man kan adskille data (JavaScript), præsentationen (HTML) og adfærd (JavaScript). Så lad os starte med Viewet, som består af HTML vist i kodeoversigt 5.5 nedenfor.

Listing 5.5: MVVM med Knockout - View (HTML)

```
<!--data-bind attributter er maaden Knockout giver mulighed for-->
<!--deklarativt, at knytte ViewModel variable med DOM elementer-->

Name: <span data-bind="text: name"> </span>
Description: <span data-bind="text: description"> </span>
Price: <span data-bind="text: formatCurrency(salesPrice)"> </span>
```

Det man kan se her er 3 simple linjer af HTML kode, som viser information om et produkt. Som det kan ses, er der vist nogle `span` HTML elementer, som på nuværende tidspunkt ikke har noget indhold. Det er vores Model objekt der har det data, der skal bindes til disse `span` elementer. Vi kan se, at `span` elementerne har en `data-bind` attribut, som er HTML5 kompatible attribut, som Knockout bruger for at finde ud af, hvilken binding det drejer sig om. I dette tilfælde kan vi se, at det er *text* binding, som er en af de mange indbyggede Knockout bindings, der er tilgængelige. Her er det *text binding* der er bundet til nogle variable (Eng. properties). Men hvor er disse variable defineret?, og hvordan finder Knockout dem? Svaret er, ViewModel, som er vist i kodeoversigten 5.6. Model er også vist i den samme kodeoversigt. Binding-syntaksen kan vi se består af to elementer inde i `data-bind` attributten; *binding-navn* og *værdi*, adskilt af et kolon. *Binding-navn* kan være en af de mange indbyggede Knockout bindings, men Knockout giver også mulighed for at oprette sin egen bindings. Det gør man ved at registrere sin binding vha. `ko.bindingHandlers`. *Værdi*-delen i en binding-syntaks kan være en ganske almindelige JavaScript varaible, en Knockout `observable`, eller en metodekald. Metoden kan være defineret i ViewModel, men den kan også defineres inline i binding-syntaksen.

Listing 5.6: MVVM med Knockout - ViewModel & Model (JavaScript)

```
// The Model
var product = {
    id: 1001 ,
    name: 'Samsung GS4',
    description: 'The Samsung GLS4 captures all the action wherever
                  you are.',
    price: 600.088
};

// The ViewModel
var ViewModel = {
    id: ko.observable(product.id),
    name: ko.observable(product.name),
    description: ko.observable(product.description),
    salesPrice: ko.observable(product.price),

    formatCurrency: function(value) {
        return '$' + value().toFixed(2);
    }
};

// Bind the ViewModel to the View using Knockout
```

```
23 // Ved at kalde denne Knockout metode , bliver HTML's 'DataContext'
24 // sat til vores ViewModel
25 ko.applyBindings(ViewModel);
```

Denne kodeoversigt viser to JavaScript objekter, Model og ViewModel. Den sidste linje kode udfører en vigtig funktion. Den aktiverer Knockout og sætter ViewModel objektet som en slagt 'DataContext' for View'et. I eksemplet her har jeg brugt en anden måde at konstruere et objekt på, som kaldes Object Literal Pattern. Det er en hurtig og nem måde at konstruere objekter på, som er bedre egnet til simple objekter. Når man har med komplekse objekter at gøre, vil man gerne have styr på f.eks. `this` referencen, og *private* og *public* attributter.

I kodeoversigt 5.6 er der defineret et Model objekt, *product*. Det er et simpelt objekt med attributter og værdier. Model objekter vil man normal modtage fra serveren via AJAX kald. ViewModel objektet har en række `observable` attributter. En `observables` er et Knockout objekt, som underretter (Eng. notify) abonnenter om ændringer. Når en observables indhold ændres, opdateres automatisk det/de brugergrænsefladeelementer, der er bundet til dette `observable`. I eksemplet her var det ikke nødvendigt at definere ViewModel properties som `observables`. Hvis jeg havde defineret dem som almidelige JavaScript variable, havde jeg fået samme output - Knockout ville binde variablernes værdier til HTML `span` elemeterne, første gang filen blev indlæst. Men efterfølgende interaktion som vil resultere i, at variablernes værdier ændres (enten i kode eller UI, fx indtastning i et tekstfelt, som er bundet til variablen), ville ikke afspejle denne ændring i brugergrænsefladen, da almindelige JavaScript variable ingen måde har at underrette, at de er blevet ændret. Her kommer `observables` ind i billedet. De er Knockout objekter, der automatisk vil udstede notifikationer, når deres værdi ændres.
For helhedens skyld viser jeg resultatet af dette eksempel i figur 5.10 nedenfor. Jeg har brugt værktøjet *jsfiddle*, som er et kodedelingsværktøj, der kan bruges til at redigere, dele, eksekvere og debugge web kode inde i browseren. Jeg har brugt værktøjet meget hyppigt i forløbet indtil videre for at sætte mig ind i de forskellige web teknologier.

Figur 5.10: MVVM med Knockout eksemplet - jsfiddle output.

5.8 Opsummering

I dette kapitel præsenterede jeg et design af løsningen til prototypen. De forskellige softwarekomponenter blev identificeret, og teknologierner, server-side og klient-side udpeget til realisering af softwarekomponenterne. Systemet blev præsenteret i form af en logisk lagdelt arkitektur, hvor lagene blev beskrevet i detaljer. Der blev præsenteret et foreslag til brugergrænsefladestruktur og organisering af indhold i brugergrænsefladen. Jeg behandlede emnet Responsive Design og de forskellige strategier for at komme ind på det mobile marked (tablets, smartphones osv.). Kapitlet slutter med et kig på designmønstre i JavaScript - både med hensyn til organisering af software komponenter, men også for at opnå en bedre håndtering af JavaScript kode.

Kapitel 6

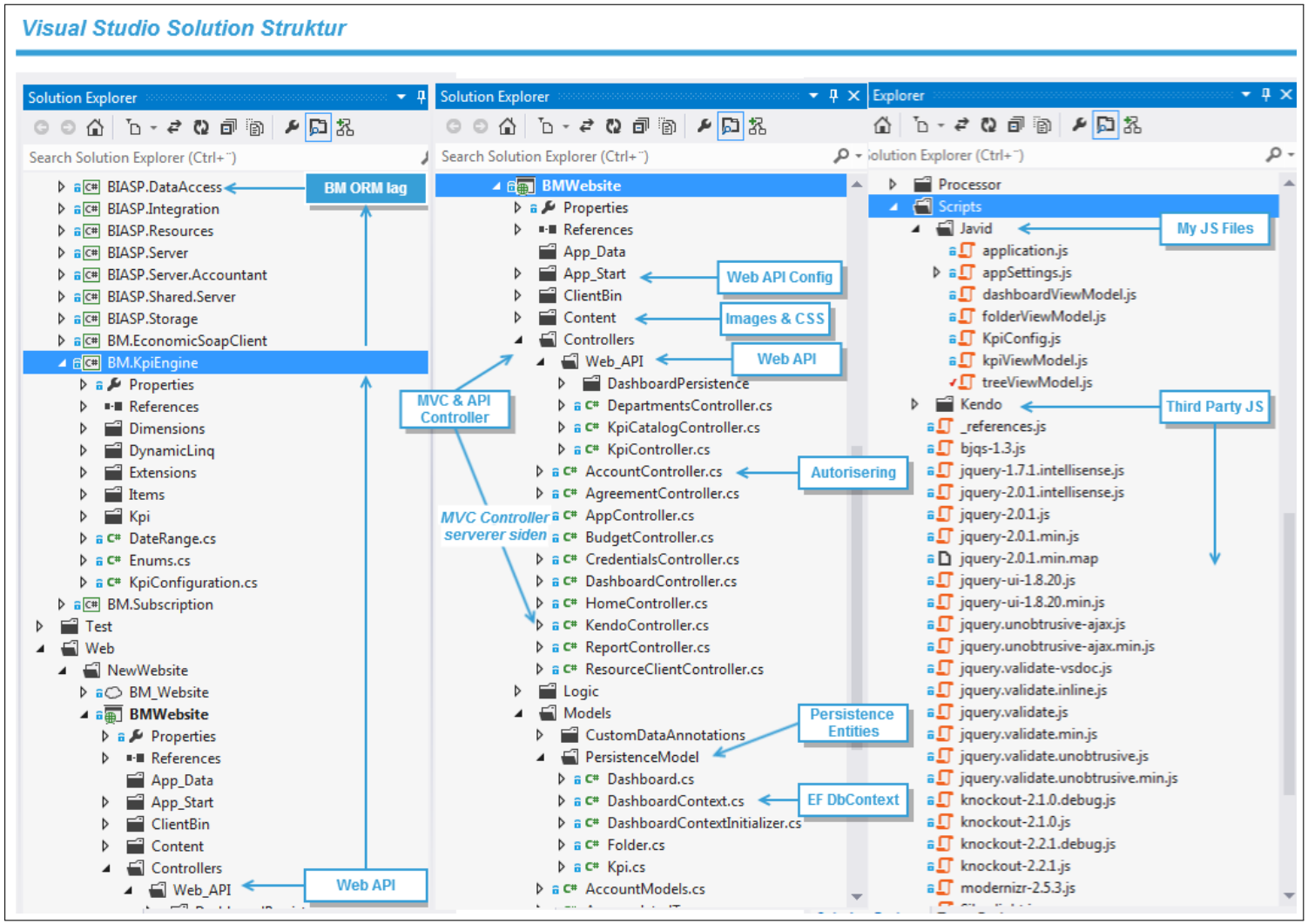
Implementering

Implementering betyder at programmere og forvandle design til code. Gennem implementeringsaktiviteter materialiseres iterativt, det der blev analyseret og designet. Det betyder ikke, at aktiviteten først startes, når alle krav er analyseret og bagefter designet. Jeg startede på implementeringsaktiviteten, da de første krav blev identificeret og den overordnede arkitektur bestemt gennem analyse og design. Senere i forløbet, når de fleste krav er identificeret, analyseret og designet, er hovedvægten lagt på implementerigsaktiviteten. Dette er i overensstemmelse med UPS iterativt udviklingsprocess.

Jeg vil i dette kapitel redegøre for implementeringsprocessen, hvorigennem prototypen blev udviklet og i den forbindelse komme ind på systemets forskellige elementer helt ned på kodeniveau.

6.1 Visual Studio Solution Struktur

Jeg vil starte med at give et hurtig overblik over de projekter, der udgør løsningen i Visual Studio. Jeg har fået udleveret hele Business Monitor codebase i form af Visual Studio Solution, hvor jeg har tilføjet nogle projekter til realisering af prototypen. Figur 6.1 illustrerer de relevante projekter og dele af dem, som er markeret med nogle pile og forklarende tekst. Her ses f.eks. projektet *BM.KpiEngine*, som bliver kontaktet af WEB API del af *BM.Website* projektet og som igen kontakter Business Monitors *DataAccess* laget for at fuldføre sine funktioner. *BM.Website* projektet har en `AppStart` folder, hvor jeg definerer forskellige konfigurationer vedrørende ASP.NET MVC og WEB API. Det er fx konfiguration af `Routes`, som definerer, når WEB API bliver spurgt om en tjeneste, hvordan skal forespørgslen navigeres til en bestemt Controller `Action` (metoder på MVC- og API Controllers), ud fra informationer i forespørgslen.



Figur 6.1: Visual Studio Solution Struktur.

En anden konfiguration, jeg har anvendt, er de nye tjenester fra ASP.NET Web Optimization framework, som er *bundling & minification*, der optimerer applikationens ydelse ved at kombinere flere scripts og styles ressourcerne til en enkel ressource, og derved får browseren til at lave færre HTTP forespørgsler.

Jeg har tilføjet et servicelag vha. ASP.NET WEB API, og som det kan ses, er dette lag en del af `BM.Wesite`, som er et ASP.NET MVC4 projekt. ASP.NET WEB API arbejder godt sammen med ASP.NET MVC og bruger det samme paradigme, men med fokus på HTTP. Som det er illustreret i figuren, er der en Controller folder, som indeholder både MVC Controllers, men også WEB API Controllers i en undermappe. API Controllers interagerer med `BM.KpiEngine` modulet for at hente og eksponere data, men den interagerer også med Entity Framework via andre Controllers (indeholdt i undermappen *DashboardPersistence*), for at eksponere og persistere data. Denne interaktion sker gennem klassen `DashboardContext`, som repræsenterer Entity Framework `Context`, som jeg har sat op for at understøtte persistens.

JavaScript kode, som vist i figur 6.1, er organiseret i mappen *Scripts*. Her har jeg lavet nogle undermapper til min egendefinerede JavaScript filer og tredjeparts biblioteker, som fx Kendo UI, some er et framework, jeg bruger til datavisualisering. Min egendefinerede JavaScript kode er fx mine ViewModels eller den AJAX kode, jeg har skrevet, for at kommunikere med WEB API fra klienten.

6.2 Server-side

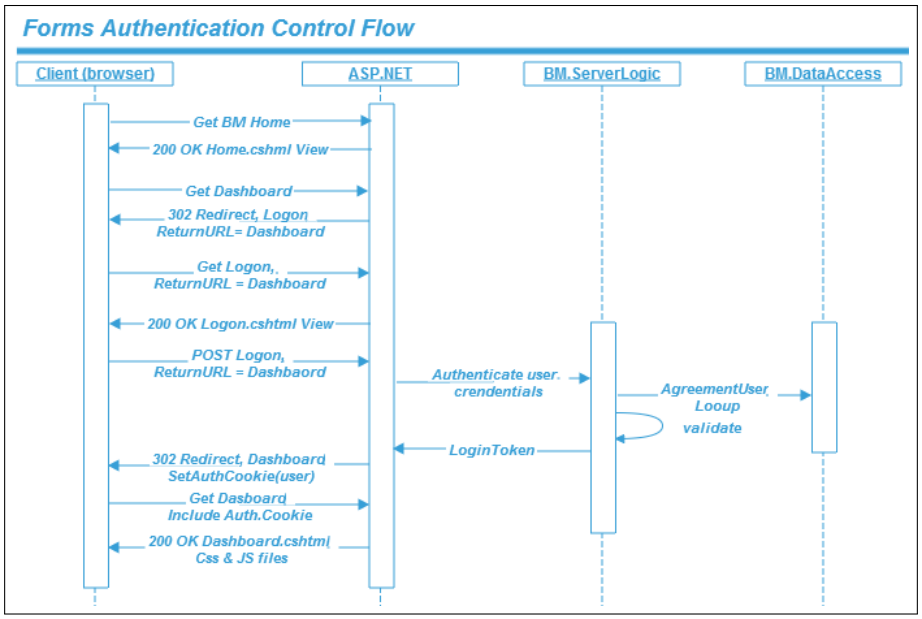
Jeg vil gennemgå implementering af systemet opdelt efter server-side-elementer og klientside-elementer. I dette afsnit kommer jeg ind på implementering af systemet mht. server-side-elementerne.

6.2.1 ASP.NET MVC4

Prototypen er en webapplikation, som en bruger interagerer med via sin browser. Så når en bruger navigerer hen til applikationen, sender brugerens browser en forespørgsel til serveren. Server-siden skal tage imod denne forespørgsel, reagere på den og sende et svar tilbage til brugerens browser. Det er en ASP.NET MVC 4 Controller, der tager imod forespørgslen, konstruerer et View og sender det til browseren. Her kan brugeren logge ind og komme ind på HTML Dashboard applikationen. Dvs. når brugeren er autentificeret, sender MVC Controlleren alle de filer, der er nødvendige for at køre HTML Dashboardet. Det er HTML og CSS filer, JavaScript filer, som udover egendefinerede filer, også inkluderer andre biblioteker og frameworks. Herefter er brugeren i stand til at bruge applikationens funktioner vha. JavaScript kode og kalde server-side tjenester via asynkrone AJAX kald. I det følgende vil jeg beskrive autentificeringsprocessen.

6.2.1.1 Autentificering

Den MVC Controller, der serverer Dashboard applikationen, hedder `KendoController`. Hvis jeg kan begrænse adgangen til denne Controller og dens metoder, og kræve at kun autentificerede bruger/forespørgsler har lov til at bruge denne Controller, har jeg opnået autentificering. Dette er præcis, hvad jeg kan opnå med `[Authorize]` attributten i ASP.NET MVC. Denne attribut kan jeg anvende på Controller niveau ved at dekorere Controlleren med attributten. MVC runtime vil nu opfange alle indkommende forespørgsler til enhver *Action* metode i denne Controller og sørge for, at adgangen er autentificeret; hvis ikke, bliver forespørgslen omdirigeret til login siden. Det er klassen `AccountController`, der står for autentificering. Her er defineret en `Logon` POST *Action* metode, som tager et modelobjekt som parameter. Modelobjektet bærer information fra formfelterne, som brugeren har indtastet. Her tjekkes de indtastede oplysninger hos `BM.ServerLogic`, om der findes en Business Monitor aftale med de indtastede oplysninger. I bekræftende fald, udstedes en `authentication ticket` for brugeren ved at kalde metoden `SetAuthCookie(userName)` på `FormsAuthentication` klassen. `authentication ticket` bliver vedligeholdt i en *Cookie*, så en autentificeret bruger ikke skal give sine credentials ved hver forespørgsel. Denne process hedder brugerautentificering. Et anden aspekt i brugerkontrol er brugerautorisationen, som angiver, hvorvidt en autentificeret bruger har rettigheder til at benytte en bestemt ressource i systemet. Denne har jeg ikke implementeret, men ville let kunne tilføjes.



Figur 6.2: Forms Authentication Control Flow.

Denne sekvens af brugerautentificering er afbildet i figur 6.2. Som det kan ses, starter sekvensen med at brugerens browser forespørger siden *Home.cshtml* fra webserveren. Da `Home` Controlleren ikke er har en begrænset adgang for bruger, returnerer ASP.NET *Home.cshtml* Viewet. Brugeren forespørger nu Dashboard-siden. Da denne kontroller kræver brugerautentificering, ser ASP.NET efter en *authentication cookie*. Når den ikke finder en *authentication cookie*, omdirigeres browseren til `Logon.cshtm` siden. Information om brugernes oprindelig forespørgsel gemmes i *Query string* med nøglen `ReturnURL`. Browseren forespørger `Logon.cshtml` siden, som webserveren så returnerer. Brugeren indtaster sine credintials og poster loginformen tilbage til serveren. `AccountController` vil nu overdrage brugervalideringen til *BM.ServerLogic*, som vil returnerer en GUID, hvis brugeren er valideret. `AccountController` opretter en *cookie*, som indeholder `forms authentication ticket`, som bliver sat for sessionen. Serveren omdirigerer browseren til den URL, som var gemt i *query string*s `returnURL` parameter. Herefter vil browseren forespørge Dashboard siden igen, og denne forespøgsel vil inkludere `forms authentication cookie`. Serveren vil nu returnere Dashboard.cshtml-siden, der også linker til andre ressourcer (CSS og JavaScript filer).

6.2.2 Web API

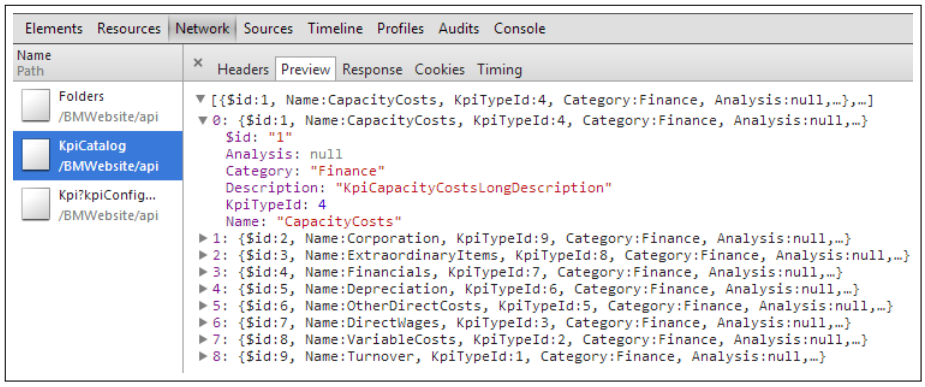
Servicelaget er implementeret med ASP.NET WEB API, som gør det lettere at udvikle REST services i .NET. Al kommunikation med klientlaget sker gennem dette lag. Der er datalag på serveren - både den eksisterende *BM.DataAccess* og laget repræsenteret med Entity Framework, som jeg har implementeret i forbindele med persistens. WEB API eksponerer disse data, pakket ind som .NET objekter, til klienten i form af JSON over HTTP, så klienten kan arbejde videre med det. F.eks. har jeg sat et API `endpoint` op, der returnerer alle KPI typer, der findes i systemet. Denne `endpoint` har URL *https://[host]/api/KpiCatalog*. Når klienten sender en HTTP forespørgsel til denne API `endpoint`, kan data, som WEB API returnerer, ses i browserens udviklingsværktøj, som vist i figur 6.3 nedenfor.

Det, vi kan se i figur 6.3, er, at der fra JavaScript kode er foretaget en AJAX forespørgsel til WEB API url *.../api/KpiCatalog*. Svaret fra WEB API er opfanget af Chromes udviklerværktøj, hvor det kan ses, at der er returneret et array af JSON objekter, der repræsenterer KPI typerne. Så ved at lave den slags simple RESTful HTTP kald til WEB API, kan jeg nemt og enkelt få mine data på klienten. Grundet denne enkelhed, og fordi det virker så godt med HTTP og JSON, er ASP.NET WEB API et godt valg for prototypen.

6.2.2.1 Data forespørgsel fra Web API

For at bygge et servicelag for applikationen har jeg overvejet følgende:

- HTTP forespørgslerne
- Routes, der skal dirigere forespørgslerne



Figur 6.3: Web API forespøgsel for KPI typerne.

- API Controllers, som skal svare på dem

Når der skal forespørges data fra API, laves en Web Request. Webserveren vil så forsøge at matche anmodningen til en `Route`, så anmodningen kan dirigeres til en Controller. Og API Controlleren vil så svare med noget data. Så det drejer sig om at lave en anmodning, matche en `Route` og få Controlleren til at svare. ASP.NET WEB API er ligesom ASP.NET MVC baseret på konventioner. Dvs. for en URL som denne *https://[host]/api/KpiCatalog* siger konventionen, at på webserveren skal vi kunne finde API Controller ved navn *KpiCatalog*. Anmodningen vil matche GET metoden i denne Controller.

Listing 6.1: WEB API Controller - HTTP GET

```
1  [Authorize]
2  public class KpiCatalogController : ApiController
3  {
4      // GET api/KpiCatalog
5      public IEnumerable<KpiCatalogItem> Get()
6      {
7          return KpiFactory.GetKpiList();
8      }
9  }
```

Kodeovesigt 6.1 viser, hvad denne URL matcher til på server siden. Navnet *KpiCatalog* matcher til den første del af klassenavnet. Og fordi det er en HTTP GET anmodning, matcher den til `Action` metoden `Get()`. GET metoden kunne hedder GET_hvadSomHelst().
Men hvordan dirigeres anmodningen ned til Controller? Her kommer Route ind i billedet. Da jeg satte WEB API op, blev der oprettet en Route vha. en standard template, som er *"api/{controller}/{id}"*, som siger, at en anmodning til WEB API har en URI, der starter med *api* efterfuglt af / [*Controller-navn*] / [*en id*]. Standard templaten definere id-delen som værende en valgfri parameter. Derfor kan WEB API frameworket dirigere anmodningen *https://[host]/api/KpiCatalog* til `KpiCatalogController` ved at at bruge standard Route template.

6.2.2.2 Opdatering af data

I sidste afsnit viste jeg, hvordan en anmodning kan sendes til WEB API baseret på HTTP GET og returnere data. Jeg kan også oprette eller modificere data ved at sende HTTP POST eller PUT anmodning til WEB API. Jeg har nogle WEB API Controllers, der håndterer CRUD funktioner på mine Dashboard entiter for at understøtte persistens. En af dem er `DashboardsController`, som ved brug af Entity Framework implementerer operationer på Dashboard entiteter. Kodeoversigten 6.2 neden for viser HTTP POST `Action` metoden fra denne Controller.

Listing 6.2: WEB API Controller - HTTP POST

```
// POST api/Dashboards
public HttpResponseMessage PostDashboard(Dashboard dashboard)
{
    if (ModelState.IsValid)
    {
        db.Dashboards.Add(dashboard);
        db.SaveChanges();

        HttpResponseMessage response = Request.CreateResponse(
                            HttpStatusCode.Created, dashboard);
        response.Headers.Location = new Uri(
            Url.Link("DefaultApi", new { id = dashboard.Id }));

        return response;
    }
    else
    {
        return Request.CreateErrorResponse(
                    HttpStatusCode.BadRequest, ModelState);
    }
}
```

Jeg kan fra klientsiden sende en HTTP POST anmodning til dette API `endpoint`. Anmodning vil inkludere noget data (`payload`), som repræsenterer et nyt dashboardobjekt. Som det kan ses, bruger jeg `DbContext` til at interagere med databasen og tilføje det nye dashboard, som bliver oprettet. Jeg returnerer en `HttpStatusCode.Created`, fordi der blev oprettet nyt dashboard, og jeg returnerer også dashboardobjektet, fordi det får tildelt et nyt id i forbindelse med oprettelse i database.

6.2.3 KpiEngine

KpiEngine, som nævnt tidligere, er det modul på server-siden, jeg måtte tilføje for at implementere noget af den KPI beregning- og filtreringslogik, der i dag er en del af Silverlight klienten og derfor ikke tilgængelig på serveren. Elementer i dette modul bliver forklaret i det følgende:

KPI

Jeg har valgt at implementere beregning og filtrering af nogle KPI typer, der er lettere at beregne, i forhold til andre mere komplicerede KPI typer.

Det er blevet til nogle af de KPI typer, som i Business Monitor er katagoriseret som `Finance` KPI'er. I *KpiEngine* er de repræsenteret ved nogle C# klasser, der alle arver fra en `abstract Kpi` klasse. Disse har nogle dimensioner, repræsenteret ved `Dimension` klasserne og en beregningsmetode. Hver KPI har derudover en type og beskrivelse. I beskrivelsesattributten gemmer jeg faktisk en `key`, som jeg kan bruge til at slå op i databasen, hvor den egentlige beskrivelse er gemt. I `Boards.cshtml` siden, som er siden, der vil blive sendt til klienten, bruger jeg ASP.NET `Razor` syntaks og slår op i databasen, hvor Business Monitor vedligeholder alle teksterne, henter beskrivelserne og gemmer dem i `localStorage`, som er en HTML5 `Web Storage` specifikation. På klienten har jeg nu disse beskrivelser tilgængelige gemt i browserens `localStorage`, som fungerer som et dictionary og gemmer *key/value* par. Når klienten forespørger listen af KPI typerne fra JavaScript gennem WEB API, overskriver jeg beskrivelsesattributten med den beskrivelse gemt i browserens `localStorage` ved at bruge den key-værdien gemt beskrivelsesattributten og slå op i `localStorage`.

Items
Beregninger sker på baggrund af nogle inputdata, repræsenteret ved `SourceItem` objekterne i *KpiEngine* modulet. Og når beregninger er foretaget, returneres en liste af typen `ChartSeriesItem`, som på klienten bliver brugt som input til datavisualiserings komponenten.

Dimensions
Selve beregninger af KPI'er sker i deres dimensioner, repræsenteret ved `Dimensions` klasserne. Hver dimensionklassen får sin dimensionskonfiguration, og på baggrund af denne konfiguration, indlæser, filtrerer og aggregerer dimensionen inputdata.

6.3 Klientside

Klient-siden blev bygget i overensstemmelse med designet præsenteret i Designkapitlet. Implementeringen af klientsiden består af at skrive HTML og CSS kode, der realiserer den brugergrænsefladestruktur foreslået i Design. Den består af at skrive den nødvendige JavaScript kode, der mapper designklassediagrammet fra Design kapitlet, til kode. Jeg har anvendt adskillelsesprincippet MVVM ved brug af Knockoutjs, til at fordele opgaver på de forskellige elementer i klienten. Jeg har brugt AJAX til asynkront, at sende og modtage data mellem server (WEB API) og klienten, hvor data på klienten modtages i JSON format. Jeg har benyttet JavaScript biblioteket jQuery til manipulering af HTML elementer, og til at udføre AJAX anmodninger. Kendo UI er et HTML5/JavaScript framework, som jeg har anvendt til datavisualisering og UI widgets. I det følgende vil jeg komme ind på nogle detaljer ved implementering af klientside-elementerne.

6.3.1 JavaScript View Models

JavaScript, som nævnt tidligere, giver mulighed for at skrive kode på flere forskellige måder. Det har været vigtigt for mig at undgå spaghettikode og en blanding af HTML og `<script>` tags forskellige steder i HTML dokumentet. Med sådan en fremgangsmåde har man at gøre med forskellige JavaScript funktioner, som alle er global funktioner og er svære at vedligeholde. Der er derfor store fordele ved at undgå spaghettikode og afkoble og indkapsle koden i objekter. Herved opnås at variabler og funktioner får sit eget scope, og det bliver nemmere at udvide og vedligeholde koden.

Jeg har nogle JavaScript objekter, som vist i klassediagrammet i Designkapitlet. Disse objekter indkapsler og eksponerer data og funktionalitet jævnfør MVVM. Jeg har brug for at kunne oprette flere instanser af disse objekter og kunne håndtere dem som objekter i objekt-orienteret sprog. Som jeg tidligere har nævnt, understøtter JavaScript ikke konceptet klasser, men JavaScript understøtter det, der hedder `constructor` funktioner. Ved at sætte `new` foran funktionen, kan jeg fortælle JavaScript, at jeg gerne vil have funktionen til at agere som en konstruktør og instantiere et nyt objekt med de medlemmer, som er defineret i denne funktion. Kodeoversigt 6.3 viser et udsnit af *FolderViewModel* `constructor` funktionen.

Listing 6.3: `Constructor` funktion - *FolderViewModel*.

```
function FolderViewModel(parent, settings) {
  var self = this;

  // Data
  self.id = ko.observable();
  self.title = ko.observable();
  self.dashboards = ko.observableArray();
  self.selectedDashboard = ko.observable();
  ...

  // Behaviour
  self.createNewDashboard = function () {
    // Oprette ny instans af DashboardViewModel
    var newDashboardVM = new DashboardViewModel(self, settings);

    ...
  };

  //Adds new dashboard to the list
  self.addNewDashboard = function () {
    // Add new dashboard to the end of the list
    self.dashboards.push(self.selectedDashboard());
  };
  ...
}
```

Som det kan ses, definerer jeg en variabel `self` i starten, som får tildelt værdien `this`. I JavaScript refererer `this` normalt til det objekt, som ejer funktionen. I global scope, når en funktion bliver kaldt, refererer `this` til det globale

objekt, som er `window`— topniveau objektet. I `constructor` funktionen refererer `this` til den instans, der bliver oprettet. Jeg bruger variablen `self` for at gemme referencen til det oprindelige `this`, som i `constructor` funktionen refererer til det objekt, der bliver oprettet og får tilknyttet alle variablerne og metoderne defineret i `constructor` funktionen. Det gør jeg fordi JavaScripts *keyword* `this` kan have forskellige kontekst i forskellige situationer.

Alle mine view models følger denne model— de bliver defineret som `constructor` funktioner, og inde i disse funktioner gemmer jeg først `this` referencen i en privat variable kaldet `self`. Denne variable bruger jeg så til at definere `constructor` objektets public variabler og metoder. Strukturen er, at i filen *application.js* definerer jeg en selvkaldende anonym funktion (Eng. self invoking function). Denne funktion kører automatisk, når siden besøges. Funktionen opretter en new instans af *TreeViewModel* vha. dens `constructor` funktion og kalder herefter dens `initialize` metode. Denne metode varetager de nødvendige AJAX kald og henter alle de nødvendige data for applikationen. Den anonyme funktion kalder herefter en vigitig Knockout funktion, som får denne *treeViewModel* instans som argument.

```
ko.applyBindings(treeViewModel, $("#container").get(0));
```

`ko.applyBinding` metoden aktiverer `Knockout` og forbinder min view model med view'et. Den tager mit view-model-objekt, sætter den som data kontekst for min HTML, hvor jeg her også har angivet, hvilket DOM element i min HTMLside, den skal bruge som root.

En anden måde at oprette objekter på i JavaScript, er `object literal` notationen, som jeg har anvendt i oprettelse af `KpiConfiguration` objektet inde i *KpiViewModel*. Det er en nem måde at oprette objekter på, som ikke inddrager kompleks logik, og når man ikke behøver at kunne oprette flere instanser af objektet. Notationen bruges meget i forbindelse med at sende flere valgfri argumenter til en funktion. Kendo UI bruger den meget i forbindelse med initialisering af widgets. En widget kan have mange konfigurationer, og man sender de kunfigurationer, man er interesseret i, i en `object literal` notation. Fx bruger jeg Kendos `DataSource` komponent til at indkapsle de data, der ligger til grund for en `kendo` datavisualiseringskomponent— fx en Chart widget. Kendo `DataSource` er en abstraktion for data, enten lokal (array af JavaScript objekter), eller fjerndata, som komponenten henter baseret på konfigurationer angivet i dens `transport` objekt. I kodeoversigt 6.4 nedenfor, har jeg først defineret et objekt (`dsConfig`) i `object literal` notation. Dette objekt definerer en enkel medlem— `transport`, som også er et objekt i `object literal` notation. Dette objekt definerer et medlem— `read`, som er et objekt med nogle medlemmer, osv. Denne `dsConfig` bruger jeg så som argument i initialisering af Kendo `DataSource` komponenten.

Listing 6.4: Initialisering af Kendo `DataSource komponent med konfigurationer i form af Object literal JavaScript objekt notation`.

```
// Et objekt der indeholder nogle konfigurationer
// til initialisering af Kendu DataSource komponent
var dsConfig = {
  transport : {
    read : {
      url : settings.kpiUrl,
      dataType : 'json',
      data: {
        kpiConfig: {...}
      }
    }
  }
}

// Oprette DataSource ved brug af konfigurationer
// defineret foroven
var ds = new kendo.data.DataSource(dsConfig);
```

6.3.2 HTML Siden - Boards.cshtml

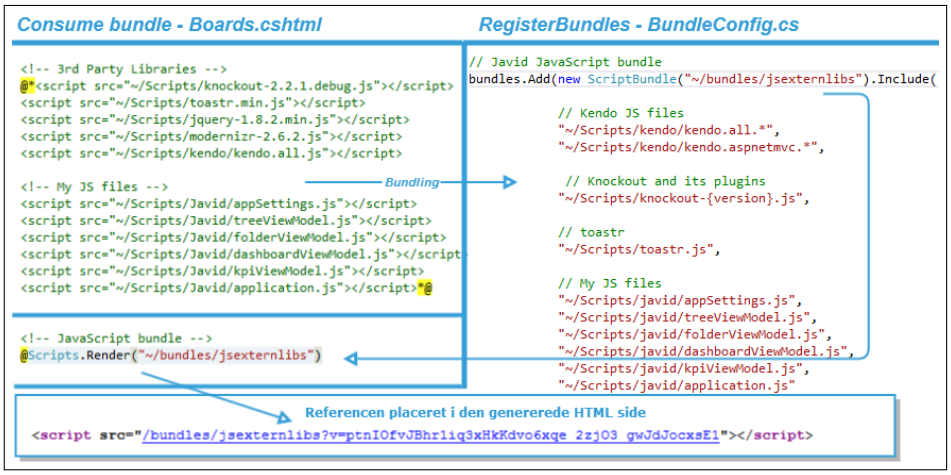
Klientsiden består af HTML CSS og JavaScript filer. JavaScript filerne repræsenterer både de filer, som jeg inkluderer fra tredjepartsbiblioteker og frameworks, men også dem jeg selv har skrevet, som faktisk gør, at applikationen kører. Nogle af disse tredjepartsbiblioteker har også CSS filer, som Kendo UI og toastr bibliotekerne. Jeg refererer til disse filer i min html side. Min pointe er, at der er mange filer, som browseren skal forespørge fra serveren, og nogle af disse tredjepartsbiblioteker fylder en del. Jeg har forsøgt at følge bedste praksis under udvikling af HTML og JavaScript kode og har derfor udnyttet muligheder i ASP.NET `Web Optimization` for at minimere antallet af web anmodninger og skrumpe filerne og reducerer derved krav til båndbrede og ventetid for brugeren.

6.3.2.1 Web Optimization features

Når en bruger navigerer til dashboardsiden, får `KendoController` denne anmodning, og hvis forespørgslen er autentificeret, serverer den `Boards.cshtml` siden til klienten. Det er en side, der bruger `Razor`[1] `view engine` til at generere HTML. Så alt `Razor` syntaks bliver konverteret til HTML, som så bliver modtaget på klienten. På server-siden drager jeg fordel af ASP.NETs Web Optimization `bundling & Minification` features. Så i stedet for at tage hver eneste JavaScript ressource som jeg vil bruge (framework, biblioteker og min egen JS kode) og placere en reference til hver af dem på siden, kan jeg i stedet lave en `bundle` og derved få browseren til at lave en enkel anmodning. På server-siden, i filen *BundleConfig.cs* placeret under mappen *AppStart*, har jeg defineret nogle

[1] `Razor` er en en markup syntaks for generering af HTML baseret på C# programmeringssprog. Det er `view engine`, som blev udgivet som en del af ASP.NET MVC3.

`bundles` til både CSS og JavaScript filer. Disse `bundles` kan så konsumeres på `Boards.cshtml` siden, som vist i figur 6.4 nedenfor.



Figur 6.4: At lave `bundle` og bruge den i *Boards.cshtml* siden.

Som vist i figur 6.4, tager jeg alle mine JavaScript filer, laver en `bundle`, som jeg giver en logisk sti (*" /bundles/jsexternlibs"*). Jeg tilføjer alle JavaScript filerne, som jeg vil have med i denne `bundle`, og jeg kan nu referere til den i `Boards.cshtml` side. Det, der også sker i denne process, er, at filerne bliver minificeret/komprimeret og får tildelt et versionsnummer. Versionsnummeret ændrer sig først, når filerne i en `bundle` ændres - hvis jeg f.eks. opdaterer en CSS eller JavaScript fil. Så det fungerer som versionskontrol. Opdateres en fil, sendes et nyt versionnummer. Browseren vil ikke sende en ny forespørgsel for en `bundle` næste gang, indtil den modtager et nyt versionsnummer. Som det kan ses nederest i figuren placeres i den genererede HTML side en enkel reference med unikt versionsnummer for ressourcerne i denne `bundel`.

Jeg har benyttet mig af disse optimeringsfeatures i ASP.NET Web Optimization til at registrere nogle bundles af JavaScript- og CSS filer og refererer til dem i min `Boards.cshtml` side. Der er lavet en `bundle` til Kendo UI framework, en til jQuery, Knockout og andre tredjeparts bibliotekerne. Og en til de JavaScript filer, jeg selv har skrevet.

6.3.3 MVVM, Knocktout og Data Binding

Data fås på klienten gennem asynkrone kald til WEB API via AJAX. Jeg præsenterer disse data i HTML vha. Knockout data binding, der bruger MVVM designmønstret. MVVM går ud på at dele koden i View (min HTML), ViewModel (min JavaScript) og Model (mine data). I det følgende vil jeg komme ind på, hvordan jeg eksponerer data og operationer gennem nogle ViewModels og binder dem til elementer i min HTML.

6.3.3.1 Bindning af en liste af detalje KPI'er

`Boards.cshtml` siden implementerer den UI struktur, jeg lagde frem i Design kapitlet. Den består af HTML markup, som indeholder Knockout binding-attributter, der hvor data skal præsenteres. Disse binding-attributter linker deklarativt, DOM elementer med Model properties, som er eksponeret via mine ViewModels. `Boards.cshtml` siden indeholder desuden markup, som jeg bruger som templates til fx popup vinduer, konfigurations menuer osv. Følgende kodeoversigt 6.5 viser et udsnit af HTML siden, hvor listen af detalje KPI'er bliver bundet til et *div* HTML element. Her kan det ses, at jeg har brugt `foreach` Knockout binding og en del andre - fx `text` binding, som jeg har gennemgået tidligere. Desuden kan det ses, hvordan `click`, `css` og `attr` bindings bliver brugt. Disse er allesammen eksempler på nogle indbyggede Knockout bindings. Jeg har også benyttet muligheden og defineret mine egne bindings, som vil blive gennemgået senere i kapitlet. Kodeoversigten viser, at HTML elementerne bliver bundet til properties og metoder, som er "synlige"for denne del af HTML dokumentet. Dvs. der er en ViewModel, som er blevet sat som kontekst for denne del.

Listing 6.5: Binder en liste af KPI'er vha. `foreach` binding.

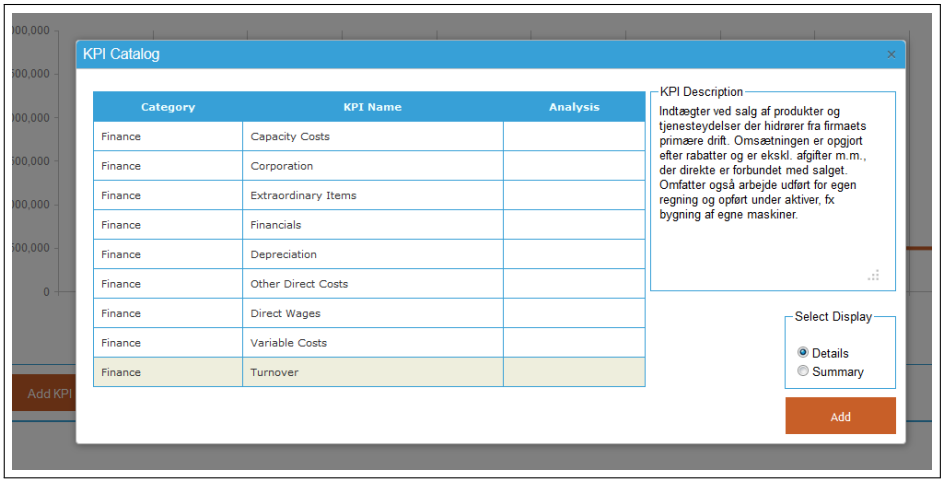
```
<!-- Details - DataContext er DashboardViewModel-->
<div id="kpiContent-details" data-bind="foreach: kpis">

  <!-- Chart/grid view -->
  <div class="tabstrip">
    <header data-bind="click: $parent.setCurrentKpi">
      <span data-bind="text: kpiName"></span>
      <div>
        <span data-bind="click: $parent.openKpiConfig"..></span>
        <span data-bind="click: toggleView,
                    css: { tableView: isTableView() }"..></span>
        <span data-bind="click: $parent.removeKpi,
                enable: $data === $parent.currentKpi()"..></span>
        <span data-bind="if: remove()">
          <span data-bind="initKendoWindow: { resizable: false..},
                template:{name: 'delete-template-kpi',
                        data: $parent }, openKenWindow: remove()">
          </span>
        </span>
        <span data-bind="click: saveChanges"..></span>
      </div>
    </header>
    <div data-bind="attr: { id: chartId }"></div>
    <div data-bind="attr: { id: gridId }" class=" hide grid"></div>
  </div>
</div>
```

Dette kode udsnit stammer fra en del af HTML siden, hvor konteksten er en *DasboardViewModel*. *DashboardViewModel* eksponerer nogle variabler og metoder til adminstrering af KPI'er. Der er fx en liste af *KpiViewModels*. Der er ligeledes eksponeret metoderne til fx at tilføje, fjerne og indstille et KPI. `kpis`, som er blevet brugt som værdi i `foreach` binding i kodeoversigten foroven, repræsenter en Knockout `observable` array af *KpiViewModels* objekter.

Den HTML kodeblock, der er defineret inde i `foreach` binding, bliver gentaget for hver *KpiViewModel* objekt, indeholdt i `kpis observable` array. Så hvis der er 10 elementer i `kpis observable` array, vil Knockout generere 10 blokke af `<div class="tabstrip> ... </div>` elementer. Inde i denne `foreach` binding, er datakonteksten det nuværende *KpiViewModel* element fra `kpis observable` array. Det er derfor jeg, i linje 7 i kodeoversigten, kan binde `<span>` elementets `text` binding til variablen `kpiName`, som er defineret i *KpiViewModel*. Hvis jeg vil hoppe tilbage til *DasboardViewModel* konteksten inde fra `foreach` binding, kan jeg bruge `$parent`, som er en Knockout variabel, der repræsentere ViewModel objektet et niveau højere i hierarkiet. På samme vis bruger jeg variablen `$data` til at referere view-model-objektet i den nuværende bindingkontekst. I kodeoversigten har jeg i linie 12 brugt `click` binding i et `<span>` element, som bliver bundet til `removeKpi` metoden i *DashboardViewModel*. Her har jeg også brugt en `enable` binding som vist i linie 13. I denne binding har jeg anvendt et udtryk, der evaluerer til `true` eller `false` og derved aktiverer eller deaktiverer `<span>` elementet. Således bestemmes om `click` event skal finde sted. Udtrykket, der evalueres her, er `$data === $parent.currentKpi()`. Ideen er kun at aktivere *slet*-ikonet (`<span> elementet`), hvis objektet i det nuværende binding-kontekst er det samme, som det jeg refererer til via `currentKpi` i *DashboardViewModel*, som er forældrekonteksten. Så hvad er *binding context*?

Binding Context er et objekt, der indeholder data, som man kan referere fra sine bindings. Når jeg bruger kontrol flow bindings, såsom `foreach, with, if`, bliver der oprettet en indlejret (Eng. nested) binding context, som refererer til det 'indlejrede' view model data.

6.3.3.2 KPI Katalog



Figur 6.5: KPI katalog modal vinduet.

Alle de financielle nøgletal, som prototypen understøtter, er dokumenteret i et KPI katalog. Det er på klienten implementeret ved et modal-vindue, der viser en liste af KPI'erne med nogle detaljer om dem, se figur 6.5 foroven. Når brugeren vælger et KPI ved at klikke på det, vises en beskrivelse af KPI'et. Dette vindue åbner, når brugeren vil tilføje et KPI, hvor han også har mulighed for at angive, om KPI'et skal tilføjes som et `detalje`- eller et `overbliks` KPI. Jeg henter alt data, der skal præsenteres i KPI kataloget, når siden indlæses første gang og gemmer dem i et `observableArray` i *TreeViewModel*:

```
self.kpiCatalogItems = ko.observableArray([]);
```

Jeg har i HTML præsenteret KPI kataloget i en tabel indkapslet i et `<div>` element. Med `Knockout with` binding har jeg sat dette elements datakontekst. Jeg har bundet elementet ved brug af `with` binding til det valgte dashboard (*DashboardViewModel*). Men hvis intet dashboard er valgt, eller med andre ord `selectedDashboard` er `null`, vil denne binding fejle, og Knockout vil stoppe med at processere alle efterfølgende bindings også. Derfor har jeg anvendt en anden binding-syntaks, som Knockout giver mulighed for. Det hedder *containerless control flow syntax* og er baseret på kommentar tags:

```
<!-- ko if: someExpressionGoesHere -->
  <li>I want to make this item present/absent dynamically</li>
<!-- /ko -->
```

Som vist ovenfor vil `<li>` elementet kun blive en del af HTML dokumentet, hvis udtrykket i `if` binding evaluerer til `true` eller noget, der ikke er `null` eller *undefined*. Så jeg har defineret `<div>` elementet, som præsenterer KPI kataloget inde i en *containerless if binding* med værdien `selectedFolder`, som vist i kodeoversigt 6.6 nedenfor. Her kan det ses, at `<div>` elementet er bundet til det valgte dashboard med `with` binding. Inde i `<table>` elementet binder jeg `<tbody>` til et `obervableArray` (`self.kpiCatalogItems`), som jeg nævnte før. Dette er defineret i *TreeViewModel*. *TreeViewModel* er sat som `root` view model. Derfor, for at binde `<tbody>` elementet med `self.kpiCatalogItems` arrayet, skal jeg kunne referere til *TreeViewModel* objektet. Med `$root` kan jeg referere til view-model-objektet i `root` kontekst.

Listing 6.6: Knockout Containerless binding syntax - KPI Katalog.

```
1  <!-- ko if: selectedFolder -->
2
3      <!-- Markup defineret i denne blok vil kun blive rendered -->
4      <!-- hvis selectedFolder ikke er null -->
5      <div id="catWindow" data-bind="with: selectedFolder().
           selectedDashboard()">
6          <table id="catTable">
7              <thead> ... </thead>
8              <tbody data-bind="foreach: $root.kpiCatalogItems">
9                  <!-- Her er Binding Context=kpiCatalogItem -->
10                 ...
11             </tbody>
```

```
12          </table>
13      </div>
14  <!-- /ko -->
```

6.3.3.3 Definition af Custom Binding Handlers

Bindings som `text, click, foreach` osv. er eksempler på indbyggede Knockout bindings. Jeg har defineret min egen Knockout binding handler, der initialiserer et Kendo window baseret på en template. Jeg bruger denne binding i et HTML element, fx et `<span>` element. Så med andre ord - jeg vil have, at `<span>` elementet bruger min egendefinerede Knockout binding og initialiserer et Kendo window baseret på nogle værdier, bindingen skal have som parameter. Jeg har defineret en anden binding handler, der åbner og lukker dette vindue.

Dette har jeg gjort ved at registrere min binding med Knockout således:

```
ko.bindingHandlers.initKendoWindow = {
    init: function(element, valueAccessor) {
        // Kaldes ved første anvendelse på elementet
        // Opsætning
    },

    update: function(element, valueAccessor) {
        // kaldes hver gang den tilknyttede
        // observable ændrer værdi
    }
};
```

Så det, jeg har gjort, er at udvide Knockouts `bindingHandlers` objektet med mit binding-navn, som fx `initKendoWindow` og implementeret `init` og `update` funktionerne. `init` funktionen vil køre første gang `initKendoWindow` binding bliver anvendt. Så når jeg anvender bindingen på et HTML element, kører `init` funktionen. `update` funktionen kører efter `init`, og hver gang den tilknyttede `obsevable` ændrer værdi. Disse funktioner tager nogle parametre, hvor nogle er valgfri. Jeg har kun brugt to af dem. Den første parameter er *element*, som repræsenterer det HTML element, der bliver bundet. Jeg bruger dette element til at initialisere et Kendo window widget. Kendo UI widgets bliver initialiseret fra eksisterende HTML elementer[2]. Kodeoversigt 6.7 nedenfor viser implementering `initKendoWindow` custom binding handler.

initKendoWindow binding

[2] `http://docs.kendoui.com/getting-started/widgets`

Listing 6.7: Definition af `initKendoWindow` custom binding handler.

```
1  //custom binding to initialize a Kendo UI Window
2  ko.bindingHandlers.initKendoWindow = {
3    init: function (element, valueAccessor) {
4      var options = ko.utils.unwrapObservable(valueAccessor()) || {};
5
6      //handle disposal
7      ko.utils.domNodeDisposal.addDisposeCallback(element,
8                                                  function () {
9        $(element).data("kendoWindow").destroy();
10     });
11
12     if (!$(element).data("kendoWindow"))
13       $(element).kendoWindow(options);
14       $(".k-window").css({ "font-family": "sans-serif" });
15       $(element).parent().addClass("catWindow");
16     }
17 };
```

Det første parameter til `init` funktionen, repræsenteret det HTML element, som jeg bruger min binding på. Så hvis jeg bruger `initKendoWindow` bindingen i et `<span>` element, er det dette `<span>` element, som første parameteren repræsenterer. Som det kan ses i kodeoversigten foroven, har jeg kun defineret `init` metoden i denne custom binding, da jeg vil blot initialisere et Kendo window widget. Jeg har tilføjet muligheden for at `dispose` vinduet på Kendo måden. Herefter initialiserer jeg vinduet, hvis ikke det allerede er initialseret, og giver det nogle CSS styles, så det matcher skriften og farverne, jeg bruger i applikationen. Den anden parameter (*valueAccessor*) til `init` metoden repræsenterer de værdier, som er givet til denne binding. I kodeoversigten 6.5 fra forrige afsnit, kan det ses i linje 15, at jeg bruger denne binding i et `<span>` element. Værdierne, som jeg sender til denne binding, er pakket i et JavaScript objekt[3]. Jeg har ikke vist alle værdierne i kodeoversigt 6.5, men det er nogle værdier, som jeg vil initialisere Kendo vinduet med. Disse værdier bliver repræsenteret i *valueAccessor* parameteren i `init` og `update` metoderne.

openKendoWindow binding

Den anden binding handler jeg har defineret, har til formål at åbne eller lukke det initialiserede vindue. De to bindings, jeg har defineret, bliver brugt i forbindelse med hinanden, så `<span>` elementet bruger både `initKendoWindow` binding og `openKendoWindow` binding, som vist i kodeoversigten 6.5 fra forrige afsnit. Som det kan ses, får denne binding et `observable` objekt (`remove`) som bindingens værdi. Denne bliver repræsenteret i *valueAccessor* parameteren i `update` metoden vist i kodeoversigt 6.8 nedenfor. Baseret på denne `observable` vil Kendo vinduet åbne, hvis `observable` er defineret og ellers lukke.

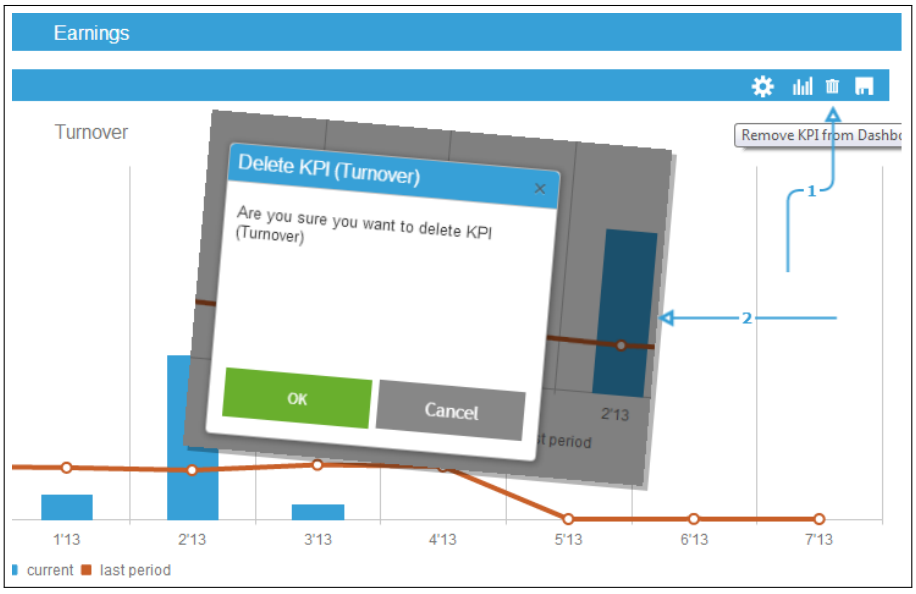
Listing 6.8: Definition af `openKendoWindow` custom binding handler.

```
1  //custom binding handler that opens/closes the window
2  ko.bindingHandlers.openKenWindow = {
```

[3]Denne format af JavaScript objektet hedder Object Literal, som jeg nævnte i Design kapitlet

```
3       update: function (element, valueAccessor) {
4           var value = ko.utils.unwrapObservable(valueAccessor());
5           if (value) {
6               $(element).data("kendoWindow").center().open();
7           } else {
8               $(element).data("kendoWindow").close();
9           }
10      }
11  };
```

`openKendoWindow` binding definerer kun `update` metoden, da jeg ved, at Kendo vinduet (på tidspunktet, hvor denne binding er aktiv) allerede er initialiseret i `initKendoWindow` binding, og de to bindings som nævnt før bliver brugt i forbindelse med hinanden. Disse to custom bindings har jeg defineret nederst i filen *folderViewModel.js*. Figur 6.6 herunder viser resultatet af de custom bindings, jeg har defineret. Når brugeren vælger at slette et KPI ved at klikke på `slet` ikonet, er det mine custom bindings, der sørger for, at et Kendo vindue blive initialiseret og popper op på skærmen og bagefter lukker ned igen, når brugeren har taget stilling til sletningen. Jeg bruger også disse custom bindings i forbindelse med sletningen af dashboard og folders i applikationen. Vinduet bliver konstrueret vha. nogle HTML templates, defineret nederst på `Boards.cshtml` siden. Anvendelsen af de to bindings er vist i kodeoversigt 6.9. Her er det gjort tydeligt, hvilke bindings `<span>` elementet anvender. Elementet bruger `initKendoWindow`, `openKendoWindow` og `template` bindings, som i kodeoversigten er vist sammen med deres værdier.



Figur 6.6: Resultatet af custom bindings.

Listing 6.9: Anvendelse af `initKendoWindow` og `openKendoWindow` custom bindings.

```
<div data-bind="if: remove()">
  <span data-bind="initKendoWindow:
                    { resizable: false,
                      visible: false,
                      modal: true,
                      draggable: false,
                      title: 'Delete Dashboard ('+ title() +')',
                      width: '270px', height: '200px'
                    },
                  template:
                    { name: 'delete-template',
                      data: $parent
                    },
                  openKenWindow: remove()" >
  </span>
</div>
```

`initKendoWindow` binding har et objekt som sin værdi. Objektet består af en række key/value par, som angiver nogle konfigurationer, som Kendo vinduet skal initialiseres med. Den næste er en `template` binding. `template` er en indbygget Knockout binding og bruges til at fylde det tilknyttede DOM element med en template. Her kan det ses, at det er en template med navnet '*delete-template*', som jeg har defineret nederest på `Boards.cshtml` siden. Jeg har også defineret i `template` bindingen, hvad dens data kontekst skal være ved at definere dens `data` parameter. Jeg sætter den til view model objektet i forældrekonteksten, repræsenteret ved `$parent` variabel, som her er *DashboardViewModel*. Det er fordi *delete-template* har nogle knapper, som jeg binder til nogle metoder i *DashboardViewModel*. Den tredje binding er den omtalte `openKendoWindow` custom binding. Dens værdi er et objekt repræsenteret ved `remove()`. Hvis objektet er defineret, åbnes vinduet, ellers lukkes det, som angivet i dens bindinghandlers `update` metode beskrevet tidligere.

Så det der sker i dette `<span>` element, når de tre bindings bliver anvendt, er at Kendo vinduet bliver initialiseret, en template bliver genereret angivet ved *delete-template*, og det initialiserede vindue popper op på skærmen med det indhold, som `template` binding genererede.

Det, der får disse `bindings` til at træde i kraft og blive aktiveret, er `binding`-udtrykket defineret i `<div>` elementet, der er defineret udenom det omtale `<span>` element. I kodeoversigt 6.9 kan det ses, at `<div>` elementet anvender en `if-binding`. Hvis udtrykket i denne `if-binding` evaluerer til `true`, vil `<span>` blive elementet 'betragtet' og alle dets `bindings` bliver anvendt. Med andre ord—det, der forhindrer et Kendo vindue i at blive initialiseret og poppe op på skærmen, er udtrykket `remove()` angivet i `if-binding` inde i `<div>` elementet. Når brugeren trykker på slet-ikonet, får denne `observable` variabel (`remove()`) tildelt en værdi i den pågældende viewmodel-instans. Når valget er truffet (`delete` eller `cancel` er klikket i popup vinduet), sættes `remove()` til `null` igen, som bevirker min `openKendWindow-binding` og lukker vinduet.

6.3.4 Datavisualisering

Et af punkterne under afsnittet Problem definition i Introduktion kapitlet er, at prototypen, ligesom den eksisterende Silverlight løsning, skal kunne repræsentere- og visualisere nøgletal i et dashboard. I Teknologi Analyse kapitlet, fandt jeg frem til, at jeg vil bruge Kendo UI[4] frameworket til dette formål. Kendi UI er baseret på jQuery. Alle dens widgets er eksponeret som jQuery plugins, initialiseres fra eksisterende HTML elementer og konfigureres ved at sende et konfigurationsobjekt som argumment til det pågældende jQuery pluging. For at visualisere data bruger jeg dens `Chart` widgets. En `Chart` widget skal have nolge data, som skal visualiseres. En `chart`s datapunkter kan specificeres som nogle series definitioner. Jeg modtager datapunterne som resultat af nogle beregninger fra *KpiEngine* modulet. Jeg kan modtage disse datapunkter (listen af `ChartSeriesItem`) ved at sende en anmodning til WEB API url *https://bmjavid.cloudapp.net/api/Kpi*. Denne AJAX anmodning skal så bære KPI konfigurationsdata, som skal sendes til serveren, og som beregningerne foretages på baggrund af. Den modtagne liste af `ChartSeriesItem` kan jeg binde til en `Chart` widget ved at konfigurere dens `series` array.

6.3.4.1 Oprettelse af Chart

Kendo UI har en `DataSource` komponent, der gør håndtering af data nemmere. Jeg har brugt denne komponent til at binde `Chart` widgeten til fjerndata. Dvs. denne komponent foretager AJAX anmodning til WEB API, når det er nødvendigt. Jeg har konfigureret `DataSource` komponenten med den WEB API `endpoint` den skal kontakte for at hente seriesdata, og jeg har givet den et kpi konfigurartionsobjekt, som skal sendes til serveren. Kpi konfigurationsobjektet indeholder information om hvilken nøgletal, der ønskes beregnet, informationer om dimensionerne og den valgte dimention. Disse informationer fås fra brugergrænsefladen vha. nogle Knockout `observables`, som er bundet til brugergrænseflade elementerne. Når brugeren vælger at beregne et nøgletal, ved at klikke *Calculate*, bliver kpi konfigurationsobjektet konstrueret, og `createChart()` metoden kaldt, som får konfigurationsobjektet som argument. I `createChart()` metoden oprettes en `DataSource` komponent, som konfigureres med ting som, hvor den skal læse data fra, hvilken format den skal forvente at modtage data i og hvilke data den skal have med til serveren.

Listing 6.10: *KpiViewModel* - visualisering.

```
1 function KpiViewModel(parent, settings) {
2   var self = this;
3   ...
4
5   self.calculate = function () {
6     var kpiConfig = { // Kpi konfiguration
7       kpiId: self.kpiTypeId(),
8       TimeDimensionConfig: {
9         FirstDate: self.firstDate().toDateString(),
```

[4] http://www.kendoui.com/

```
10          LastDate: self.lastDate().toDateString(),
11          Representation: self.timeDimensionRepresentation()
12        },
13        DepartmentDimensionConfig: {
14          DepartmentIds: self.selectedDepartments()
15        },
16        SelectedDimension: self.selectedDimension()
17      };
18
19      self.createChart(kpiConfig); // pass kpiConfig
20    }
21
22  ...
23
24    self.createChart = function (config) {
25      self.dataSource = new kendo.data.DataSource({
26        transport: {
27          read: {
28            url: settings.kpiUrl,
29            dataType: "json",
30            data: {
31              kpiConfig: config
32            }
33          }
34        }
35      });
36
37      chartView(); // Creates the Chart
38    }
39
40  }
```

Herefter kaldes metoden `chartView()`, som laver en Kendo `Chart` widget, som det fremgår af kodeoversigt 6.10 foroven.
`DataSource` komponenten er brugt til at binde `Chart` widgeten til fjerndata ved at sætte dens `dataSource` property til mit `dataSource` objekt oprettet tidligere. Nu kan jeg konfigurere `Chart` widgetens `series` array og binde series objekterne til data repræsenteret ved `dataSource`.

En Kendo UI widget, som tidligere nævnt, oprettes ved at vælge et eksisterende HTML element vha. jQuery `selectors`, og kalde den pågældende jQuery plugin for den ønskede widget. Så for at oprette en `Chart` kaldes

```
self.myChart = $(self.chartID).kendoChart({
  // konfigurationer...
  dataSource: self.dataSource,
  series: [
      {name: 'current', field: 'Amount',..},
      {name: 'last period', field: 'LastPeriodAmount',..}
  ]
  ...
```

```
}).data("kendoChart");
```

Her kan det ses, at jeg opretter en `Chart` widget og konfigurerer den. Som det kan ses, bliver dens `dataSource` property sat til den `dataSource`, jeg definerede tidligere, og derfor kan jeg nu i `series` objekterne binde til felterne i mine data, fx [`field: 'Amount'`], som binder series `field` property til feltet `Amount` i min data. `Amount` er et felt i `ChartSeriesItem`, som serveren returnerer. For at få en reference til `Chart` instansen, bruger jeg jQuerys `data` metoden og spørge efter widgeten som vist foroven.

Jeg nævnte tidligere, at Kendo widgets oprettes ved at vælge eksisterende HTML elementer. Men den markup, hvor en ny kpi skal visualiseres, eksisterer ikke på forhånd— da KPI'er bliver tilføjet dynamisk. Hvordan kan jeg så finde det element via jQuery, som skal repræsentere den nye `Chart` widget? Jeg kan ikke vælge elementet vha. `id`, da jeg ikke ved, hvilken markup og hvilket element. Svaret er vha. Knockout— når brugeren vælger at oprette nyt KPI, opretter jeg en ny instans af *KpiViewModel* og tilføjer den til listen af KPI'er. Denne liste er bundet i viewet til et HTML element ved brug af `foreach` binding som for hver *KpiViewModel* instans i listen, genererer noget markup, defineret i `foreach` blokken. Nu ved jeg hvilken markup, så det næste er at kunne identificere det HTML element i denne markup, der skal bruges til oprettelsen af `Chart` widget, så jeg kan referere til den vha. jQuery. Det gør jeg ved at definere et element, hvis `id` er bundet til en `observable` property i *kpiViewModel* instansen, der er datakonteksten for denne markup ved brug af `attr` binding:

```
<div data-bind="attr: { id: chartId }"></div>
```

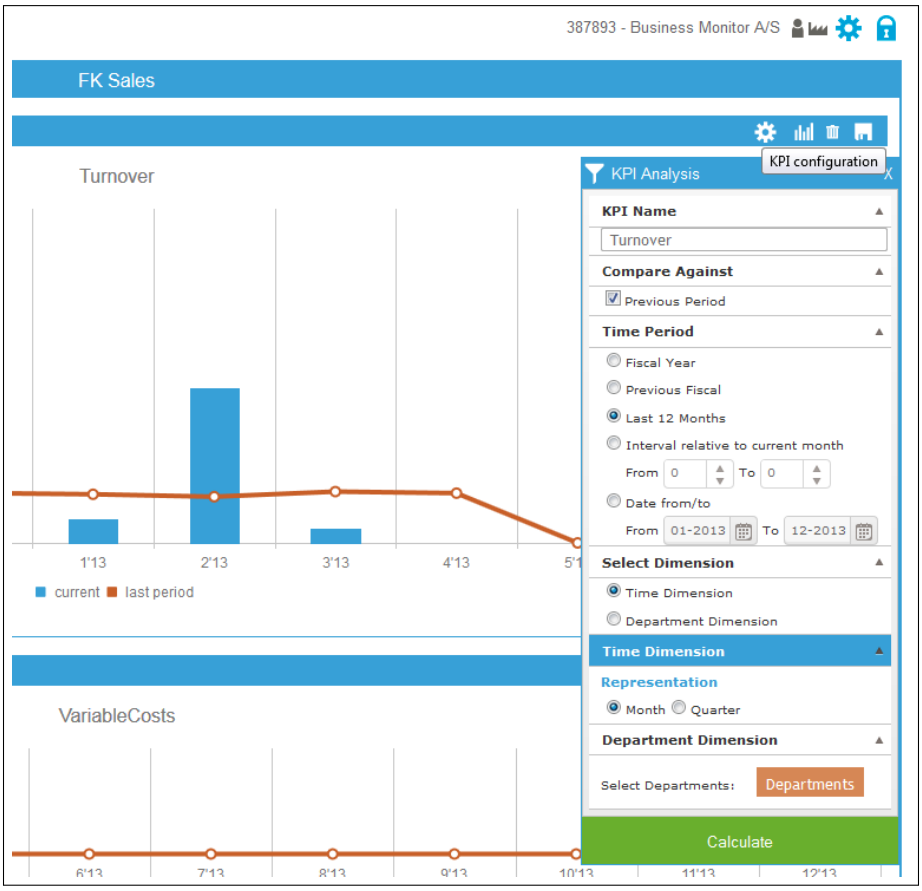
`chartId` repræsenterer et timestamp, og på tidspunktet denne binding bliver anvendt, vil elementets `id` have et timestampværdi. Værdien lever så længe denne viewmodel-instans lever. På den måde vil den være unik, og jeg kan referere til det med jQuery for at oprette grafen, som tidligere vist:

```
self.myChart = $(self.chartId).kendoChart({ ...
```

6.3.5 KPI konfiguration

KPI konfigurationer er de analyseindstillinger, der er implementeret i prototypen. Jeg har valgt at repræsentere alle analyseindstillingerne i en menu i brugergrænsefladen. Menuen er skjult til at starte med og kan foldes ud ved at klikke på tandhjul-ikonet tilknyttet det ønskede KPI. Menuen afspejler de valgte indstillinger for det valgte KPI. Dvs. den markup, der repræsenteret menuen, bruges

som en template og har det valgte KPI som sin datakontekst. Jeg repositionerer vha. CSS menuen og sætter dens datakontekst, afhængig af hvilket KPI der klikkes ved— se figur 6.7 nedenfor.



Figur 6.7: KPI konfigurations menu.

De ændringer, man foretager på et KPI via analysemenuen, afspejles først i grafen, når man har klikket *Calculate*, og persisteres først, når man har klikket på `save` ikonet tilknyttet KPIet. Dette er i modsætning til andre interaktioner med applikationen, som udløser umiddelbare persistens; fx når et nyt KPI oprettes, et dashboard oprettes eller slettes osv. Som det ses i figur 6.7, har jeg implementeret to dimensioner; `tidsdimension` og `afdelingsdimension`. Når afdelingsdimension er valgt, bliver den orange knap aktiv. Ved at klikke på den, åbnes et popup vindue, hvor de afdelinger man har i virksomheden, vises i en `multiselect` widget. Her kan vælges eller fravælges de afdelinger, man gerne vil se data for.

Alle beregningerne sker på baggrund af den valgte tidsperiode. For den valgte tidsperiode udregnes i *KpiViewModel* nogle datoer; `firstDate` og `lastDate`. Disse er i viewmodel repræsenteret ved en særlig `Knockout observable`, kaldet `Computed Observables`. De er funktioner, som er afhængige af en eller flere andre `observables`. Jeg har fx følgende `observable` i min *KpiViewModel*, som er bundet til `radio` knapperne i `Time Period` del af analysemenuen i brugergrænsefladen. Den repræsenterer et regnskabsår eller et andet interval, brugeren angiver for at se data for.

```
// Fiscal year - default is FiscalYear
self.period = ko.observable("currentFiscalYear");
```

`firstDate` og `lastDate` udregenes på baggrund af denne `observable`. Så hvis `period` har en værdi af `'currentFiscalYear'` (indeværende regnskabsår), som er default valgt, sættes `firstDate` til 1. Januar indeværende år, og `lastDate` sættes til 31. December indeværende år. Hver gang en anden `radio` knap vælges, reberegnes `firstDate` og `lastDate`, som resultat af ændringen i `period observable`.

```
// First date computed property
self.firstDate = ko.computed(function () {
  var period = self.period(),
      date = new Date();

  if(period === 'interval') {
    // Get the related values from UI
    ...
  }
  elseif...

  return date;
}
```

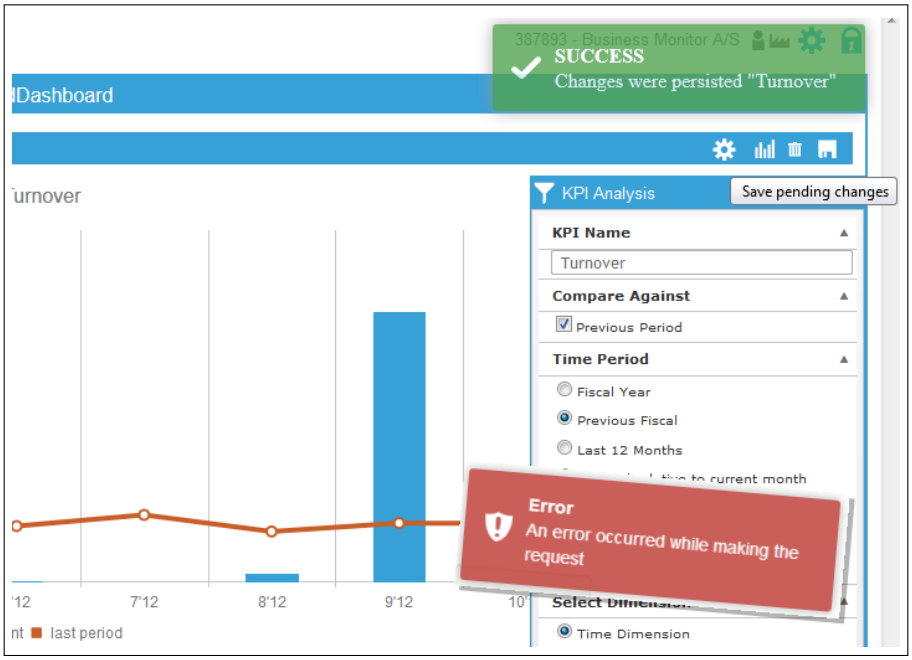
Som det kan ses, defineres `firstDate` som en `ko.computed(callback) observable`, og den gives en `callback` funktion, som kører hver gang `period` opdateres.

6.3.6 Feedback til bruger

I en applikation som denne, hvor der foretages mange asynkrone kald i baggrunden, er det vigtigt, at brugeren hele tiden er med i, hvad der sker og får besked om succes eller fejl i kommunikationen med serveren. Hvis brugeren foretager

en ændring og trykker `save` for at persistere ændringen, er det en god brugeroplevelse, at der kommer en hurtig feedback fra systemet, om tingene gik som forventet, eller der opstod en fejl.

Jeg har anvendt et lille JavaScriptbibliotek kaldet toastr.js, som giver simpel feedback om operationer i et lille popup. Den bliver vist et kort øjeblik, fylder kun så meget som beskedens størrelse, og siden forbliver synlig og interaktiv. Som det fremgår af figur 6.8, vises en lille grøn, gennemsigtig popup notifikation i toppen, efter jeg har trykt `gem`. Når en AJAX anmodning lykkes, vælger jeg at sende en `toastr.success(...)` notifikation. Hvis den fejler, sender jeg en `toastr.error(...)` notifikation. Jeg har placeret et udklip af fejlbeskeden i figur 6.8 nedenfor.



Figur 6.8: Feedback til brugeren via toast beskeder.

Når brugeren logger ind, skal alle hans data indlæses— det er alle hans dashboards, KPI'er og deres konfigurationer, samt hvilke folder de forskellige dashbords tilhører. Mens disse data anmodes fra serveren og grænsefladen forberedes, sender jeg en `toastr.info(...)` notifikation om, at systemet er ved at indlæse data.

6.4 Opsummering

I dette kapitel har jeg belyst nogle detaljer omkring implementering af prototypen. Jeg startede med at vise `Solution` struktur i visual studio, om hvordan de forskellige projekter er organiseret, og hvad de består af. Forklaringen af nogle detaljer omkring implementering er delt op i to sektioner— en der berører implementering af server-side-komponenter, og en om klientsidekomponenterne.

ASP.NET MVC4 blev anvendt til at serverer HTML, CSS og JavaScript til klienten. ASP.NET Web Optimization features blev anvendt til at bundle og minificere JavaScript og CSS, og derved få browseren til at sende færre forespørgsler samt en række andre optimeringer som beskrevet. ASP.NET Forms Authentication blev anvendt til bruger autentificering. Der blev opstillet en kommunikationsgrænseflade mellem klienten og serveren vha. AP.NET WEB API. Jeg tilføjede et nyt modul på serveren, der håndterer KPI'er og deres beregninger. Strukturen kan let udvides til at understøtte flere KPI'er uden det kræver ændring i andre dele af systemet.

Klientsiden blev udviklet med anvendelse adskillelsesprincippet, hvor ansvaret er fordelt på forskellige elementer en grænsefladen. Knockout JavaScript biblioteket har spillet en vigtig rolle i den forbindelse. Datavisualiseringer blev understøttet af JavaScript frameworket Kendo UI, som er jQuery baseret HTML5/JavaScript framework med stort udvalg af UI- og datavisualiseringswidgets.

Kapitel 7

Test

Der er ingen tvivl om at test er en vigitig process under udvikling af software, idet det er vigtigt at sikre at den skrevne kode eksekverer korrekt, og at programmet virker som forventet. Jeg har løbende testet systemets data og adfærd vha. browserens udviklerværktøj, som har været mit vigitigste testværktøj under udviklingen af prototypen, idet jeg mest har været optaget af udvikling på klientsiden. `Chrome`s udvilkerværktøj, som jeg har brugt, giver gode muligheder for JavaScript `debugging`— inklusiv mulighed for at sætte `breakpoints` og trinvis eksekvering af kode, hvor varaiblernes indhold og objekternes tilstand kan iagttages.

White-Box og Black-Box testing
`White-Box` test og `Black-Box` test er de to grundlæggende teknikker til software test. Der er flere typer tests, der inddrager disse teknikker. `unit` test, fx er en type test, der bruger `White-Box` teknikker. Her skriver en tester (som ofte er udvikleren) testkode, som verificerer at den implementerede kode gør, som tilsigtet, helt nede på kodeniveau. Funktionelle tests, derimod, er en type test, der bruger `Black-Box` teknikker, og undersøger systemet set fra et brugerperspektiv. Her skrives testcases baseret på kravspecifikationen. Jeg har udført disse to typer tests, for at teste prototypen, som beskrevet i de følgende afsnit.
Der er taget en `White-Box` tilgang mht. test af systemets WEB API. Med sådan en tilgang testes de interne strukturer af programmet— testcases udarbejdes med indsigt i systemets interne opbygning, og også programmeringsevner. Dette er i modsætning til `Black-Box` test, som tester systemet funktionaliteter eksponeret til brugeren. `White-Box` test anvendes normalt på `unit` niveau, som er det stykke software komponent, der testes. Der findes, heldigvis, automatiserede unit-test værktøjer— QUnit er jQuerys unit test framework til test af JavaScript kode.

7.1 Test af Web API Anmodninger

Med hensyn til WEB API er det vigtigt at sikre, at alle WEB API anmodninger bliver testet, og sikre at de agerer som forventet. Her vil jeg teste følgende:

- At sikre at alle API `endpoints` eksisterer, og svarer på en anmodning.
- At sikre at GET anmodninger returnerer det forventede data.
- At sikre at alle mine CUD[1] operationer virker.

Jeg vil udfører disse tests vha. værktøjet Qunit[2], som er et JavaScript bibliotek beregnet til test af JavaScript kode.

7.1.1 Web API Endpoints Tests

I dette afsnit vil jeg teste om alle mine WEB API endpoints kan blive anmodet med success. Jeg bekymrer mig ikke om jeg får det forventede resultat— jeg er blot interesseret i at WEB API URLs giver et svar, når en AJAX anmodning bliver sendt, og at et eller andet data returneres.

Jeg har skrevet nogle QUnit tests baseret på dens `ok() assertion`, som har denne signatur:

```
ok( state, message )
```

`state` angiver et udtryk, der evaluerer til `true` eller `false`. `message` er en beskriverlse af `assertion`. For at køre QUnit test, skrives testen i JavaScript fil (eller `<script>` tag i HTML filen), som indkluderes i en HTML side oprettet til formålet. I HTML siden refereres derudover, til QUnit CSS- og JavaScript filer. Jeg har oprettet en mappe ved navn `Test` i Web projektet *BMWebsite* i Visual Studio `Solution`. Her har jeg oprettet en undermappe 'QUnit' med QUnit CSS- og JavaScript filer, og en anden mappe 'web api', som indeholder alle mine testfiler (JavaScript- og HTML) til test af WEB API.

7.1.1.1 Testen

Jeg vil teste følgende:

- Om WEB API endpoints kan forespørges med success, ved at sende HTTP GET `requests` til hver af dem, foretaget via AJAX.
- Om alle disse GET `requests` resulterer i nogle data, som WEB API URL'erne returnerer.

Testfilen
webapi.endpoint.tests.js

[1] `Create, Update, Delete`
[2] `http://qunitjs.com/`

Testrunner

webapi.endpoint.tests.html

Alle mine WEB API URL'er defineret i JavaScript filen *appSettings.js*, som jeg sender en instans af rundt til alle mine view models. I min testfil refererer jeg til denne fil, og henter alle WEB API adresserne, som jeg gemmer i et array. Jeg løber arrayet igennem i en løkke, definerer QUnit `asyncTest`, som jeg giver et navn og en `callback` funktion, der skal kører i testen, for hver URL i arrayet. `callback` funktionen er vist i kodeoversig 7.1 nedenfor.

Listing 7.1: Test af WEB API Endpoints.

```
1  // Test only that the Web API responded to the request with '
       success'
2  var endpointTest = function (url) {
3    $.ajax({
4        url: url,
5        dataType: 'json',
6      success: function (result) {
7        ok(true, 'GET succeeded for ' + url);
8        ok(!!result, 'GET retrieved some data');
9        start();
10     },
11     error: function (result) {
12         ok(false, stringformat('GET on \'{0}\' failed with
13                         status=\'{1}\': {2}', url, result.status,
14                                                 result.responseText));
15       start();
16     }
17   });
18 };
```

Denne metode bliver anvendt som en `callback` i QUnit test definitionen, som er vist nedenfor:

```
// Test each endpoint in apiUrls
for (var i = 0; i < apiUrlsLength; i++) {
    var apiUrl = apiUrls[i];
    // Arguments: testName, callback
    asyncTest('API can be reached: ' + apiUrl,
                                    endpointTest(apiUrl));
};
```

Som det kan ses, løber jeg mit array '`apiUrls`' igennem, og kalde kalde QUnit `asyncTest` testen for hver WEB API URL, og giver den min `callback` funktion (figur 7.1), som kører i testen. Når AJAX anmodningen bliver udført med success, bruger jeg først `ok() assertion` med udrykket `true`, fordi anmodningen var en success, og igen med udtrykket `!!result`, som konvertere `result` objektet til `boolean` værdi— hvis `result` er defineret, evaluerer udtrykket til `true`, og ellers til `false`. Fordi det er en asynkron test, skal testen vente med at køre indtil AJAX anmodningen er færdig. Derfor har QUnit metoden `start()`, som det fremgår af kodeoversigt 7.1, linie 9. Fordi jeg definerer en `asyncTest(...)`,

QUnit vil sætte kørsel af testen på pause indtil den får en notifikation. Det er præcis hvad `start()` gøre— at fortælle QUnit fortsætte kørslen.

7.1.1.2 Kørslen

Testen køres, som nævnt, i en HTML side. Jeg navngiver JavaScript filen, som indeholder testen, og HTML siden, som vil kører testen, med samme navn. På den måde er det nemt at se, at de hører sammen. Nedenfor i kodeoversigt 7.2 er vist min HTML side, som kører min WEB API `endpoints` test. Filen kaldes også for *Testrunner* i QUnit terminoligi.

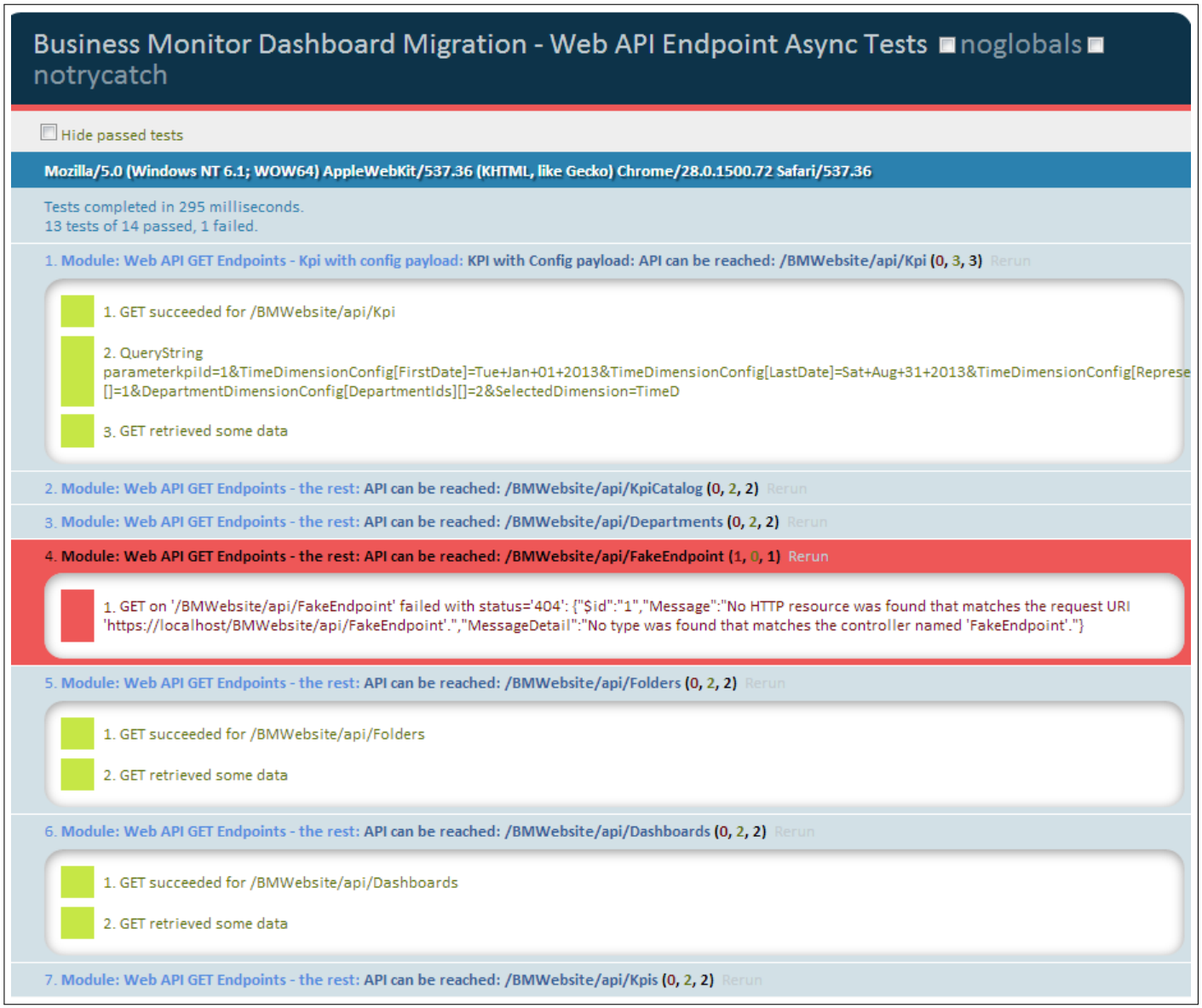
Listing 7.2: Kørsel af WEB API `endpoints` tests - (Testrunner).

```
<h1 id="qunit-header">BM Dashboard Migration - Web API Endpoint
    Async Tests</h1>
<h2 id="qunit-banner"></h2>
<div id="qunit-testrunner-toolbar"></div>
<h2 id="qunit-userAgent"></h2>
<ol id="qunit-tests"></ol>
<div id="qunit-fixture">test markup, will be hidden</div>

<script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.
    min.js"></script>
<script src="../qunit/qunit.js"></script>
<script src="../../Scripts/Javid/appSettings.js"></script>
<script src="webapi.endpoint.tests.js"></script>
```

Som vist i kodeoversigt 7.2 foroven, indkluderer jeg først nogle standard QUnit markup. Derefter tilføjer jeg nogle referencer, fx til jQuery, QUnit biblioteket og den testfil jeg har skrevet. Ved at åbner denne fil i en browser, bliver mine tests eksekveret, og resultatet kan læses, som det fremgår af figur 7.1 nedenfor. Figuren viser resultatet af testkørslen, og som det kan ses, har jeg foldet nogle af testerne ud så de enkelte `assertions` kan studeres. Der er to ting, der er nævneværdige i forhold til resultatet; først, som det kan ses i figuren, er der en test, der har fejlet. Dette er fordi jeg, med vilje, introducerede et '`FakeEndpoint`' i mit array af WEB API URLs, dvs. resultatet er som forventet. Det er en god måde at 'teste' testværtøjet på. Så min test af testværktøjet var successfuld ☺ Det ses at testen har fejlet, da den prøvede at kontakte WEB API med adressen */BMWebsite/api/FakeEndpoint*. Testen udskriver fejlbeskeden, som AJAX anmodningen returnerede. Der er en god grund til at testen fejler— der findes ikke en WEB API Controller med navnet `FakeEndpointController`.

Det andet jeg vil nævne, er i forhold til den første test i figuren— GET `Action` metoden i *KpiController.cs* WEB API Controlleren, forventer noget data (`KpiConfig`), når der sendes en GET anmodning til den. Det bliver sendt som en del af `QueryString` i URLen, som på server siden vil blive mappet til `kpiConfig` parameteren i Controllerens GET metode. Hvis `QueryString` ikke indeholder disse data, returnerer jeg i `Action` metoden en `HttpResponseException` med `HttpStatusCode.NotFound`. Derfor, for at kunne teste denne WEB API adresse, skal jeg indkludere et objekt med kpi konfigurationer i AJAX anmodningen i min

test. jQuerys AJAX metoden vil serialisere dette objekt til `QueryString` format og sætte det sammen den egentlige URL. Dette er hvad man kan se i testen `assertion` nummer 2.



Figur 7.1: WEB API endpoints testresultater.

7.1.2 Web API GET Result Tests

Her vil jeg, som nævnt, undersøge om WEB API GET anmodningerne, returnerer det forventet data. Jeg vil se på det returnerede data, og sammenligne med det jeg forventer at få tilbage, ved at anvende QUnit `eqval() asserion`. Signaturen for `eqval() asserion` er som følgende:

```
equal( actual, expected, message )
```

`eqval() asserion` bruger en simpel sammenligning vha '==' operatoren til at sammenligne de to argumenter— `actual` og `expected`. Således baseres testens udfald på om det aktuelle resultat er det sammen som det forventede.

7.1.2.1 Testen

Jeg vil test om:

- WEB API GET anmodninger returnerer det forventede data.
- WEB API kan forespørges om noget specifikt data ved angivelse af dataets `id`.
- WEB API `Route` konfigurationer virker efter hensigten.

Testfilen
webapi.get-result.tests.js

Testrunner
webapi.get-result.tests.html

Jeg vil fx teste om min WEB API kan anmodes om et bestemt dashboard, ved at indkludere dashboardets `id` i WEB API `endpoint` adressen for `Dashboard` Controller. Derved testes ligeledes, om WEB APIs `Route` konfiguration, navigerer anmodningen til den pågældende HTTP `Action` metode i Controlleren, der finder et dashboard vha. dets `id`, hvor `id`, fra `QueryString`, mappes til metodens id-parameter. Kodeoversigt 7.3 viser, hvordan denne test er sat op.

Listing 7.3: Test af WEB API mht. *../api/Dashboard/2.*

```
1  module('GET the Dashboard with id');
2  asyncTest('Expected results (Dashboards/2): ' +
3                  settings.dashboardsUrl+'/'+2, 4,  function () {
4    var url = settings.dashboardsUrl + '/' + 2;
5    $.ajax({
6      url: url,
7    dataType: 'json',
8    success: function (result) {
9      ok(result.Id === 2, 'Got the dashboard from ' + url);
10       equal(result.Title, 'Costs', 'The title is as expected');
11     equal(result.Id, 2, 'The id is as expected');
12     equal(result.FolderId, 1, 'The folder id is as expected');
13     start();
```

```
14    },
15      error: function (result) {
16        ok(false, 'GET on \'{0}\' failed with status=\'{1}\': {2}'
17                  .format(url, result.status, result.responseText));
18      start();
19      }
20    });
21  });
```

Jeg bruger `asyncTest`, da der foretages en AJAX anmodning i testen. Første argument til testmetoden er navnet på testen, som indkluderer den WEB API URL, jeg vil teste. Det andet argument angiver antallet af `assertions` i testen, som testen forventer skal bestås med success. Her har jeg angivet 4, da jeg har 4 `assertions` i min test. Det tredje argument er `callback` funktionen, som er selve testens krop. I modsætning til det forrige eksempel, har jeg her defineret `callback` funktionen på stedet.

Jeg retter AJAX anmodning mod adressen *../api/Dashboards/2*, og forventer at få dashboard med `id` nummer 2 returneret. Hvis anmodningen lykkes, undersøger jeg det returnerede data i nogle `assertions`.

7.1.2.2 Kørslen

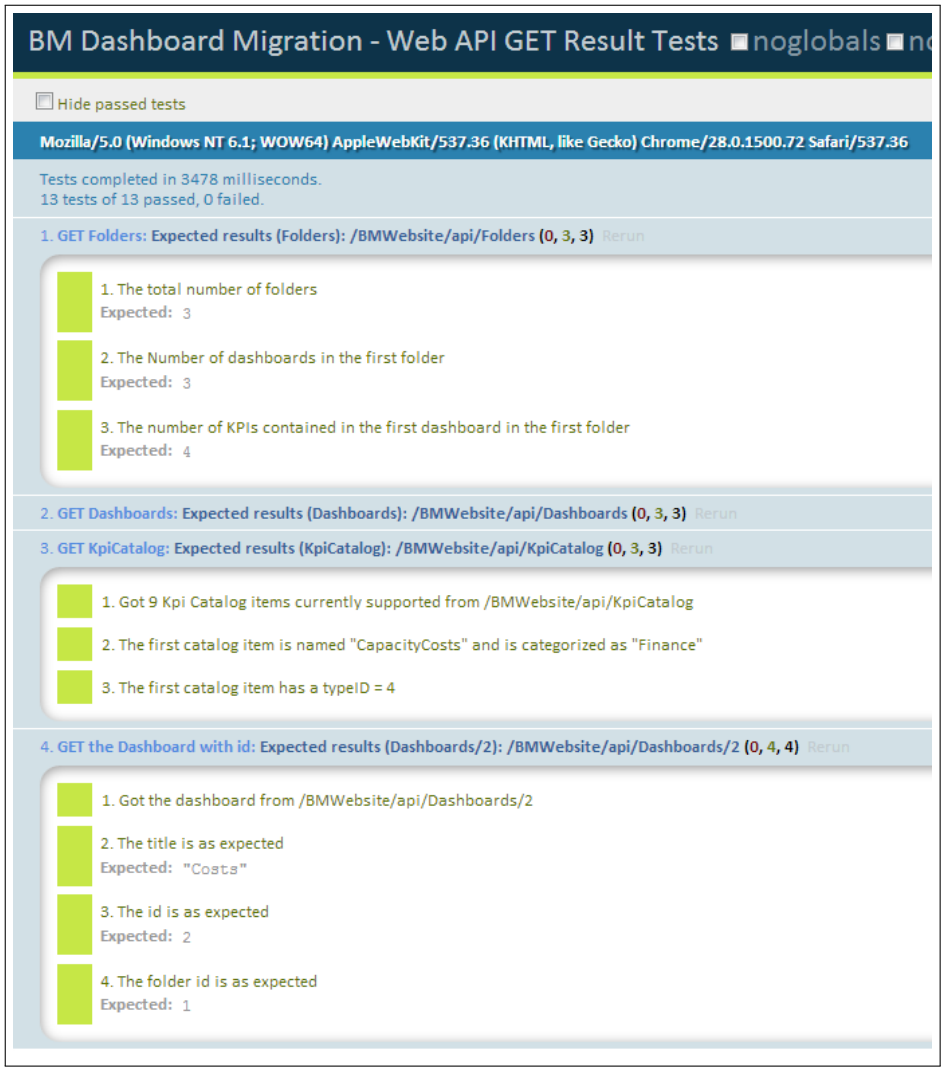
Figur 7.2 viser resultater af kørslen, som udover den beskrevne test, også indkludere andre lignende tests. Den beskrevne test er vist som den sidste test i figuren, og som det ses, er alle tests udført med success.

7.1.3 Web API CUD Tests

Jeg vil også teste mine WEB API CUD (CREATE, UPDATE, DELETE) operationer. Derfor har jeg skrevet en række tests, der verificerer disse WEB API operationer. Jeg har skrevet CUD tests for `Folders` og `Dashboards` Controllers. For at udføre disse tests, har jeg fulgt følgende procedure:

7.1.3.1 Testen

- Først anmoder jeg fx et dashboard.
- Jeg foretager en ændring på dashboard objektet, fx ændrer jeg titlen, og laver en PUT anmodning til WEB API Controlleren for at gemme ændringen.
- Jeg forespørger WEB API for at hente dasboardet igen, og bekræfter ændringen.
- Jeg gendanner det oprindelige dashboard i db.
- Jeg bekræfter gendannelsen.



Figur 7.2: WEB API GET result testresultater.

- For at teste POST og DELETE anmodninger, opretter jeg et dashboard i et andet testmodul, og poste til WEB API. Når operationen lykkes, slette jeg dette testdashboard ved at lave en DELETE anmodning.

Testfilerne

webapi.dashboard-cud.tests.js
webapi.folder-cud.tests.js

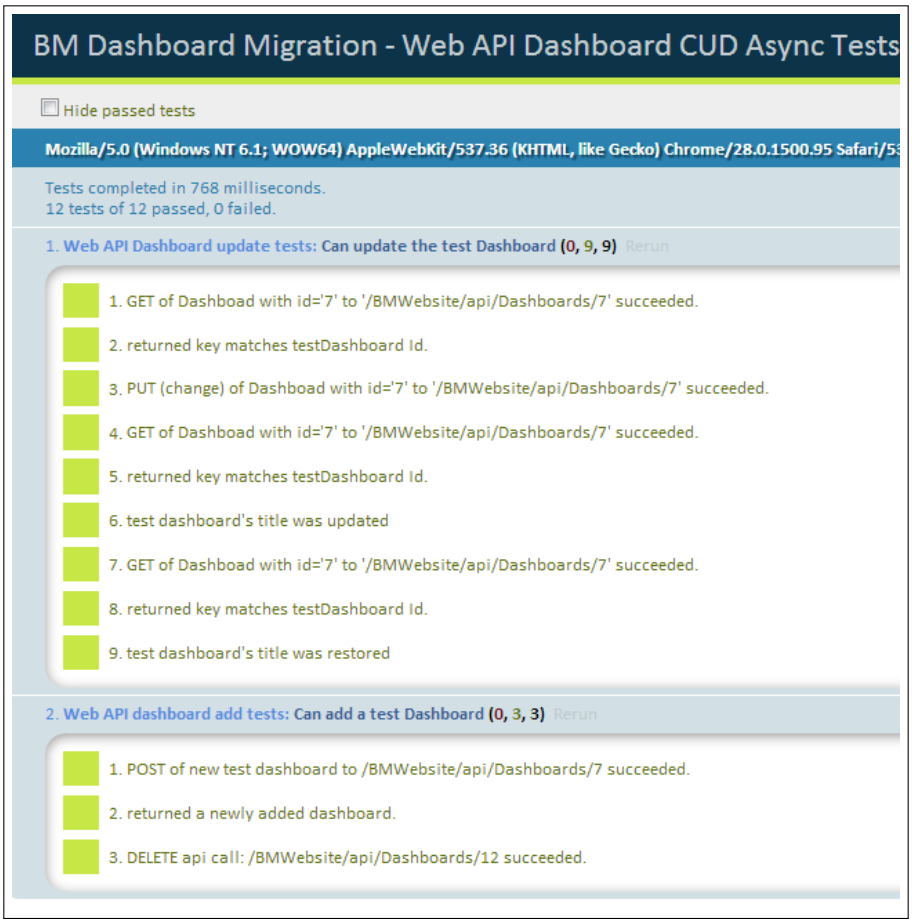
TestRunners

webapi.dashboard-cud.tests.html

webapi.folder-cud.tests.html

7.1.3.2 Kørslen

Figur 7.3 viser resultatet af kørslen af dashboard CUD testen. Som det kan ses, er alle trinene lykkedes.



Figur 7.3: WEB API dashboard CUD testresultater.

7.2 Test af systemets funktioner

I denne sektion, vil jeg undersøge systemets funktioner, hvor jeg tager en `Black-Box` tilgang til test. Normalt vil sådan en test blive udført af en tester, der ikke ved noget om den interne implementering af systemet, men som blot

ved, hvad systemet skal kunne. Med udgangspunkt i kravspecifikationen, er der udarbejdet en række test cases, som testen er baseret på.

7.2.1 Test Cases

Jeg vælger selv at udføre de funktionelle tests, men som nævnt før, er det bedst at `Black-Box` tests bliver planlagt og udført af en, som ikke har udviklet koden, og som ikke har indsigt i kodens strukturer. Ulempen ved at udvikleren selv tester, er at de er tilbøjelige til at teste, at programmet agerer som de har programmeret det til at gøre, i stedet for at teste og sikre at programmet gør som brugeren vil have det skal gør.

Der er defineret følgende test cases. Ideelt, vil man teste alt hvad programmet kan gøre, men for at kunne overholde tidsrammen for projektet, skriver- og udfører jeg test cases for de mest almidelige brug af systemet.

Definition af test cases		
Test ID	**Navn**	**Kommentar**
TC1	*Tilføj et Detalje-*KPI	Verificerer muligheden for at kunne tilføje et KPI i detaljevisning. KPI'et vælges fra KPI kataloget, hvor man også kan læse en beskrivelse af KPI'et. (Et KPI kan også tilføjes som et overbliks-KPI).
TC2	*Indstil et* KPI *med ændret tidsperiode*	Verificerer KPI konfigurations muligheder. Til hvert KPI hører der en analysvisning (en menu), hvor KPI konfigurationer kan indstilles.
TC3	*Skift visning af et Detalje-*KPI	Verificerer, at et Detalje-KPI kan vises på to måder: som en graf eller tabel.
TC4	*Opret et dashboard*	Verificerer muligheden for at kunne oprette nyt dashboard tilknyttet en folder. En folder kan indeholder flere dashboards, hvor det valgte dashboards indhold vises i indholdsområdet.
TC5	*Slet et dashboard*	Verificerer at det pågældende dashboard kan slettes, og alt dets indhold vil ligeledes også være slettet.

Tabel 7.1: Definition af test cases

Specifikationer af test cases			
ID	**Beskrivelse**	**Forventede Resultater**	**Observerede Resultater**
TC1	*Indgangsbetingelser*: Aplikationen er indlæst, brugeren er autentificeret, kan se en oversigt over sine dashboards i venstre side, og det valgte dashboard i indholdsområdet til højre. Brugeren klikker på knappen '`Add KPI`' i det valgte dashboard.		
		Et modalt vindue er åbnet, hvor der er vist en tabel. Tabellen viser en lise over KPIer, som systemet understøtter. '`Add`' knappen skal kun være aktiv for brugeren, hvis et KPI er valgt i listen. Brugeren har desuden mulighed for at lukke dette modal vindue, uanset om et KPI er valgt, og vender tilbage til sit valgte dashbaord.	
	Brugeren vælger et KPI ved at klikke på det pågældende element i listen.		
		Den pågældende række i tabellen er markeret. Der er vist en beskrivelse af det valgte KPI i beskrivelsesområdet for KPI'er. Visningen for KPI'et er sat til *Detalje* i *Select Display* området.	
	Brugeren klikker på '`Add`' knappen.		
		Et detalje-kpi er tilføjet til det valgte dashboard. KPI'et er visualiseret med tilhørende knapper til konfiguration, skift af visning, sletning og gem funktionerne.	*Verificeret*
TC2	*Indgangsbetingelser*: TC1 Brugeren klikker på KPI konfigurationsikonet for det pågældende KPI.		
		KPI analysemenuen folder ud, og viser de nuværende indstillede konfigurationer.	
	Brugeren vælger en anden regnskabsperiode i '*Time Period*' området i menuen, ved at klikke på den pågældende radioknap, og klikker herefter på knappen '`Calculate`'.		

Fortsætter på næste side

Tabel 7.2 – *Fortsættelse fra forrige side*

ID	Beskrivelse	Forventede Resultater	Observerede Resultater
		Datavisualiseringsgrafen for det pågældende KPI skifter til en opdateret graf, der afspjler den nye periode.	*Verificeret*
TC3	*Indgangsbetingelser*: TC1, KPI'et er vist på dashboaret som en graf. Brugeren klikker på ikonet for tabelvisning.		
		Datavisualiseringsgrafen for det pågældende KPI er forvandlet til en tabel. Ikonet indikerer grafvisning	*Verificeret*
TC4	*Indgangsbetingelser*: Aplikationen er indlæst, brugeren er autentificeret, kan se en oversigt over sine dashboards i venstre side, og det valgte dashboard i indholdsområdet til højre. Brugeren klikker på en folder, hvortil et nyt dashboard skal tilføjes.		
		Den valgte folder er markeret og der er fremkommet et lille plusikon ved denne folder. *(Indholdsområdet til højre er eventuelt initialiseret med indhold af det dashboard i folderen, der var valgt, sidst folderen blev besøgt).*	
	Brugeren klikker på plusikonet ved folderen.		
		Et modalt vindue er åbnet bestående af et indtastningsfelt med teksten *dashboard title*, og knappen `Create`, som aktiveres, når indtastningsfeltet er udfyldt.	
	Brugeren Indtaster en titel og klikker på knappen '`Create`'		
		Der er oprettet nyt dashboard, som er vist i indholdsområdet, med titlen vist i toppen og knappen '`Add KPI`'.	Det blev konstateret, at hvis brugeren klikker på '`Create`' knappen to eller flere gange hurtigt efter hinanden, før modal dialogen når at lukker ned, oprettes flere dashbaords med samme titel i folderen. Det er selvfølgelige noget der skal forhindres.
TC5	*Indgangsbetingelser*: TC4 Brugeren klikker på et dashboard i navigationsområdet til venstre.		

Fortsætter på næste side

Tabel 7.2 – *Fortsættelse fra forrige side*

ID	Beskrivelse	Forventede Resultater	Observerede Resultater
		Der er fremkommet et lille sletikon ved titlen af valgte dashboard. Indholdsområdet er initialiseret med det valgte dashbards indhold.	
	Brugeren klikker på sletikonet.		
		Et modalt dialog er åbnet med mulighed for be- eller afkræfter sletningen.	
	Brugeren klikker på knappen 'OK'.		
		Dashboardet er slettet, og indholdsområdet er initialiseret med indhold af næstsidste dashboard i folderen.	Det næstsidste dashboard vises, men dets indhold vises ikke korrekt. Klikkes på dashboard titlen i navigationsområdet, bliver dets indhold vist genindlæst og vist korrekt. Desuden blev det konstateret, at modaldialogen ikke lukkede med det samme, da jeg klikkede OK, så jeg nåede at klikke OK igen. Jeg fik en *Error - not found* notifikation på skærmen, men det valgte dashboard blev slettet.

Tabel 7.2: Specifikationer af test cases

7.3 Opsummering

Der findes flere typer af software tests. Hver type har specifikationer, der definerer korrekt adfærd, som testen undersøger, for at identificere ukorrekt adfærd. Overordnet set, er der to måder at gribe det an på— `White-Box` test, og `Black-Box` test. Disse angiver en testers gennemsigtigheden til koden. Der blev i dette kapitel udført `unit` tests, som er et eksempel på `White-Box` test, og der blev udført funktionelle tests, som er et eksempel på `Black-Box` test. I fobindelse med de funktionelle tests, blev der identificeret nogle ukorrekt adfærd, som skyldes den asynkrone kommunikation med serveren og dens hastighed. Der skal træffes de nødvendige foranstaltninger, såvel på serveren som på klienten, der forhindre disse typer fejl.

Kapitel 8

Konklusion

En af de støreste fordele ved en HTML5 applikation, i modsætning til fx Silverlight, er at den ikke kræver en browser `plugin`. HTML5 har den bredeste rækkevidde på tværs af platforme og enheder, og dens muligheder for udvikling på tværs af platforme har opnået store popularitet. Business Monitor er en Silverlight applikation, og kræver et `run-time` miljø baseret på en browser plugin. Den støreste ulempe ved det er, at understøttelsen er begrænset, og på mobile platforme er den slet ikke undestøttet. Dette projekt skal ses som et svar på spørgsmålet om, hvad der skal til for Business Monitor applikationen at migrere til en HTML5/JavaScript løsning, med udgangspunkt i Business Monitor Dashboard modulet.

8.1 Problemstillingerne

I Introduktions kapitlet blev problemområdet defineret gennem en række problempunkter, som jeg i det følgende ser tilbage på, om hvorvidt de er blevet mødt gennem projektet.

8.1.1 Udviklingsmetoden

UP har været en passende udviklingsmetode til styringen af projektet. I startfasen, har der været mange åbne spørgsmål mht. valg af teknologier, krav osv. Størstedelen af arbejdsbyrden i startfasen har derfor været relateret til det at vælge passende teknologier, identificere de mest centrale krav og analysere dem. Det har også været vigtigt for mig at finde ud af, tidligt i forløbet, hvilke teknologier jeg kunne bruge, og hvordan. Derfor har der også været en del eksperimenterende implementeringsarbejde, så de sorte huller kunne lukkes. Arbejdsbyrden har langsomt flyttet sig på tværs af aktiviteterne fra `Krav, Analyse, Design` til `Implementering` og `Test` senere i forløbet. Dette er i overensstemmelse med principperne i UP, hvor indsatsen i forhold til de firskellige aktiviteter ændrer sig over tid.

8.1.2 Teknologi analyse

Indledningsvis blev problemområdet defineret og et sæt krav til prototypen blev defineret, der førte til design og implemetering af prototypen. Der blev desuden lavet en analyse af teknologierne, anvendt til realisering af prototypen— jQuery gør DOM manipulering, og udførelse af AJAX anmodninger nemmere. Knockout giver mulighed for `data-binding` og anvendelse af MVVM i en webapplikation. Kendo UI frameworket blev anvendt til datavisualiserings widgets. Det blev dækket gennem projektet, hvordan disse biblioteker og den udviklede kode, arbejder sammen på en afkoblet måde, ved brug af designmønstre som MVVM og JavaScripts `Constructor Pattern`.

8.1.3 Arkitektur

Den overordnede arkitektur er baseret på et `client-server` miljø, hvor en webserver eksponerer HTML/CSS/JS til klienten. Webserveren eksponerer også nogle datatjenester, som jeg implementerede ved brug af ASP.NET WEB API. Klientsiden består af HTML/CSS, som definerer sidens struktur og udseende, og JavaScript kode, som indkapsler applikationens adfærd. Et af formålene med denne arkitektur har været at eksponere data og operationer på en simpel måde, så man nemt kan tilføje nye måder at hente data, eller tilføje nye data. Ideen er at holde WEB APIs metoderne simple, hvilket jeg mener er opnået.

8.1.4 Persistens

Persistens blev understøttet af EF med Code First modellerings workflow. Jeg valgte EF fordi det er en solid ORM, som virker godt med webteknologier. Dens Code First workflow har givet mig mulighed for at definere mine models ved brug af POCO (plain old class objects), som EF bruger til at generere databasen og udføre operationer.

8.1.5 Designmønstre

Jeg har gjort brug af desginmønstre (`patterns`) i udviklingen af prototypen, på serveren og klienten, for at overholde SRP (Single Responsibility Principle) princippet— `Factory pattern` til oprettelse af KPI'er i *BM.KpiEngine* modulet. MVVM på klienten vha. Knockout View Models. `Constructor pattern` til oprettelse af JavaScript objekter. Dette er med til at gør applikationen nemmere at debugge, skalere og vedligeholde.

8.1.6 Web API

Efter applikationen er indlæst i browseren, foretages al efterfølgende kommunikation med serveren gennem asynkrone AJAX anmodninger. Denne kommunikation sker gennem web service laget med udveksling af JSON data. Jeg implementerede web service laget ved brug ASP .NET WEB API, som gør JSON datatjenester simplere. En simpel WEB API gør applikationen skalerbar.

8.1.7 Mobil strategi

For at komme på det mobile platform, er der en række tilgange man kan vælge. I Design kapitlet har jeg behandlet dette emne, hvor jeg gennemgår de forskellige tilgange og undersøger fordele og ulemper ved hver. Overordnetset, er der mulighed for at vælge mellem en mobiloptimeret webapplikation eller native app. En native app skal specialudvikles til hver af de forskillige mobilplatforme, kræver bredere udviklingskompetencer og er derfor også dyere at udvikle. En mobiloptimeret webapplikation (vha. Responsive Design), derimod er billigere, både at udvikle og i drift, da den virker på alle enheder og platformer.

8.2 Projektforløbet

Der blev udviklet en fungerende prototype— en HTML5 dashboard løsning[1] med mulighed for organisering og opbygning af dashboards. Løsningen giver mulighed for håndteringen og beregningen af nogle udvalgte nøgletal. Den implementerede arkitektur giver mulighed for nemt at udvide *BM.KpiEngine* modulet, til at kunne håndtere flere nøgletal, uden at de andre dele af systemet behøver at have kendskab til denne udvidelse— de nye nøgletal vil være præsenteret i nøgletalskataloget i brugergrænsefladen– de vil kunne vælges og tilføjes til et dashboard, og indstilles og gemmes i systemet.
Løsningen omfatter dashboards med deres KPI indhold visualiseret både som graf og som tabel, hvor visning kan skiftes mellem de to former. Nøgletalsberegningerne er baseret på nogle indstillinger, brugeren definerer. Nøgletallene er dokumenteret i nøgletalskataloget, hvor man også kan læse en beskrivelse af dem.

Projektet har været en god, udfordrende og lærerig oplevelse. God vejledning, såvel intern som ekstern, har været en vigtig ingrediens i min positive erfaring med projektet. Projektet har givet mig nogle vigitige, tidssvarende og fremtidssikrede kompetencer, som jeg vil få brug for i mit fremtidigt arbejdsliv.

`Business Monitor Dashboard Migration` projektet har været en erfaringsrig oplevelse. Selvom jeg ikke havde haft berøring med teknologierne fra tidligere, er det lykkedes at anvende dem sammen under realisering af prototypen. Projektet vil være et godt fundament for den egentlige konverteringsprocess af Business Monitor applikationen.

[1] `https://bmjavid.cloudapp.net/`

8.3 Videreudvikling

Den opstillede kravspecifikation er blevet overholdt, men der er selvfølgelig plads til forbedringer. Nu, hvor jeg befinder mig på den anden side af projektforløbet, føler jeg at jeg har lært meget i forhold til HTML og JavaScript. Den fleksibilitet og de velafprøvede designmønstre, som server-side tekonologier giver adgang til for at opnå bedre organisering af kode, bliver efterhånden også tilgænglige i HTM5 verdenen i kraft af de mange JavaScript biblioteker. De giver udviklingen i HTML5/JavaScript de samme kvaliteter.

Modularitet er en vigitig ting i forbindelse med strukturering af kode, nu hvor klientsiden indeholder så meget logik med HTML5/JavaScript udvikling. Under udvikling af prototypen har jeg benyttet JavaScripts `constructor pattern` til at indkapsle variabler og metoder, og undgår derved spaghetti-kode. Dette kan tages et skridt videre således at man modularisere så meget som muligt. Jeg kunne fx definere et modul til håndtering af data (`datacontext`). Dette modul vil således alene være anvarligt for hentningen og opdateringen af data—hvis jeg, fx fra min view model, ville spørge efter et dashboard, ville jeg kalde `datacontext.dashboards.getDashboardById(id)`. På den måde ville mine view models være fri for håndtering af asynkrone AJAX kald, og henvender sig til `datacontext` modulet hvis de skal bruge eller gemme data. `datacontext` vil ligeledes have metoder til håndtering af `folders`, `Kpis` og andre entiteter.

View'et i prototypen består af en lang HTML-fil med sektioner bundet til data, som er eksponeret via mine view models. HTML siden kunne med fordel splittes op, således at jeg fuglte den konvention, hvor jeg havde en hoved HTML-side, og en partiel HTML-side for hver view model. Det ville give en mere modulariseret og overskuelig struktur.

De nævnte optimeringer og effektiviseringer (og mange flere), kan opnås vha. JavaScript biblioteker som `Durandal` og `RequireJS`. De giver mulighed for konventionbaserede designmønstre som i WPF/Silverlight udvikling. Hvis jeg skulle lave projektet i dag, ville jeg helt klart anvende og følge de nævnte praksisser.

29. juli 2013

8.4 Udtalelse fra virksomheden

Udtalelse om bachelorprojektet: Business Monitor Dashboard Migration

Business Monitor er en webapplikation, der afvikles i Microsoft Silverlight. Silverlight er et browser plug-in, der skal installeres i browseren, før applikationen kan køres. Vi har oplevet flere kedelige konsekvenser ved brugen af Silverlight, hvor de kritiske omhandler memory leaks i Apples OS X, brugergrænseflade fejl i Google Chrome på IOS (ikke muligt at indtaste i indtastningsfelter) og at Silverlight ikke kan afvikles i browseren på de mest brugte mobile apparater (Android og iOS). Vi frygter, at Silverlight har fået en dødsdom efter HTML5, CSS og JavaScript frameworks, som jQuery, er kommet på banen. Vi ønsker at følge med tiden, så derfor søsatte vil *Business Monitor Dashboard Migration*-projektet.
Vores motivation for at få kortlagt og implementeret en prototype af Business Monitor med en HTML/CSS/JavaScript (herefter omtalt som blot HTML) har været stor. Vi ønskede at få afklaret, hvad der er muligt, og hvordan det kunne implementeres med brug af de nyeste webteknologier. Udgangspunktet blev Business Monitor Dashboard – en Business Intelligens løsning, der tager udgangspunkt i økonomiske nøgletal og giver brugeren mulighed for selv at tilpasse sine egne Dashboards.

Javid har udviklet en fungerende prototype, der viser, at en HTML brugergrænseflade godt kan leve op til Silverlights formåen. Især brugen af asynkrone server forespørgsler har imponeret os og vist, hvor langt en interaktiv og dynamisk en HTML brugergrænseflade er.
Han har udviklet et WebAPI, der håndterer beregningen af nøgletal. Nøgletalsberegningen er påvirket af brugerindstillinger, og beregningsstrukturen kan håndtere flere dimensioner og er let at udvide.
Javid har også udviklet et system til vedligeholdelse af brugerdefinerede opsætning. Dette omhandler organisering af Dashboards i en mappestruktur, Dashboards og deres indhold (nøgletal) samt brugerens opsætning og indstilling af nøgletal og deres dimensioner.
Javids arbejde har givet os et overblik over konverteringsprocessen fra en Silverlight- til HTML frontend samt belyst, hvilket udfordringer vi vil støde på.

Business Monitor Dashboard Migration vil blive brugt som erfaringsgrundlag og udgangspunkt for udarbejdelsen af et endeligt konverteringsdesign. Alle midler, energi og fokus er rettet mod salg og en konsolidering af den eksisterende løsning.

Med venlig hilsen

Frederik Kiær
Partner
E-mail: kiaer@businessmonitor.dk

Business Monitor® • Fruebjergvej 3 • DK-2100 Copenhagen

Bilag A

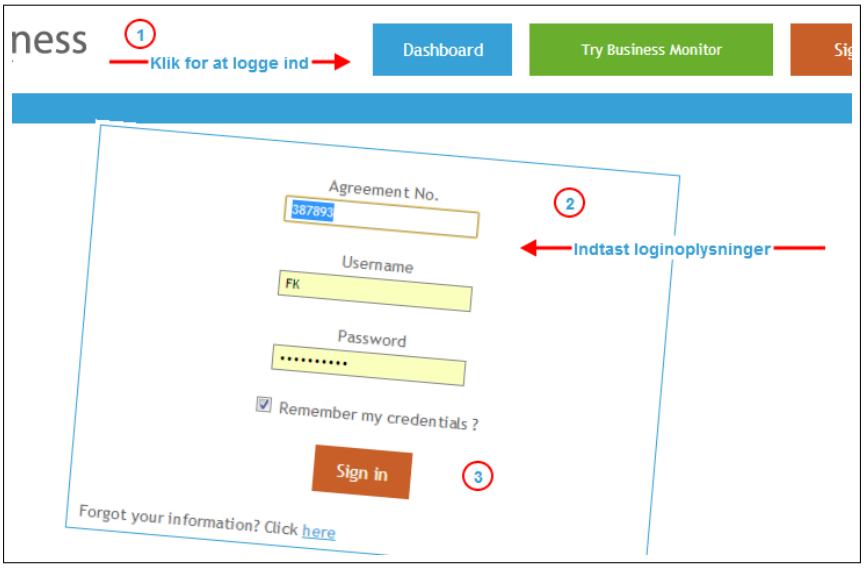
Appendiks

Dette kapitel indeholder vejledninger, skærmbilleder og glossar.

A.1 Manual & skærmbilleder

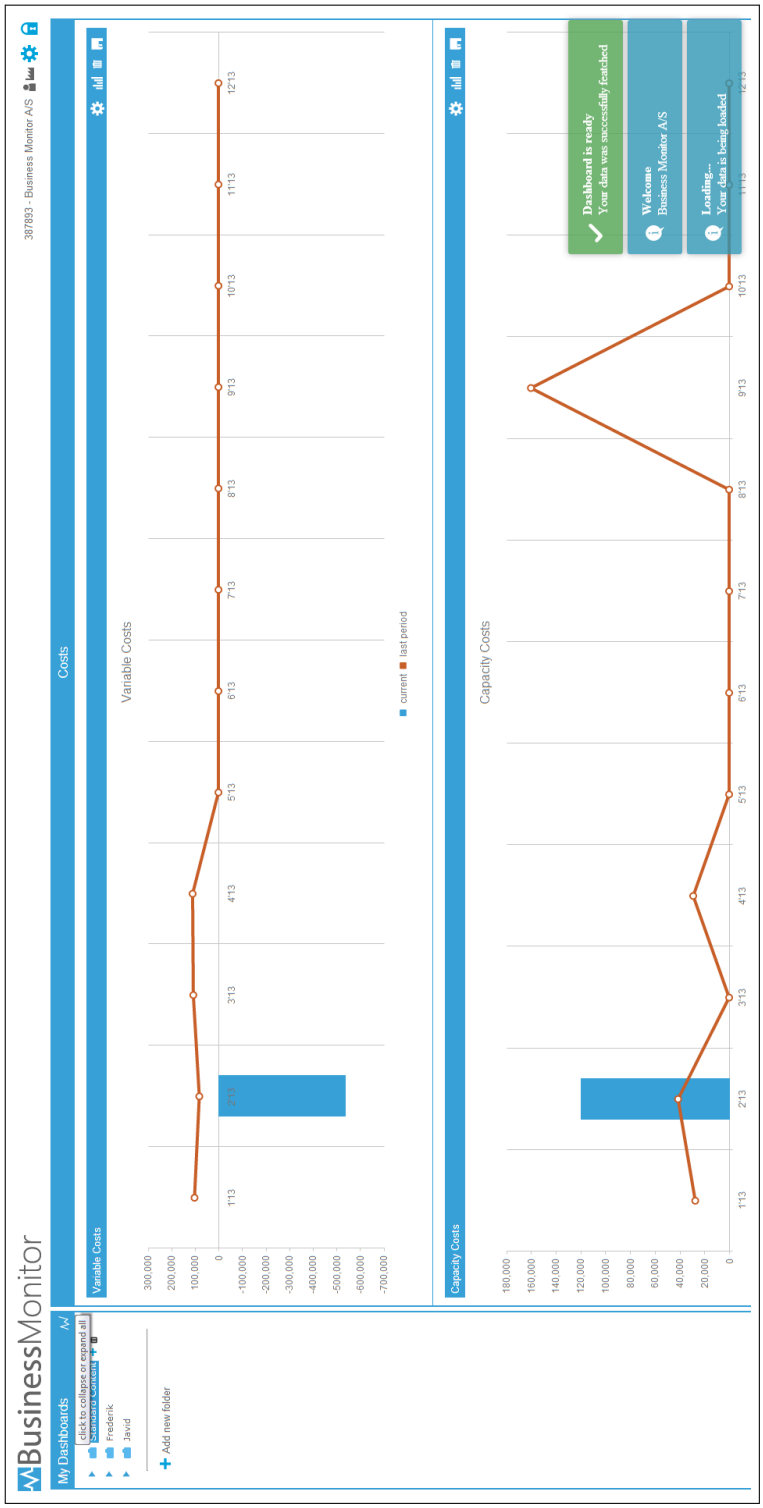
A.1.1 Log på

Gå ind på webadressen `https://localhost/BMWebsite/`, se bort fra SSL advarslen og klik på `Dashboard` knappen øverst. Du bliver omdirigeret til en loginside. Her skal du indtaste login oplysnigerne som vist i figur A.1 nedenfor.



Figur A.1: Gå ind på `https://localhost/BMWebsite/`, klik `Dashboard` og indtast login oplysninger.

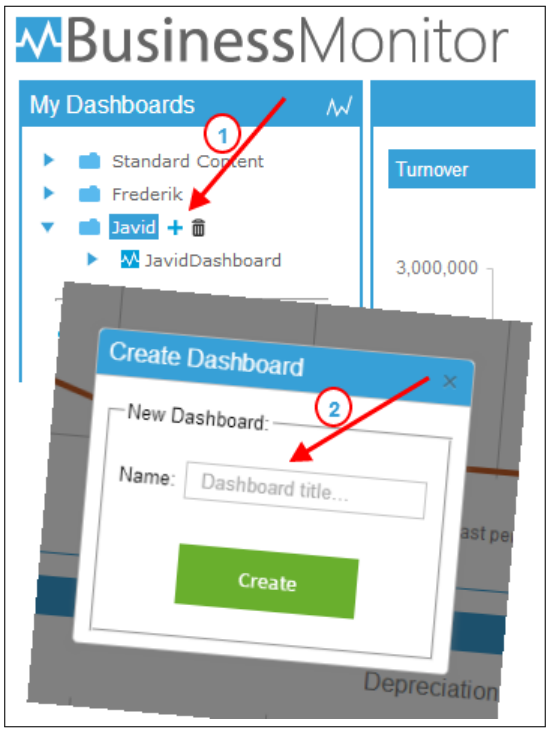
Herefter indlæses applikationen som vist i figur A.2 på næste side. Som det kan ses, har bruger `FK` med aftalenummer `387893`, Nogle dashboards organiseret i tre mapper. Ved at klikke på `My Dashboards` øverst i venstreside, skjules og/eller udvides mapperne med deres indold arrangeret i en træstruktur.



Figur A.2: Applikationen indlæses efter login

A.1.2 Opret Dashboard

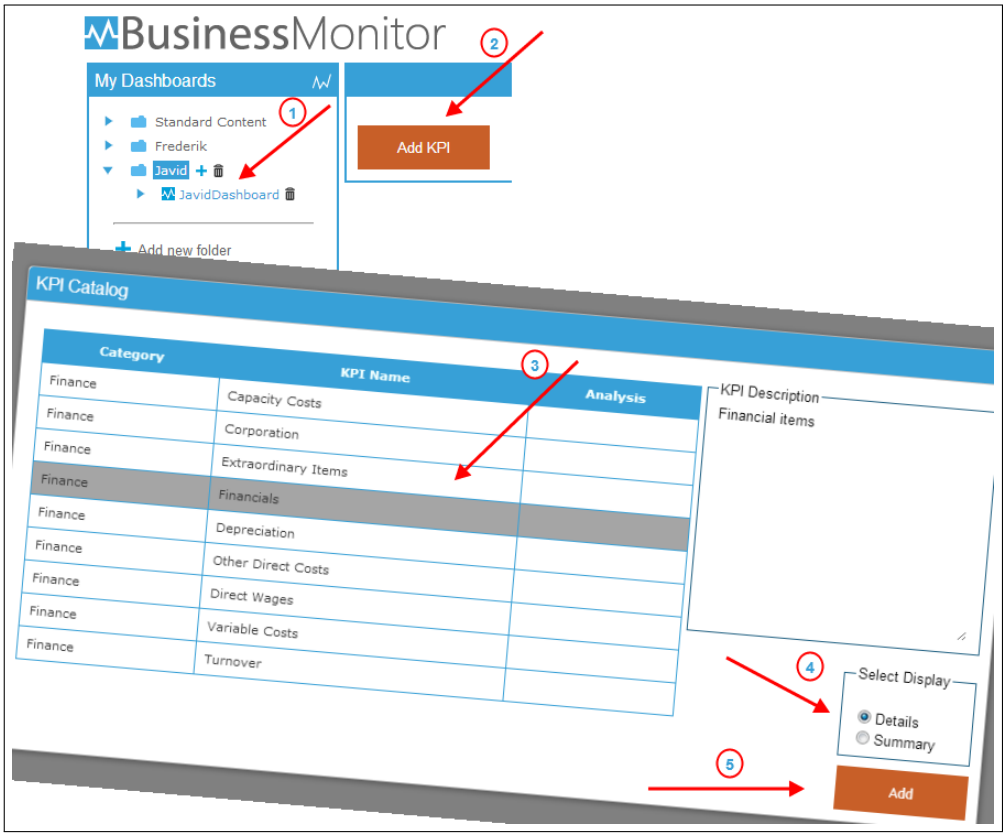
For at oprette nyt dashboard, vælges mappen hvor dashboardet ønskes tilføjet. Klik på det lille plusikon ved siden af den valgte mappe. Et modalt vindue åbnes, hvor der indtastes en titel til dashboardet. Klik herefter opret som vist i figur A.3 nedenfor. Et Nyt dashboard er nu oprettet, klart til at tilføje KPI indhold.



Figur A.3: Opret nyt dashboard.

A.1.3 Tilføj KPI

For at tilføje et KPI, vælges dashboardet hvor KPI'et ønskes tilføjet. Klik på `Add KPI` på dashboardet. Et modalt vindue åbnes indeholdende alle KPI'erne, som systemet understøtter. Vælg et KPI i kataloget og klik - vælg eventuelt en visning (detalje-kpi eller overbliks-kpi) - og klik `Add`. Se figur A.4



Figur A.4: Tilføj KPI.

På næste side vises KPI'et tilføjet både som detalje- og overbliks KPI.

A.1.3.1 Detalje- og overbliks KPI

Figure A.5 nedenfor illustrerer, hvordan et `Financials` KPI ser ud, når det er tilføjet både som detalje-kpi og overbliks-kpi.

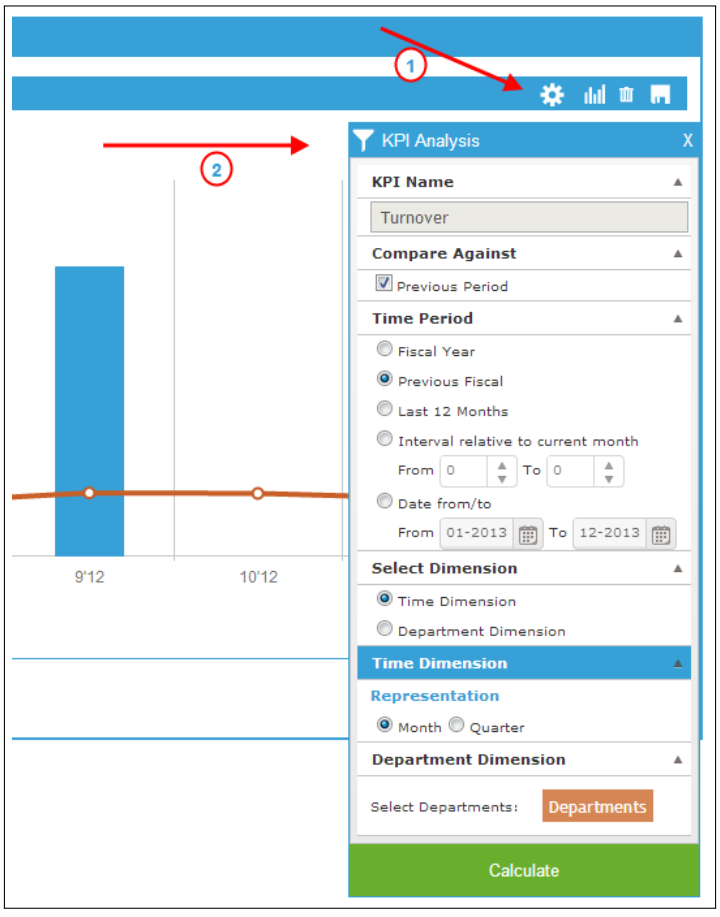
Figur A.5: `Financials` KPI tilføjet både som detalje- og overbliks-kpi.

A.1.4 KPI Analyse

Til hvert KPI på dashboardet hører en værktøjslinje med funktionerne:

- KPI konfiguration - KPI-analyse menu.
- Skift visning - mellem graf- og tabelvisning.
- Slet KPI.
- Gem KPI - persisterer ændringer foretaget via KPI analye menuen.

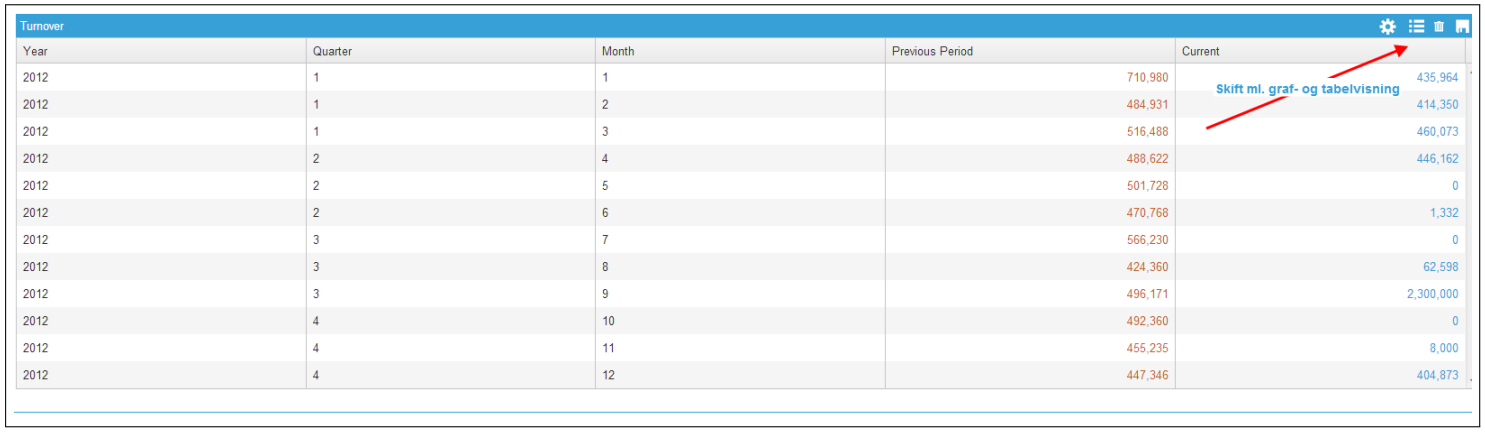
A.1.4.1 KPI Konfigurationsmenu



Figur A.6: Konfigurationsmenu - klik på tandhjulsikonet for at folde menuen ind og ud.

A.1.4.2 Skift Visning

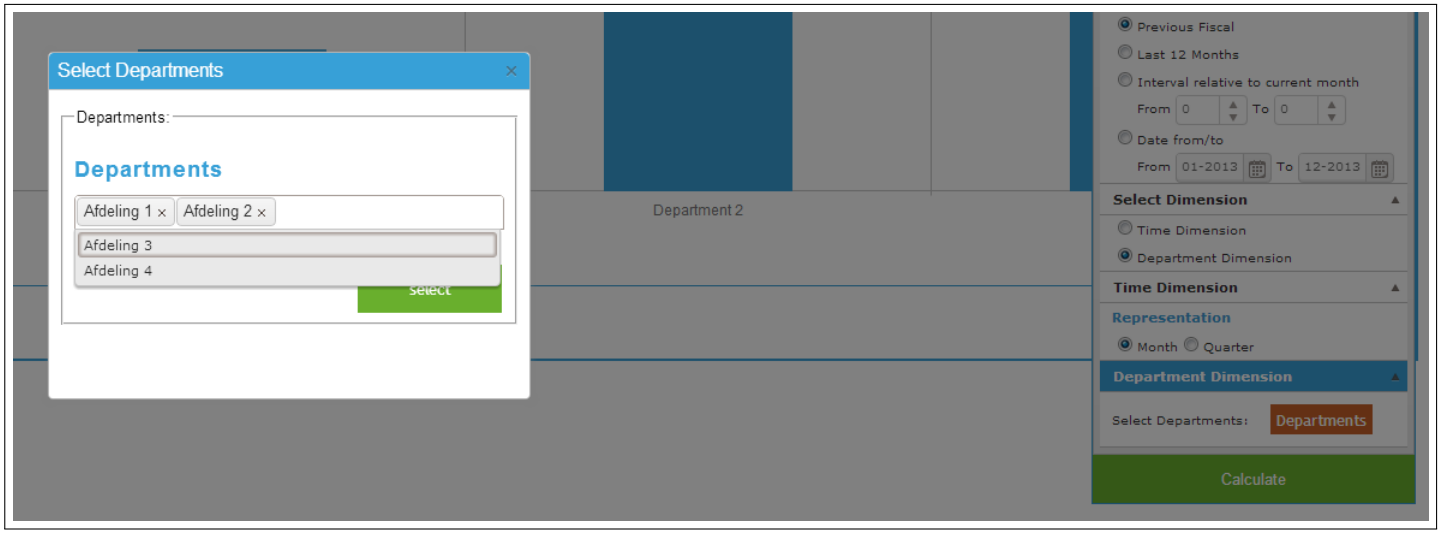
Visningen af et detalje-kpi kan skiftes mellem graf- og tabelvisning som vist i figur A.7 nedenfor. Klik igen for at skifte visningen til graf.



Figur A.7: Konfigurationsmenu - klik på tandhjulsikonet for at folde menuen ind og ud.

A.1.4.3 KPI Dimension

Der kan vælges mellem tidsdimension eller afdelingsdimensionen. Når afdelingsdimension er valgt, aktiveres knappen `Departments` i KPI analysemenuen. Ved at klikke på den, åbnes et modalt vindue, hvor de tilgængelige afdelinger kan vælges vha. et `multiselect input`felt. Dette er vist i figur A.8 nedenfor.



Figur A.8: Vælg afdelinger, når afdelingsdimensionen er valgt i analysemenuen.

A.2 Glossar

AJAX
AJAX (en forkortelse for Asynkron JavaScript og XML) er en webudviklingsteknik til at udvikle interaktive webapplikationer med det formål at øge websidens interaktivitet, hastighed og brugervenlighed. Selvom XML indgår i navnet bruges i dag typisk JSON som format til at overføre data mellem server og klient.

CRUD
`Create, Read, Update` og `Delate` er de 4 grundlæggende funktioner af persistens.

DOM
`Document Object Model` er en tvær-platform sprog-uafhængig konvention for at repræsentere og interagere med objekter i HTML dokumenter.

EF
`Entity Framework` er en videreudvikling af teknologierne i ADO.NET-dataadgang pakken. `Entity Framework` gør det muligt for udviklere at programmere mod relationsdatabaser i overensstemmelse med programspecifikke domænemodeller i stedet for de underliggende databasemodeller.

iOS
Et styresystem udviklet af Apple Inc. til iPhone, iPod Touch, iPad og 2. generation af Apple TV.

JSON
JavaScript Object Notation JSON er tekstbaseret åben standard til at læsning/skrive data.

KPI
Key Performance Indicator (KPI) bliver brugt i forretningshenseende i forbindelse med overvågning af, hvordan virksomheden klarer sig. På den måde kan man måle en virksomhed ud fra nogle parametre, som er defineret med udgangspunkt i overordnede mål.

MVC
`Model-View-Controller` er et design anvendt i programmering til at adskille data, logik og visuelle elementer på en måde, så det er mere overskueligt og tilgængeligt fremadrettet at skulle vedligeholde kildekoden.

MVVM
`Model View ViewModel` er et designmønster introduceret af Microsoft i forbindelse med WPF/Silverlight. MVVM er et adskillelsesmønster i GUI-baseret programmering som HTML5 og WPF/Silverlight.

Responsive Design
: `Responsive Design` er en tilgang til design af websiden, som skaber en optimal oplevelse og en brugervenlig navigation uanset hvilken enhed websiden besøges fra. Websiden tilpasser sig med andre ord dens omgivelser, og omstrukturerer indhold afhængig af enheden websiden besøges fra.

OOA/D
: `Objekt-Orienteret Analyse og Design` er en udviklingsmetode i objektorienteret programmering, som indeholder en række metoder og værktøjer til at udvikle software-systemer.

ORM
: `Object-Relational Mapping` er en programmeringsteknik til at konvertere data mellem objekter i et programmeringssprog og en relational database. En ORM gør det muligt at skrive sin programmeringslogik med objekter frem for i SQL.

POCO
: `Plain old CLR object`

REST
: `Representational State Transfer` eller REST er en måde at lave web services og SOA på som er meget HTTP orienteret. Data er repræsenteret ved URL'er. GET henter data. POST opdaterer data.

RIA
: `Rich Internet Application`

SPA
: `Single Page Application`

UML
: `Unified Modeling Language`

UP
: `Unified Process` er en populær iterativ og inkrementel udviklingprocess.

Bilag B

Litteratur

[1] Adapt A/S. Digital strategi, 1998 - 2013.

[2] Innologic Business Monitor a/s. En Dashboard & Budget løsning for e-conomic kunder, 2012 - 2013.

[3] Bear Bibeault and Yehuda Katz. *jQUERY IN ACTION*. Manning Publications Co., second edition, 2010.

[4] LARMAN Craig. *APPLYING UML AND PATTERNS*. John Wait, third edition, 2007.

[5] e-conomic danmark a/s. Online regnskabsprogram, 2002 - 2013.

[6] Ben Frain. *Responsive Web Design with HTML5 and CSS3*. Packt Publishing Ltd., first edition, 2012.

[7] Jim Jackson II and Ian Gilman. *HTML5 FOR .NET DEVELOPERS*. Manning Publications Co., first edition, 2013.

[8] Jeffrey Palermo, Jimmy Bogard, Eric Hexter, Matthew Hinze, and Jeremy Skinner. *ASP.NET MVC 4 IN ACTION*. Manning Publications Co., a revised edition of asp.net mvc 2 in action edition, 2012.

[9] Nicholas C. Zakas. *Professional JavaScript for Web Developers*. John Wiley & Sons, Inc., third edition, 2012.