Udvikling af Gadgets til Borger.dk

Martin Kirk

Og

Kristoffer Nielsen

Kgs. Lyngby 2007

IMM-B.Eng-2007-**65**

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Summary

Borger.dk is a digital representation of the public sector, produced by the cooperation of the ministry of science and KL. The portals purpose is to give easy access, to public services.
In 2008 the plan for the portal, is to expand the feature set with a ‘My Page’, which will have similar aspects to home banking. Different from home banking, the ‘My Page’ will let the user logon, using a digital signature. Afterwards he will have access to all the public services presented.
In order to offer these public services, a method of integration must be found. This has lead to this project ‘Development of Gadgets for Borger.dk’. The project span, is to investigate what means are needed for the service development to developing a prototype that proves it’s possible with the given set of requirements.

NNIT A/S is hired to expand and develop Borger.dk and have already made a huge part of the functionality of the portal. Amongst responsibilities, NNIT A/S also has the privilege to guide the decisions of The Digital Taskforce: an organisation that dictates the architecture of the system. Something new to Borger.dk (Q4, 2007) is a change in platform provider, from Oracle to Microsoft (.NET/SQL), which has influenced our choice of programming language. The prototype will therefore be developed in C# / ASP.NET.

The goal is to make a standard, which in the same way as Google Gadgets, will provide guidance and a toolset, for public service developers to develop their services.

Resumé

Borger.dk er en digital indgang til den offentlige sektor, der bliver tilbudt i samarbejde med Videnskabsministeriet og KL. Sidens formål har til hensigt på en overskuelig måde, at give den enkelte borger adgang til det offentlige.
I 2008 skal borger.dk udvides til at indeholde "Min Side", der skal minde om en netbank [1].
I modsætning til en netbank, skal brugeren her logge på med sin digitale signatur og bagefter have adgang til digitale selvbetjeningsløsninger, også kaldet services eller Gadgets[1].
For at tilbyde de digitale selvbetjeningsløsninger, skal der findes en måde at tilføje disse services på en såkaldt "Min Side", som derfor har skabt grundlaget for vores opgave "Udvikling af Gadgets". Opgaven spænder over at undersøge de aspekter der skal til, for at udvikle en komponent som kan bruges på Borger.dk, til udvikling af en prototype som beviser at det kan lade sig gøre med de stillede krav.

NNIT A/S har til opgave at udvide Borger.dk og står allerede for en stor del af funktionaliteten af portalen. NNIT A/S er med til at præge de beslutninger der tages af "Den Digitale Taskforce" som overordnet tager beslutningerne for hvordan Borger.dk skal udvikles. Helt nyt for borger.dk (4. kvartal 2007) er et leverandørskifte til Microsoft (.NET/MSSQL) som har haft betydning for valget af programmeringssprog. Gadget prototypen vil derfor blive udviklet i C# / ASP.NET.

Målet er at lave en standard som på samme måde som Google Gadgets vil give udviklere fra forskellige serviceudbydere (så som Skat.dk), et sæt af retningslinier for hvordan de kan designe services til "Min Side".

[1] Se afsnittet begreber omkring services og Gadgets

Forord

Denne rapport og præsentationen af produktet, er afslutningen på vores uddannelse som IT Diplomingeniør på DTU. Opgaven er lavet med vejledning af docent Robin Sharp (IMM-DTU) og i samarbejde med NNIT A/S. Hos NNIT har Morten Post Lüneborg været kontakt-person og projekt/praktik vejleder. Mange tak til dem begge

Projektet omhandler de overvejelser der skal gøres, for at udvikle sin egen Gadget standard. Hensigten med opgaven er at løsningen skal benyttes på den offentlige borgerportal og skal derfor leve op til et sæt krav om bl.a. tilgængelighed og sikkerhed. Det er meningen at portalen skal logge ind på vegne af brugeren, for derved at minimere behovet for at taste brugernavn, adgangskode eller benytte digital signatur mere end én gang. Gadget siden på ”Min Side” vil åbne op for en nemmere tilgang til hurtige og nemme services på tværs af den offentlige Internetsektor. Produktet vil dels være en implementering som viser konceptet, dels være en analyse som belyser de aspekter der skal håndteres ved senere udvidelse.

Indholdsfortegnelse

Kapitel 1

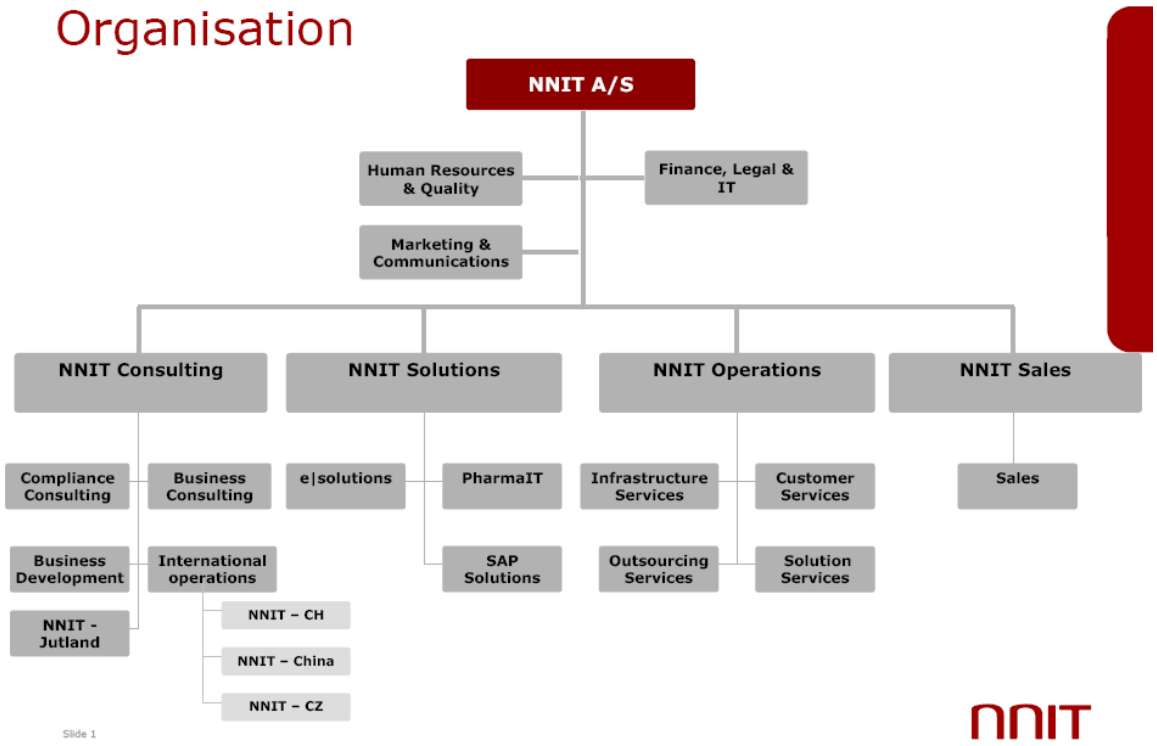
1. Introduktion

Dette kapitel vil give en introduktion til dels NNIT og dels projektets grundlag. Kapitlet vil også beskrive NNIT's vision med projektet, samt beskrive projektplanlægningen.

1.1. NNIT A/S

NNIT er et af Danmarks ledende konsulentfirmaer i IT udvikling, implementering og håndtering. I dag har NNIT mere end 1000 ansatte og en omsætning på over 1 mia. og sidste år havde de en vækst på 14 %.

NNIT A/S er et datterselskab af Novo Nordisk A/S. Novo Nordisk er stadig den største kunde for NNIT, men der kommer hele tiden flere og flere opgaver til NNIT fra andre kunder. Den offentlige sektor (herunder IT & Telestyrelsen: borger.dk) er i dag også blevet en stor kunde for NNIT.



Figur 1 - NNITs organisation

1.2. Borger.dk

Regeringen, Amtsrådsforeningen, KL, Københavns Kommune og Frederiksberg Kommune iværksatte i 2001 Projekt Digital Forvaltning, som har til formål at fremme omstillingen til digital forvaltning i Danmark. Projektet blev iværksat for en treårig periode og ledes af en fælles bestyrelse, kaldet Den Digital Taskforce (DTF). DTF har til opgave at fremme omstillingen til digital forvaltning på tværs af den offentlige sektor. DTF har fungeret som katalysator for løsning af tværgående koordinationsproblemer i digitaliseringsprocessen.

I forbindelse med økonomiaftalerne mellem regeringen og de kommunale parter i 2003 blev det imidlertid aftalt at videreføre Projekt Digital Forvaltning til udgangen af 2006 med sekretariatsbetjening af DTF. Parterne besluttede i 2007 at videreføre Projekt Digital Forvaltning frem til 2010.

Projekt Digital Forvaltning indeholder som løsning, bl.a. Borger.dk projektet som ejes og udliciteres af IT & Telestyrelsen. DTF har under analysefasen stået for at samle kravene fra de individuelle instanser, for derved at kunne aflevere en projekt plan til IT & Telestyrelsen.

Borger.dk projektet er i 2007, blevet opdelt i 2 versioner:

1.2.1. Version 1

Bestod af den indledende udvikling af borger.dk portalen og indeholdte kun basale funktioner, portalen blev udviklet af forskellige leverandører.
Version 1 som man kender den i dag, bliver kun vedligeholdt og opdateret med indhold indtil version 2 er klar til levering.

1.2.2. Version 2 (Borger.dk 2008)

Indeholder en række nye tiltag til portalen: ”Min Side”, et nyt Redaktørmiljø, forbedret Emneordsopbygning, Borgertemaer, Kommuneindhold – kort sagt mere personalisering overfor brugeren af siden og et udbud af flere services.

Arbejdet med at udvikle Portalen og "Min Side" er delt op i fem faser:

1. Planlægningsfasen
2. Konceptudviklingsfasen
3. Kravspecifikationsfasen
4. Testfasen
5. Lanceringsfasen

Det er i konceptudviklingsfasen at dette projekt kommer ind i billedet. konceptudviklingsfasens formål er at samle viden, som DTF vil bruge til det endelige projekt oplæg til IT & Telestyrelsen. Kommunerne har bl.a. udført et større Proof of Concepts (PoC) med henblik på, at finde løsningen på et fælles login system. Projektet kom blandt andet ind på de problemstillinger der er ved at benytte forskellige platforme og blandt projektets konklusioner var en hængende problemstilling overfor remote services, som ikke kunne løses med daværende produkter.

1.3. Projekt formulering

Forudgående for Version 2 er der lavet en række projekter som havde til formål at teste allerede eksisterende produkter. Ingen af disse produkter er til dato blevet valgt, grundet forskellige mangler.
Til borger.dk projektet ønskes et løsningsforslag til indlejring af services, som kan konkurrere med de allerede afprøvede produkter. Løsningsforslaget skal tage stilling til sikkerhed, tilgængelighed, funktionalitet og hastighed.

1.4. Projekt afgrænsning

Projektet har til formål at undersøge de aspekter, der er ved, at udvikle et system til visning af services på en portal. Forudgående for projektet ligger en række produktundersøgelser af sammenlignelige løsninger, som vores løsning skal sammenlignes med.
Produktet bliver en portal-komponent som kan indlejre services fra eksterne ressourcer. indlejringen skal ske sådan, at servicen stadig er interaktiv på portalen og indlejret på en sådan måde, at tilgængelighedsaspekterne vedholdes. Produktet skal endvidere leve op til de krav der er om sikkerhed på en offentlig portal.
Version 2 af borger.dk vil få et Single Sign On system, men da det ikke er klar til hverken brug eller afprøvning i dag, kan det ikke inkluderes i projektet.

1.5. Begreber

1.5.1. Portalen (consumer)

Portalen skal følge ”glaspladeprincippet”, det vil sige at portalen kun er et samlested for services og informationer, til rådighed for brugeren. Portalen skal således ikke lave tunge beregninger, eller gemme oplysninger om brugeren, men kun sørge for at brugeren får adgang til relevante services.
Ved at logge på systemet og derved automatisk blive logget på de tilmeldte service udbydere, vil brugeren få adgang til personlige informationer og mulighed for at udføre service-specifikke transaktioner i de tilmeldte Gadgets. F.eks. opdatering af informationer til SU styrelsen.

1.5.2. Identity Provider

Identity Provider (også kaldet IDP) er et system, der tager hånd om brugeren der logger på med et brugernavn og en adgangskode. Hvis brugeren er kendt af systemet, returnerer systemet en profil, hvor der står at brugeren er kendt. Det er således IDP’en der vurderer om brugeren, er den vedkommende udgiver sig som, og hvis en service udbyder eller portalen har godkendt IDP’en, har de også godkendt de brugere IDP’en udsteder profiler til.

1.5.3. Serviceudbyder (producer)

En serviceudbyder er en offentlig instans, der stiller en service til rådighed via borgerportalen. Portalen kender samtlige serviceudbydere på forhånd og kender deres krav om sikkerhed.
Serviceudbyderen er stillet med et sæt regler for hvordan servicen kan blive optaget på portalen, bl.a. hvordan kommunikationen skal fungerer, samt hvilke regler serviceudbyderen skal overholde.
For at bruge en service fra en serviceudbyder, kræves det at der logges på ved hjælp af en Identity Provider. Det kræver derfor at serviceudbyderen, tror på identiteten af en bruger, der er blevet fastsat af IDPen.

1.5.4. Service

En mini-hjemmeside som kan afvikles i et lille vindue: iFrame eller Gadget. En service er en forenklet udgave af serviceudbyderens normale hjemmeside, som repræsenterer den offentlige instans.

1.5.5. Gadget

Ved en Gadget forstås en service tilføjet på portalen. Både betegnelsen Gadget og service vil blive brugt i rapporten.

1.5.6. Brugeren (end user)

Brugergruppen som vil benytte sig af portalen, kan både være en privat person, eller en medarbejder i et firma. Fælles for brugerne er at de alle skal logge på ved hjælp af en identity provider, der laver en bruger profil til dem.

1.6. Rapport skitse

Projekt rapporten er bygget op på følgende måde:

Resume

Giver en abstraktion over projektet

1. Introduktion

Her gives en introduktion til projektet med baggrund for projektet og hvad der vil blive fokuseret på, i dette projekt.

2. Projekt planlægning

Her gives en introduktion til valg af udviklings proces i vores projekt og en plan for projektforløbet.

3. Krav

Her gives et overblik over krav i for dette projekt.

4. Analyse

Her gives en analyse af de eksisterende løsninger for vores problemstilling, og der vises use-cases for vores projekt.

5. Design

Her vises sekvens diagrammer for de enkelte use cases og klasse diagram for systemet.

6. Implementation

De implementerede dele beskrives.

7. Test

Her gives en beskrivelse af de test, som er blevet brugt.

8. Konklusion

Her gives en konklusion på projekt, hvor der evalueres på produktet.

9. Litteraturliste

Indeholder referencer på bøger og sider, som er brugt i dette projekt.

10. Bilag

Indeholder relevante dokumenter.

Kapitel 2

2. Projekt planlægning

I projektplanlægningen vil vi fortælle om hvilken udviklingsmetode vi har valgt at bruge, og hvorfor vi har valgt den. Vi vil fortælle om de metoder fra udviklingsmetoden vi har brugt.

2.1. Valg af udviklingsmetode

2.1.1. Unified Process

Til udvikling af vores projekt har vi valgt at bruge udviklingsmønstret Unified Process, da vi har brugt den arbejdsmetode før, og der er en række fordele ved den, som vi kan drage nytte af.

Unified Process er en populær objektorienteret softwareudviklingsproces. Den er baseret på iterativ udvikling. Det er ikke meningen at Unified Process skal følges til punkt og prikke, men den er lavet som et skelet der kan udformes til det enkelte projekt. [11]

Fordele:

- Det er muligt at ændre i kravene undervejs.
- Man kan hurtigt se fremgang i projektet.
- Man får lidt erfaring efter hver iteration.
- Det er lettere at nedbryde komplekse opgaver.
- Efter hver iteration er der sket en udvikling af projektet.

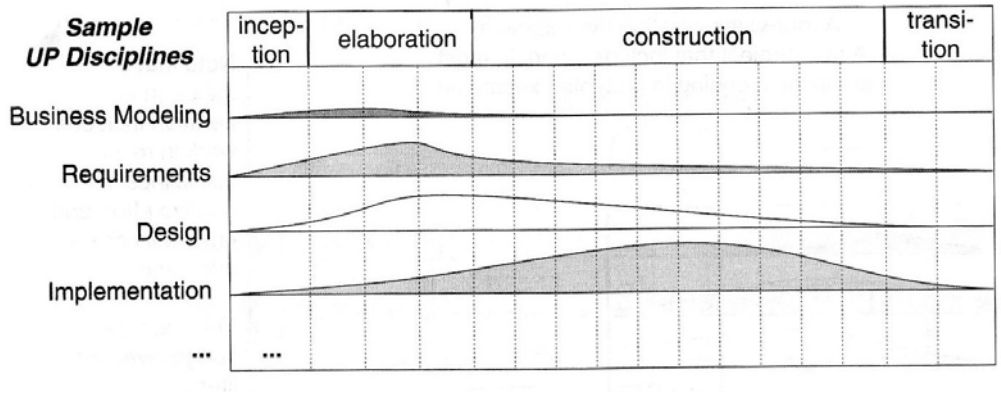
Ulemper

- Det tager tid at lave ændringer i de forskellige diagrammer.
- Man kan lave et design i UML der ikke kan implementeres i praksis.

Da vi har forholdsvis kort tid, til et stort projekt, er det en fordel med en udviklingsmetode der består af korte iterationer. Hvis vi efter en iteration finder ud af at en proces ikke kan bruges eller lade sig gøre, kan vi stadigvæk nå en ny iteration.
Meningen ved en iteration er, at man får et produkt ud af det, der kan mindre end det ønskede produkt. Man beskæftiger sig altså med nye krav i hver iteration, der hele tiden vil udvide systemets funktionalitet.

2.1.2. Unified Process faser

Unified Process er delt op i flere iterationer, fordelt over 4 faser. Det ser således ud:



Figur 2 - unified process faser [12]

På Figur 2 ses det, at der i starten lægges mest vægt på forretningsmodellen, samt krav til projektet, mens der senere bliver lagt mere vægt på design og implementation.

Inception (forberedelse):
I forberedelsesfasen skal man analysere de kritiske krav og fastslå de grundlæggende ideer om systemet, samt lave en afgrænsning for projektet.
Der udarbejdes use cases for projektet. Alle krav defineres ikke i denne fase, men målet er at finde ud af om projektet kan lade sig gøre, og om det kan betale sig at fortsætte til næste fase.

Elaboration (udarbejdelse):
Under udarbejdelsesfasen begynder projektet at tage form. Her tages fat på høj risiko emner i projektet, og der laves en analyse af systemet. En projektplan udarbejdes i denne fase. Ved slutningen af denne fase, vil de fleste krav være fundet.

Construction (konstruktion):
Det er i konstruktionsfasen at systemet udvikles og testes. Alle de produkter der har stor værdi for kunden og systemet laves her. Der laves brugervejledning og dokumentation i denne fase.

Transition (overgang):
I overgangsfasen testes projektet, og systemet bliver sat op. Der foretages små justeringer og efter denne fase, er projektet kar til at blive afleveret til slutbrugeren.

2.1.3. UML

Unified Modeling Language er en standard der beskriver hvordan diagrammer til et software objekt orienteret system ser ud. Det skal gøre det lettere for programmøren at gå i gang med at udvikle sit produkt, når der er lavet diagrammer, der beskriver forløbet.
Vi vil bruge følgende UML diagrammer:

Use case:	Beskriver hvordan aktørerne bruger systemet.
System sekvens diagram:	Viser, hvordan aktørerne kommunikere med systemet
Klasse diagram:	beskriver sammenhængen mellem de forskellige klasser, samt hvilke metoder de enkelte klasser indeholder.
Sekvens diagram:	viser interaktionen mellem klasserne i systemet, med hvilke klasser der bliver brugt.

2.1.4. Projektplan

I starten af projektet udarbejdede vi en projektplan, vi kunne følge. Den er ikke lavet som en fuldstændig projektplan, der skal følges til punkt og prikke, da meget kan ændre sig undervejs i projektet. Den er beregnet som et udkast, vi kan bestræbe os på at følge.

Faser	**Iteration**	**Handling**	**Uge**
Krav	Inception	Finder ud af hvad systemet skal kunne, samt hvad der er forsøgt indtil videre. Udførelse af iterations plan.	1
Krav Analyse	1 iteration	Beskrivelse af usecases.	2
Design	1 iteration	Der udarbejdes et klassediagram, samt sekvens diagrammer for usecases.	3
Design, implementering	2 iteration	Implementering, rettelse af usecases og klassediagram.	4
Design Impelementation Test	3 iteration	Der fortsættes med implementering.	5
Design Impelementation Test	4 iteration	Der fortsættes med implementering, samt testes for om usecases er overholdt.	6
Afslutning	5 iteration	Retter design til, samt afslutter implementering.	7
Afslutning	6 iteration	Retter små justeringer / Rapport	8
Afslutning	6 iteration	Rapport	9
Afslutning	6 iteration	Rapport	10
Aflevering	6 iteration	Rapport afleveres 3. december	11

2.2. Resume

Vi har fået begrundet vores valg af Unified Process, samt hvordan den kan bruges, og vi har tænkt os at bruge den. Til sidst har vi lavet et udkast til vores projektplan for vores bachelor projekt.

Kapitel 3

3. Krav

3.1. Kravspecifikation for services på borger.dk

Borger.dk skal over de næste par år udvides med mere funktionalitet for brugerne (borgere i samfundet). Blandt særligt nyttige funktioner er der planer om en ”Min Side”, hvor brugeren skal have adgang til services fra forskellige dele af den offentligt tilgængelige sektor.

Folk som ikke har handicaps, vil i et givent omfang være i stand til at udnytte mere komplekse elementer på portalen og der tænkes derfor på at optimere portalen til begge parter. Endvidere skal portalen kunne åbnes og benyttes af de mest benyttede browsere, således at besøgende ikke vil opleve fejl eller på anden måde være nægtet adgang til portalen eller de Gadgets som eksisterer.

3.1.1. krav om visuel integration

”Min Side” er ikke tiltænkt at erstatte de eksisterende løsninger, men at supplere dem med en lettere og mere overskuelig oversigt, og hvor brugeren automatisk er logget ind på de services, han/hun allerede er tilmeldt eller pr. definition har adgang til (f.eks. Skat). Systemet bør ikke kræve yderligere legitimation, idet brugeren er logget på portalen med digital signatur og derved benytter den Single Sign-On (SSO) arkitektur som er planlagt for portalen. Skulle en serviceudbyder alligevel kræve, at brugeren sender nogle bekræftende oplysninger, som f.eks. *Captcha* eller et fortroligt id af en art, skal systemet så vidt muligt understøtte disse.

1. Services skal ’flyde sammen’ med borger.dk så de præsenteres som en del af borger.dk
2. Det er vigtigt at services optræder og føles som en del af Borger.dk og ikke blot iFrames som behandler indholdet af en side eksternt og asynkront.
3. Som udgangspunkt bør de services som brugeren er tilmeldt være synlige fra starten, mens nogle services kan være valgfrie.
4. Løsningen skal integreres med et fælles login system (SSO, SSL).
5. Der skal være mulighed for at brugeren kan poste formularer igennem systemet og ned til serviceudbyderen.

3.1.2. krav om valgfri arkitektur

Idet Portalen ikke nødvendigvis er bygget på samme arkitektur som alle de services, der kan være tilgængelige, er det derfor et krav at systemet ikke er baseret på, kun at understøtte samme arkitektur. Serviceudbyderne bør derfor have muligheder for selv at vælge deres platform.

1. Services skal kunne bygges på deres egen arkitektur (Java, .NET, ASP, PHP osv.)

3.1.3. krav om tilgængelighed

Portalen er tiltænkt at kunne benyttes af alle, som er i stand til at benytte Internettet til offentlige gøremål. Dette gælder således også svagtseende eller andre handicappede, som benytter sig af software som læser op fra skærmens indhold.

1. Borger.dk såvel som Gadgets bør overholde Web Accessibility Initiative (WAI) [14] niveau AA.
2. For at gøre siden optimal for skærmlæsere, må siden ikke indeholde iFrames eller scripts som danner indhold.

3.1.4. krav om fortrolighed og sikkerhed

Portalen skal opfylde det samme fortrolighedsniveau som de services der skal udbydes. ”Min Side” skal opfylde det krav der i dag betragtes som værende ”sikker nok”, til at brugeren kan sende fortrolige oplysninger igennem portalen og ud til serviceudbyderen. Desuden er der krav om, at Portalen følger ”glasplade princippet”: At der ikke gemmes oplysninger og at portalen kun er en præsentations side.

1. Service over portalen, skal være så sikkert som fra bruger til service direkte.
2. Portalen må ikke gemme fortrolige oplysninger
3. Services skal have en stærk sikkerhedsrelation til borger.dk hvilket betyder at services ikke kan benyttes uden det sker direkte igennem borger.dk såvel som at borger.dk kun benytter betroede services.

3.1.5. krav om performance

Service komponenten skal være en del af Portalen og det er derfor et krav, at mange brugere uafhængigt af hinanden, kan benytte sig af ”Min Side”. Dette, uden fare for at deres data blandes, eller at oplevelsen forringes af antallet af samtidige brugere.

1. Løsningen skal implementeres så den er ’let’ og derved kan levere services til flere samtidige brugere.
2. Det skal sikres at data bliver sendt til den rigtige serviceudbyder, ligesom det skal sikres at brugeren får det rigtige data retur, og kun sine egne data.

3.2. Krav fra NNIT til projektet

NNIT’s største interesse med projektet, er at få belyst nogle af de aspekter, der ikke normalt er afsat tid til at undersøge. Den største problemstilling er at der ikke findes et passende produkt som kan bruges til visning af Gadgets på portalen, WSRP og JSR-168 har en hel del af faciliteterne til rådighed, men er ikke ideelle.

Idet projektet er et eksamensprojekt som ikke med sikkerhed kan benyttes af NNIT, har vi relativt frie hænder til at producere et løsningsforslag. NNIT’s succeskriterium er et forslag som ideelt set kan benyttes direkte som løsning i borger.dk projektet.

3.3. Udvikling

3.3.1. Værktøjer

Til udvikling af vores eksamensopgave, benytter vi os af Microsoft Visual Studio 2005 professional og C# til at lave vores Gadgets til borger.dk.
Til tegningerne benyttes Visio samt Visual Studio (klassediagram).

3.3.2. Test

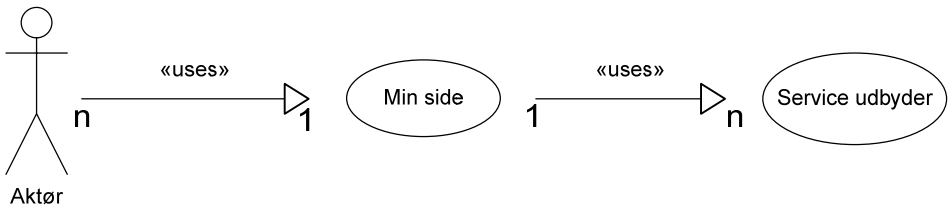
Til test vil vi gøre brug af Smoke-test [13]. Smoke-test er en løbende form for test, der skal afsløre almindelige fejl, som vil gøre at produktet fejler. Smoke-test kan både udføres auto-

matisk af systemet, eller som i vores tilfælde af udviklerne. Efter kode review er Smoke-test den mest effektive måde, til at afsløre fejl i software.
Derudover har vi testet vores produkt undervejs, når vi har lavet nye implementeringer, for at se om det kunne hente vores Gadgets, samt at det var muligt at benytte Gadgets.

3.4. Use case model

3.4.1. Identifikation af aktører

Vi har en aktør der vil bruge vores portal, der repræsenterer den almene borger. Grunden til at vi kun har én aktør er, at der ikke skal være forskel på, hvad forskellige borgere vil se på ”min side”. Aktøren kan dog både være en privat person, eller en person der logger på i forbindelse med et firma.



Figur 3 - viser aktøren der bruger min side, som bruger service udbyderen

3.5. Resume

I dette kapitel er det blevet beskrevet hvilke krav der er stillet til projektet og hvilke krav vi har tænkt os at beskæftige os med. Det er ligeledes beskrevet hvilke parter som indgår i systemet

Kapitel 4

4. Analyse

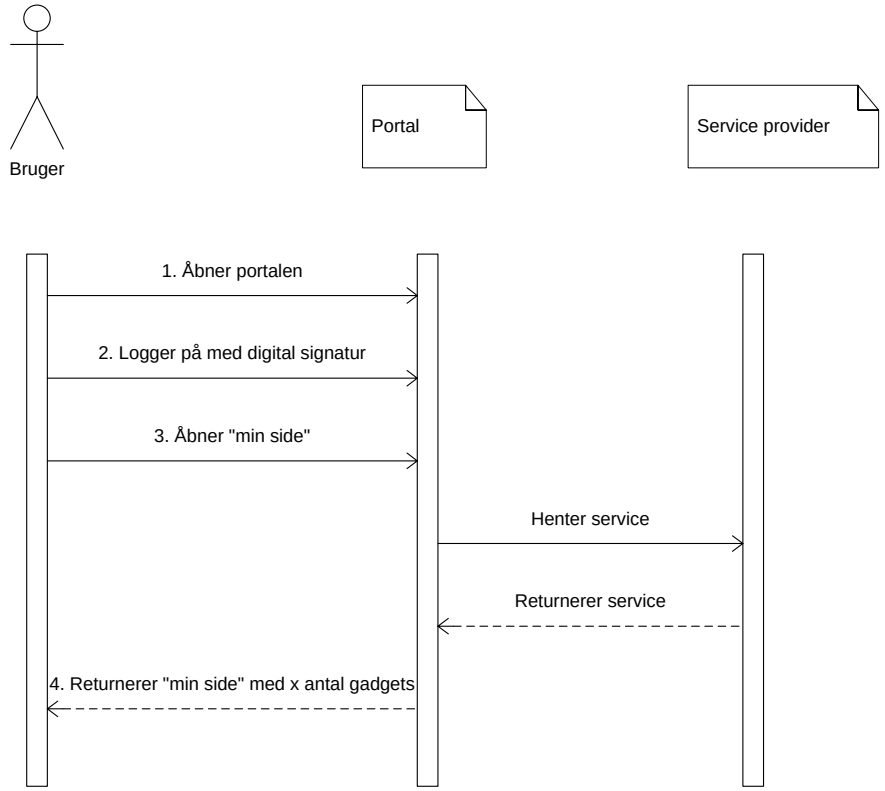
I vores analyse har vi tænkt os at komme ind på hvordan man vil kunne lave en sikker forbindelse til en service. Vi vil fortælle om hvilke eksisterende løsninger der findes i dag, samt hvordan vi har tænkt os at klare problemet med at tilknytte services til en portal. Vi vil udarbejde forskellige usecase scenarier, samt system sekvens diagrammer der viser hvordan vi har tænkt os at interaktionen mellem bruger og system vil være.

4.1. Usecases og system sekvens diagrammer

Vores usecases beskriver målsætningen ved en given handling for en aktør. Der kræves nogle forudsætninger for handlinger, derudover stilles der nogle succeskriterier for en vellykket handling, samt en beskrivelse af hvornår en kørsel anses som fejlet.

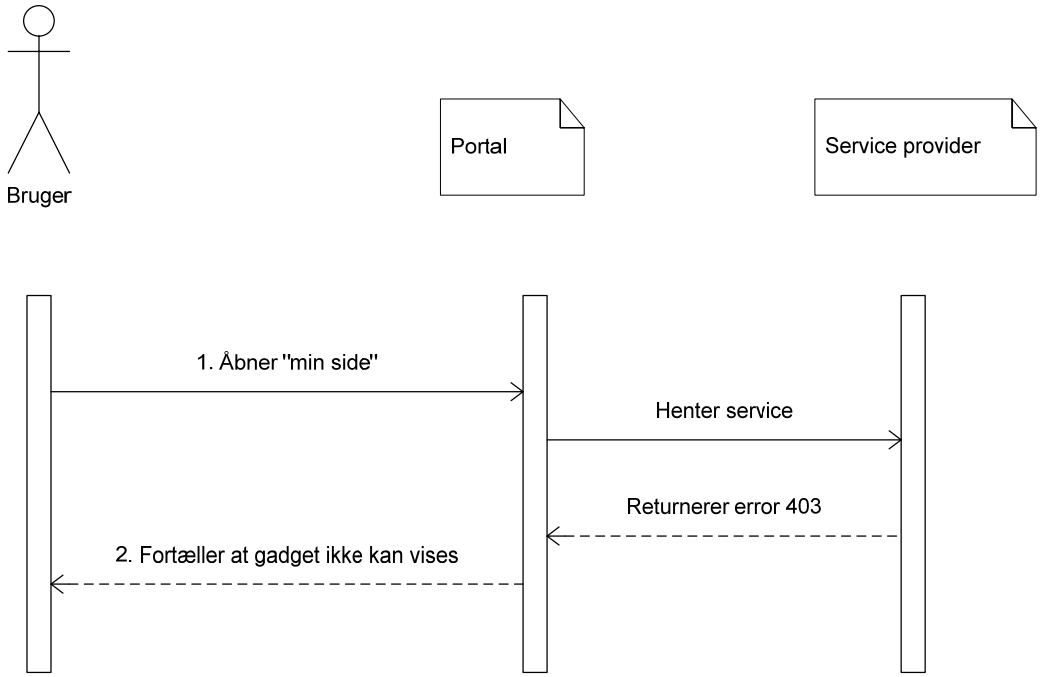
System sekvens diagrammerne viser brugerens interaktion med systemet.

USE CASE 1	Brugeren åbner portal med Gadgets	
Målsætning	Portalen åbner siden og beder om login hvis dette er påkrævet, og viser derefter 1 til N antal Gadgets på siden. Gadgets vises som en del af portalen og loades samtidig	
Scope		
Forudsætninger	Brugeren findes i systemet og har adgang til Gadget siden	
Succeskriterier	Portalen viser en side med N antal Gadgets som viser nogle relevante informationer	
Fejlet kørsel	Portalen viser en fejlside eller laver timeout grundet fejl – eller der vises en Gadget som ikke tilhører brugeren	
Primær Aktør	Slutbrugeren	
Sekundær Aktør	Normalt ingen, idet systemet kører automatisk	
Udløser	Brugeren har klikket på linket som viser siden	
Beskrivelse	Trin	Handling
	1	Brugeren åbner portalsiden ved at skrive linket i en browser
	2	Brugeren logger på med digital signatur eller lign autentifikation
	3	Brugeren åbner siden med Gadgets
	4	Der vises 1 til N antal Gadgets



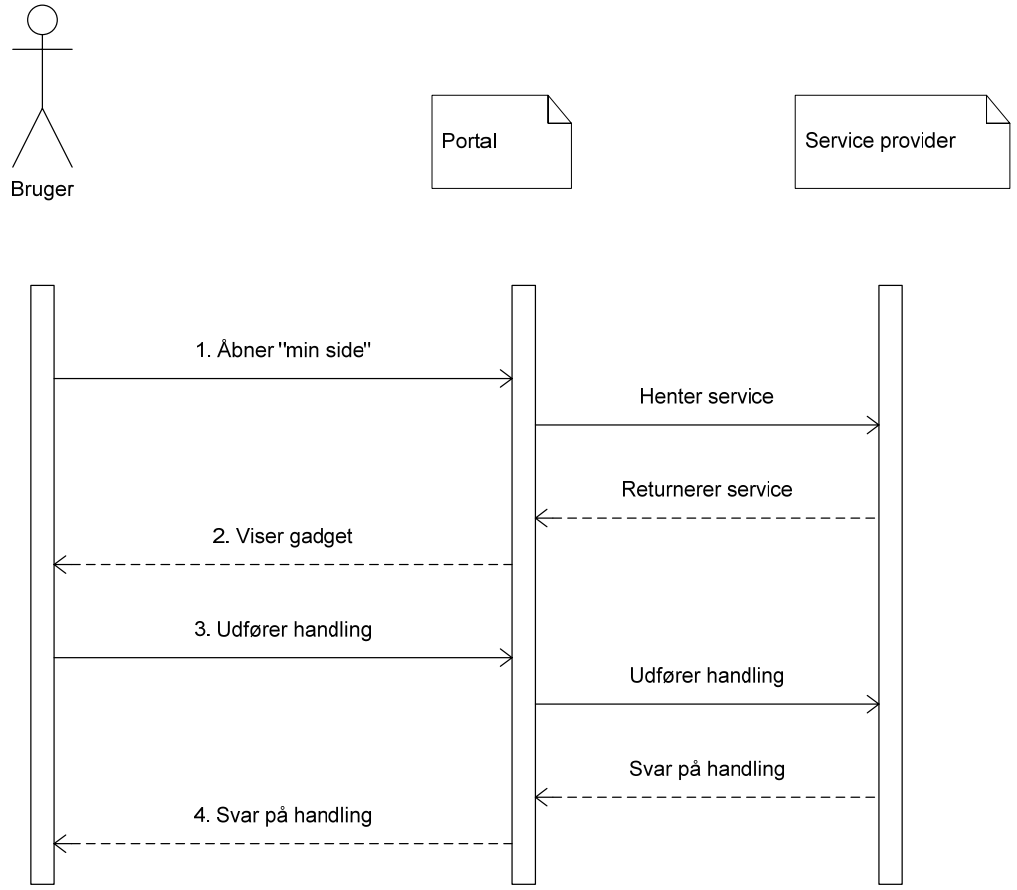
Figur 4 - viser hvordan brugeren kommer ind på min side.

USE CASE 2	Portalen viser ikke en Gadget	
Målsætning	Portalen skal ikke vise Gadget, fordi der ikke er adgang	
Scope		
Forudsætninger	Brugeren er logget på portalen	
Succeskriterier	Portalen viser kun Gadgets som brugeren har adgang til	
Fejlet kørsel	Der vises Gadgets som brugeren ikke har adgang til Der sker fejl i systemet og siden vises ufuldstændig Der kommer timeout fejl Intet sker	
Primær Aktør	Slutbrugeren	
Sekundær Aktør	Normalt ingen, idet systemet kører automatisk	
Udløser	Brugeren åbner siden med Gadgets	
Beskrivelse	Trin	Handling
	1	Brugeren trykker sig ind på Gadget siden
	2	Portalen viser at Gadget ikke kan vises, fordi der er opstået en fejl.



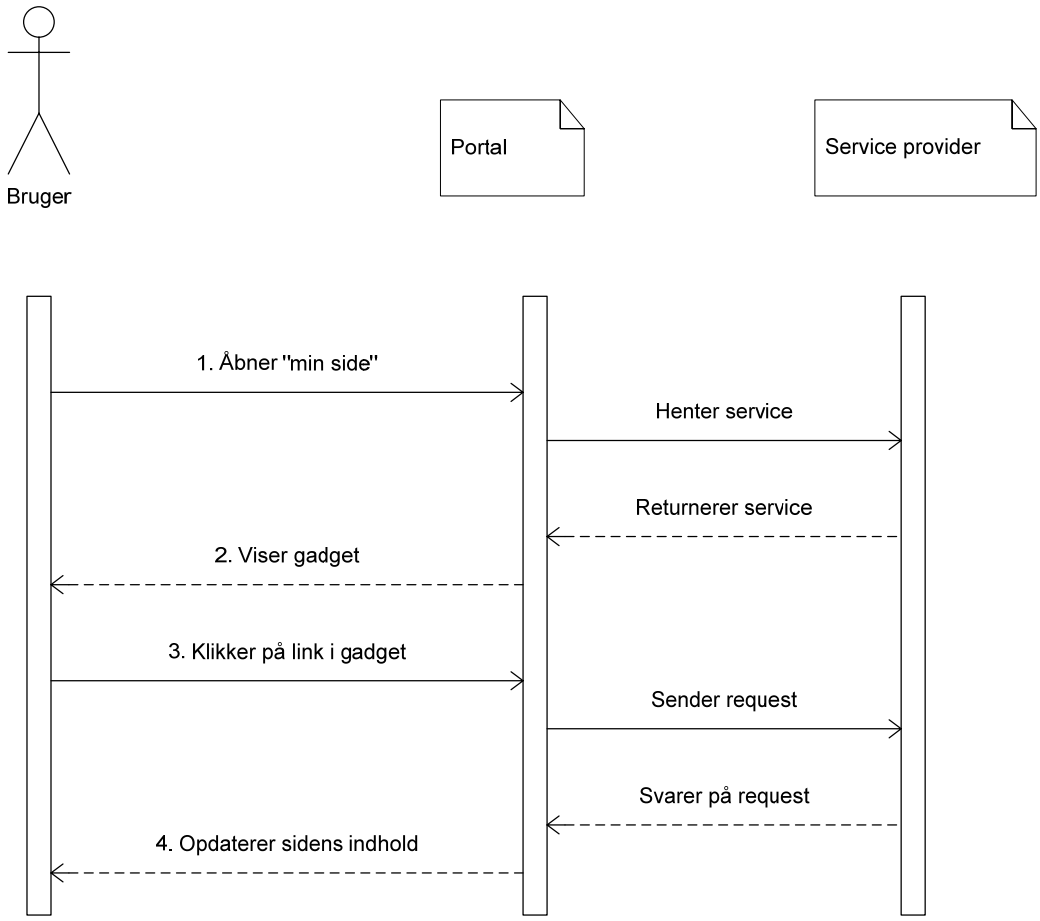
Figur 5 - viser handlingsforløbet, hvis der ikke er adgang til servicen.

USE CASE 3	Brugeren udfører en handling i en Gadget	
Målsætning	Portalen udfører handlingen i den aktive Gadget	
Scope		
Forudsætninger	Brugeren er logget på	
Succeskriterier	Portalen reloader siden helt eller delvist, hvor det eneste område som ændres er den aktive Gadget	
Fejlet kørsel	Andre elementer på siden ændres (med mindre det er meningen) Gadgetten flytter position eller forsvinder Brugeren logges ud Intet sker	
Primær Aktør	Slutbrugeren	
Sekundær Aktør	Normalt ingen, idet systemet kører automatisk	
Udløser	Brugeren klikker på et element i en Gadget	
Beskrivelse	Trin	Handling
	1	Brugeren åbner ”min side”
	2	Portalen viser en Gadget
	3	Brugeren udfører en handling i en Gadget
	4	Hele portalsiden reloader hvorefter den aktive Gadget viser noget ændret indhold



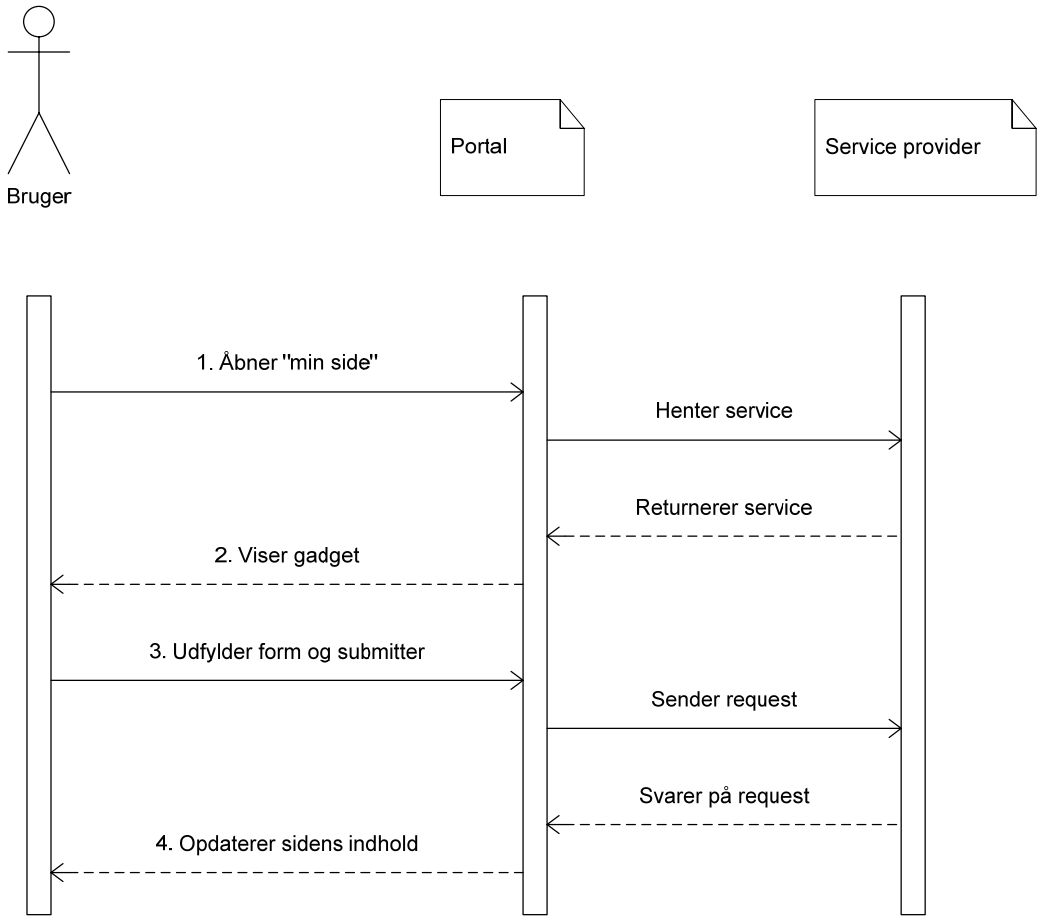
Figur 6 - udfører en handling på en Gadget.

USE CASE 3.1	Brugeren klikker på Gadget-link til ”næste side”	
Målsætning	Gadgetten viser noget nyt indhold	
Scope		
Forudsætninger	Brugeren er logget på systemet og har en Gadget på portalen	
Succeskriterier	Noget nyt indhold vises i Gadgetten	
Fejlet kørsel	Det samme indhold vises i Gadgetten eller der sker fejl	
Primær Aktør	Slutbrugeren	
Sekundær Aktør	Normalt ingen, idet systemet kører automatisk	
Udløser	Brugeren klikker på et Gadget link	
Beskrivelse	Trin	Handling
	1	Brugeren åbner ”min side”
	2	Gadget vises for brugeren.
	3	Brugerne klikker på et link
	4	Portalen opdaterer siden med nyt indhold



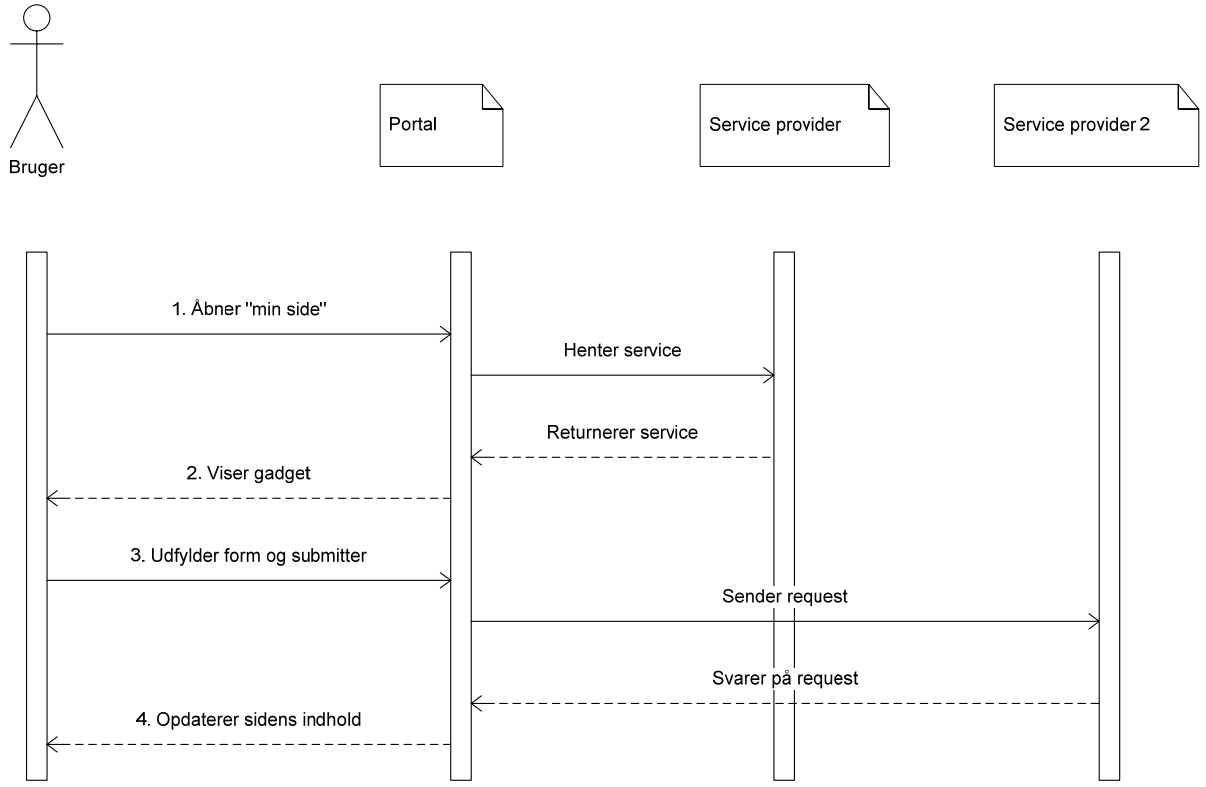
Figur 7 - et link i en Gadget følges.

USE CASE 3.2	Brugeren udfylder en formular på en Gadget	
Målsætning	Formularen udfyldes af brugeren og sendes af sted til serviceudbyderen	
Scope		
Forudsætninger	Brugeren har adgang til Gadgetten	
Succeskriterier	Dataene i formularen sendes af sted og gemmes hos serviceudbyderen	
Fejlet kørsel	Dataene kommer ikke frem til serviceudbyderen	
Primær Aktør	Slutbrugeren	
Sekundær Aktør	Normalt ingen, idet systemet kører automatisk	
Udløser	Formularen sendes (tryk på "submit")	
Beskrivelse	Trin	Handling
	1	Brugeren åbner ”min side”
	2	Gadget vises for brugeren.
	3	Brugeren udfylder en formular
	4	Gadgetten viser et svar på modtagelse



Figur 8 - der udfyldes en form, i en Gadget, som submittes.

USE CASE 4	To serviceudbydere arbejder sammen	
Målsætning	To eller flere serviceudbydere har indgået en aftale om vidensdeling, med henblik på at yde bedre service overfor brugeren. Formålet er at en procedure, som brugeren normalt skal overføre fysisk imellem serviceudbydere, kan ordnes igennem en Gadget.	
Scope		
Forudsætninger	Brugeren har adgang til begge services	
Succeskriterier	Proceduren udføres	
Fejlet kørsel	En fejl i systemet indtræder som medfører at proceduren ikke udføres	
Primær Aktør	Slutbrugeren	
Sekundær Aktør	Normalt ingen, idet systemet kører automatisk	
Udløser	en ”samarbejds-knap” på Gadgetten aktiverer relationen	
Beskrivelse	Trin	Handling
	1	Brugeren åbner ”min side”
	2	Gadget vises for brugeren.
	3	Brugeren udfylder en formular, der skal sendes til en anden service udbyder
	4	Gadgetten viser et svar på modtagelse



Figur 9 - handlingsforløbet når to serviceudbydere skal kommunikere med hinanden.

4.2. Autentifikation og Autorisation

Disse to begreber vil for de fleste betyde det samme, men er i sagens natur to sider af samme sag. Idet Borger.dk i fremtiden kommer til at håndtere begge begreber, vil vi her forklare forskellen og brugen af disse.

4.2.1. Autentifikation

Autentifikation fortæller ikke noget om hvad brugeren må, men bekræfter og beviser hvem brugeren er.

I dag (november 2007) er der implementeret en simpel autentifikation på borger.dk, borgere kan logge på ved hjælp af Digital Signatur. Den digitale signatur er signeret og er blevet udstedt af en betroet instans i samfundet, dette være sig TDC eller andre. Signaturen bekræfter at brugeren som logger på, ER hvem han siger han er. Mekanismen fungere fordi signaturen indeholder en krypteret nøgle og en hash af nøglens indhold – Certifikat Autoriteten (CA) har en kopi af certifikatet og kan ved hjælp af nøgle og hash verificere at certifikatet er godkendt.

4.2.2. Autorisation

Autorisation er det aspekt ved login eller ”tilkendegivelse” af brugeren, som gør ham i stand til at gøre ting. I et virksomhedsdomæne vil autorisations-niveau som administrator gøre personen i stand til, at administrere hele eller dele af domænet. Det kan også være som superbruger, som ofte har adgang til at se ting, men ikke må ændre dem.

Autorisation sker på basis af de oplysninger man opgiver, eller som findes i systemet om brugeren. Ser man på en virksomhed, har alle brugerne en konto på hovedserveren, som fortæller hvad de må og ikke må, men kræver at de bliver oprettet på forhånd, før de kan logge ind.

Overført til en portal, vil man normalt have en database, hvori alle kan registrere en konto, som på basis af de givne rettigheder, får lov til at administrere et domæne. Dette kan være så simpelt som administration af deres egen konto, og op til alle konti på portalen.

Autorisation kan også ske på basis af faktuelle oplysninger, som uomtvisteligt godkendes af CA'en. Oplysninger som f.eks. rolle i et firma som bemyndiger adgang til bestemte ressourcer.

Idet autorisation sker på basis af de oplysninger, som man kan tilkendegive via sit certifikat eller "Token", er det derfor vigtigt at kunne beskrive brugeren rigt, hvilket har ført til udviklingen af SAML.

4.2.3. SAML og SSO

SAML står for Security Assertion Markup Language [8] [9] og en en xml standard der bruges i forbindelse med udveksling af autentifikation og autorisation mellem domæner. SAML er en standard der består af en række profiler, hvor man kan vælge de profiler man har brug for. Ved at bruge en SAML profil, sikrer man at oplysningerne der bliver udvekslet foregår på en ensartet måde, og derfor vil være let at implementere, så der kan laves en uafhængighed af serviceudbyderne. SAML tillader overførsel af identitet og ægtheds informationer. Byggestenen i SAML specificerer strukturen og indholdet af ægtheden på brugeren. Hvilke ægtheds informationer der skal med, er defineret i SAML standarden. SAML kan f.eks. sendes over http eller SOAP, dette er defineret i en binding. Sættes bindingen og protokollen sammen, har man en SAML profil. Da profilen sendes over http er der ikke nogen sikkerhed indbygget i SAML, som derfor skal laves ved siden af. Muligheden for at kommunikere oplysninger på tværs af domæner og platforme, er en fordel, da services ikke nødvendigvis vil være hostet det samme sted. Det vigtigste problem SAML løser, er Web Browser Single Sign-On (SSO) problemet, hvor det skal kunne lade sig gøre, kun at logge ind et sted, for derefter at kunne tilgå flere services. SSO-systemet skal bruges på Borger.dk

SSO problemet kan løses på to måder ved hjælp af SAML. Den kan løses ved at brugeren logger ind på portalen, som omdirigerer brugeren til IDPen, hvor ved man kan logge på, eller brugeren kan logge direkte på IDPen, og derfra tilgå portalen. For at se om en bruger er logget på, laver IDPen en cookie, samt en liste over hvem der er logget på. Tilgår brugeren en ny service, vil denne service bruge cookien, og kalde IDPen op, for at tjekke at det er rigtigt brugeren er logget på, hvorved et SSO system vil være til stede.

De fleste SSO produkter der findes i dag, benytter sig af cookies, til at holde på oplysninger om brugeren, så brugeren ikke skal identificere sig hele tiden. Men cookies bliver ikke sendt

mellem domæner, så hvis der er to service på to forskellige domæner, kan cookies ikke bruges. I organisationer, kan man også risikere, at services er delt ud over flere domæner, og man får derfor problemet med at brugeren skal identificere sig flere gange. Man kan derfor benytte sig af Cross-Domain Single-Sign On (CDSSO). Med CDSSO kan man nøjes med at logge ind, hos en af service udbyderne, men det kræver dog at alle udbyderne bruger det samme SSO produkt.

Udover at SAML kan bruges til et SSO system, indeholder det også en profil der kan bruges til single logout, hvor brugeren bliver logget ud af alle services, hvis han logger ud et enkelt sted. Dette sker ved, at IDPen udover at have en liste over hvilke brugere der er logget på, også har en liste over hvilke services brugeren har brugt. Derved kan IDPen når brugeren logger ud et sted, sende oplysninger om at brugeren er logget ud, til samtlige services der er i brug.
En tredje profil i SAML, federation, er sammenkobling af flere brugere. Hvis flere uafhængige services skal arbejde sammen, kan de ved hjælp af federation, lave en tilstand der gør det muligt at samarbejde og sætte brugernes identitet sammen. Da brugere ofte er oprettet på flere services, kan disse brugere sammenkobles, så det ikke er nødvendigt at oprette nye, ens brugere, men i stedet for bruge de gamle, hvor de kobles sammen til en konto.
Disse tre profiler, er blandt de profiler, der er brugt til at danne DK-SAML [10].

4.2.4. Autentifikation og Autorisation gennem en Proxy

En Proxy er et aktivt led i netværket fra slutbruger til ressource. Man kan sige at en portal som henter ressourcer på vegne af en bruger, fungere som en Proxy. Mængden af manipulation af de originale data, afgør hvorvidt portalen kan defineres som en Proxy: Hvis portalen henter et eksternt XML dokument som referere til yderligere ressourcer, er der tale om en Remote-Service. Mens visning af originalt indhold hvor kun referencer er ændret, kan betegnes som en Proxy.

I begge tilfælde er det dog vigtigt at ressourcen tilgås, på en sådan måde at ressourcen ved hvem der bedte om dem. Der er 2 scenarier for hvordan dette kan lade sig gøre:

Portal og ressource har indgået et samarbejde om at dele data over en protokol. Portalen sender et unikt ID med i sit request, som identificerer brugeren på begge systemer
Portalen benytter brugerens brugernavn og password eller certifikat, logger på ressourcen og agere derpå som brugeren.

Scenario 2 er på ingen måder tilladt, idet portalen delvist vil gøre brugeren anonym (IP logning), dernæst at denne type håndtering af certifikater og login, ikke er tilladt

4.3. Eksisterende løsninger

4.3.1. JSR-168

Java Specification Requset (JSR-168) kan ikke direkte sammenlignes med en Gadget/Portlet idet JSR-168 eksekveres som et program på en portal. JSR-168 har derfor ikke rige muligheder for at kommunikere tilbage til sin oprindelse på samme måde som WSRP. Ved at indkapsle JSR-168 portlets i WSRP er det dog muligt at få en tvær-domæne kommunikation men vil lide under alle WSRP's svagheder.

JSR-168 er en Java standard og kan derfor ikke bruges i sammenhæng med .NET platforme. Dette medføre at JSR-168 ikke kan bruges på Borger.dk, hvilket også er konklusionen i den offentlige PoC [PoC].

Det skal dog siges at JSR-168 har nogle gode features, bl.a. inter-portlet kommunikation som ikke ses mange andre steder. JSR-168 bliver generelt rost meget [PoC] for at have god styring af indholdet, og have god performance.

4.3.2. WSRP

Web Services for Remote Portlets (WSRP) er en protokol standard, designet til at kommunikere med portlets, placeret forskellige steder. Der er ikke nogen sikkerhed indbygget i WSRP, men er udelukkende lavet for at hente og præsentere data. Remote Portlets er konfigurerbare web komponenter, hvilket vil sige, at portalen kan bestemme udseende på servicen, men det er serviceudbyderens ansvar at lave et design, der kan bruges. Ved at benytte nogle standard klasse navne for WSRP portletten, gør det er portalen kan style den, og få den til at passe ind i

portalens layout. Da koden til en portlet, ligger hos serviceudbyderen, er det lettere at udvikle og drifte den, i modsætning til hvis det lå direkte på portalen. Derudover er fordelen ved at koden ligger hos udbyderen, at koden har adgang til back end systemer, hvis dette er nødvendigt. Er koden indbygget i portalen, er det nødvendigt at åbne for adgangen til back end systemerne. Ulempen ved at koden ligger hos serviceudbyderen er at load tiden, vil blive forøget i forhold til hvis det hele ligger på portalen.
WSRP profilerne kører over enten SOAP eller http, og der er derfor ingen sikkerhed ved kun at bruge WSRP. En WSRP portlet er derfor god at bruge hvis det ikke er nødvendigt med sikkerhed på ens data. Dette kunne være ikke personfølsomme data, som f.eks. information om en institution eller åbningstider for et told og skat kontor, men hvis det f.eks. er et skattekort der skal hentes, er det nødvendigt at det bliver hentet ved hjælp af en sikker forbindelse, og at man kender identiteten på brugeren, der er logget på. Hvis portalen og WSRP portletten er hostet det samme sted, er det dog ikke et problem, da det så må være givet, at det er den rigtige bruger der er tale om, men i tilfældet med Borger.dk vil services være hostet forskellige steder, hvilket øger problematikken med WSRP. Sikkerhed kan tilføjes ved at lade WSRP portletten køre over en SSL eller VPN forbindelse.

4.3.3. IFrames

Services kan vises på portalen ved hjælp af iFrames. IFrames indeholder ikke fælles login, hvilket derfor skal laves ved siden af. Portalen har ved denne løsning ikke styr på styling af servicen, idet denne sker hos serviceudbyderen. Ved at bruge denne metode, kræver det regler om hvordan servicen skal styles. Det er nemt at integrere iFrames både for portalen og serviceudbyderen, og er en simpel måde at bruge eksisterende webløsninger. [6] [7]
IFrames er ikke lavet til en speciel platform, og kan derfor bruges på alle eksisterende løsninger. Da indholdet der bruges i en iframe ligger hos serviceudbyderen, er det nemmere udviklings og driftsmæssigt, set fra serviceudbyderens synspunkt. Når kode bliver indlejret i en iFrame, fungerer det som en helt ny side, der kører ovenpå en anden side. Den nye side der er i iFramen, har altså ikke noget at gøre med hovedsiden. En iFrame kan derfor ikke ændre på hovedsiden, ligesom den heller ikke kan bruge metoder derfra. At iFramen ikke kan ændre på hovedsiden, gør at man ikke risikere at få skadelig kode ind på siden, der f.eks. prøver at stjæle oplysninger, men da man må gå ud fra at alle services er trustet, burde dette ikke blive et problem.

Der er dog en del ting der taler imod at bruge iFrames. [17]

- Søgemaskiner kan ikke indeksere siden ordentligt, et søgeresultat på nettet kan resultere i at brugeren kommer til at åbne den side som ligger inden i iFramen, og ligger derfor ude af kontekst.
- Man kan ikke lave et bogmærke til en bestemt underside. Vælger man at linke til noget inde i framesettet, vil man kun få vist framesettets side, når linket åbnes igen, og ikke hele portal siden som forventet.
- Skærmlæsere (handicaphjælp) understøtter ikke iFrames ordentligt, hvilket er et krav til portalen.
- iFrames loades asynkront, hvilket bryder helhedsprincippet. Det kan hænde at brugerens tillid svækkes til portalen, hvis iFrames skiller sig ud designmæssigt.
- IFrames understøtter ikke SAML 2.0
- IFrames opfører sig ikke godt, hvis man ændrer størrelsen på browservinduet, det kan derfor risikeres at man ikke kan se hele iFramen på en gang
- Udskriver man en side der indeholder en iFrame, vil man kun få udskrevet det synlige indhold.

4.4. Datatransport

Transporten af data fra portal til serviceudbyder skal foregå over en protokol som alle parter er enige om. Den nemmeste at implementere er http med html, som allerede bruges til de internetbaserede serviceløsninger. Dog findes der nogle alternativer som måske kan bruges.

4.4.1. Sockets

Fundamentet i overførsel af data over nettet er Sockets. En adresse og en port angiver server og service som skal tilgås og over denne forbindelse, kan en besked service fungere. Sockets kan oprettes i 2 varianter: UDP og TCP, hvor TCP giver sikker overførsel af dataene.

Sockets kan i sin simpleste form, bruges til at skabe overførselsprotokoller som kan overføre store mængder data meget hurtigt. Dette kræver dog at man selv står for at analysere de pakker som sendes og modtages.

Det største problem ved at oprette sin egen protokol, er at både portalen og samtlige service-udbydere skal benytte sig af denne protokol. Kravene om sikkerhed vil gøre det meget svært, at overbevise alle parter om at systemet kan fungere.

4.4.2. (http)WebRequest

Indbygget i de fleste programmerings sprog, er brugen af WebRequest – et indbygget objekt, som kan oprette forbindelse til en server over nettet og hente de ressourcer man beder om. Resultatet er en tekststreng som indeholder HTML eller andre tekstdokument typer, hvorpå man kan udføre manipulationer, indtil indholdet kan bruges til præsentation.

HttpWebRequest og WebRequest er i ASP.NET en færdigudviklet løsning, som giver objektmæssigt adgang til http-protokollen. Tegnsæt, headers og andre aspekter er lette at tilgå og gør det derfor nemt at lave programmer som tilgår ressourcer på nettet.
WebRequest er oprettet i .NET som en skalerbar komponent, som benytter sig af tråde. Jo flere cpuer .NET har til rådighed, desto mere arbejde kan WebRequest udføre.

4.4.3. JSON

JavaScript Object Notation [15] - udtalt ”Jason” – er en tekstbaseret protokol, til at overføre objekter over en kommunikations kanal, oftest ses JSON i AJAX, hvor objekter kan overføres fra Klient til Server, men kan også bruges fra Server til Server.

JSON er en meget specifik protokol, som kræver at objekter på hhv. modtager og afsender siden, forstår at bruge disse objekter. En JSON kilde, kan derfor ikke bruges alene, men kræver at klienten som tilgår ressourcen, kan pakke ressourcen ud, så den præsenteres korrekt.

JSON ville kunne bruges til Portal-Service kommunikation såvel som kommunikation imellem services.

4.4.4. SOAP

Simple Object Access Protocol [16] – en tekst protokol som bruges til transport af f.eks. html dokumenter på tværs af forskellige protokoller. F.eks. kan man ved brug af SOAP sende et html dokument over SMTP eller HTTP – hvoraf HTTP er den mest benyttede i dag.

SOAP bruges til indkapsling af beskeder, som skal sendes på tværs af domæner, oftest i form af RPC (Remote Procedure Call) – hvor klienten beder om en handling udført på serveren, hvorefter serveren svare tilbage. Dette kan ske i samme stil som normal webside browsing.

SOAP er en XML-baseret standard, som er sløv og derfor velegnet til systemer, hvor man ikke forventer et hurtigt svar. SOAP kræver på samme måde som JSON, mekanismer hos afsender og modtager, som forstår disse pakker.

4.5. Kildekonvertering i en Remote-Proxy / Gadget

4.5.1. Parsing teknikker

Før HTML'en fra en remote service kan vises på portalen, skal den konverteres til en "remote-proxy" type. Referencer skal laves om i den oprindelige HTML så de benytter sig af enten portalen, eller eksterne ressourcer. Der findes 3-4 umiddelbare løsninger på at gennemføre ændringer i kildekoden fra de forskellige services.

HTML parsing kræver at man konstruere en parser, hvilket ikke er hensigtsmæssigt set i lyset af, at serviceudbydere derefter er nødt til at skrive deres Gadgets, i den understøttede HTML version. Optimistisk set, kan en parser gennemføre ændringer meget hurtigt, idet koden kun gennemlæses 2 gange (grundet parallelle referencer: F.eks. Stylesheets og Scripts). Parsingen foregår ved at dokumentet læses fra start til slut, og kastes node-vis igennem funktioner, som enten lader den passere, eller retter eventuelle links/referencer. Når dokumentet er parsed, løbes det igennem igen for at samle det. Denne gang bliver referencer fra dokumentet (id, name) rettet til i eventuelle scripts/styles

n = antal noder som skal læses/skrives
x = antal opslag i referencer (id, name, styles, classes)
l = antal links som skal "krypteres"

Hastighed[2]: O(2n + log(x) + log(l))

[2] Hastighederne er skønsmæssigt anslået og afhænger af implementeringen.

Regulær Match'n'Replace bygger på Regex (forkortelse af Regular Expressions). Regex benytter sig af specifikke søgeudtryk, som parser kildekoden som ren tekst. Ved hver match udføres en handling på teksten som parses videre til næste match. Regex er en motor som er optimeret mod at udføre hurtige *søg og erstat* handlinger, men kan også bruges til mere formål som f.eks. opdeling af tekst til Array. Ved Regex 'parsing' udføres et antal handlinger på teksten i en sekventiel rækkefølge. Hvor udtryk i højere og højere grad, er afhængige af eller determineret af udtryk som kom før.

Det særligt gode ved Regex, er at Regex er ligeglad med om dokumentet er velformuleret og tillader derfor et dårligt HTML dokument at blive behandlet. Det negative ved Regex er at dårlig HTML slipper igennem, til fare for at ødelægge den HTML hvori teksten skal udskrives.

Ved Array opdeling, benytter man sig ofte af muligheden for at udføre noget kompliceret på den fundne match, hvorefter en ny streng indsættes på match'ens position. Hastighedsmæssigt kan Regex afvikles meget hurtigt, men jo længere teksten bliver, jo mere arbejde bliver der udført.

n = antallet af tegn i dokumentet
r = antal Regex operationer
l = links som skal "krypteres"

Hastighed[3]: O(r * n + log(l))

DOM parsing og XSL-Transformation benytter sig af XML-parsing teknikken som er veldefineret. XHTML dokumentet foldes ud som et træ af noder, og laves på den måde om til et objekt-træ. Man kan derefter benytte sig af XPATH og DOM -kald på træet for at konvertere/redigere bestemte noder.

[3] Hastighederne er skønsmæssigt anslået og afhænger af implementeringen.

DOM/XSLT forudsætter af dokumentet lever op til XML standarden, hvilket indebære at alle tags skal være rigtigt afsluttet, at alle attributter er omringet af pling'er og at alle elementer indeholder gyldige tegn.

Ved at sætte disse "høje" krav til serviceudbyderen, sikres det at portalen har rigelige muligheder for, at transformere layout så det passer med portalen, samt at kunne validere en service inden den behandles. Hagen ved at benytte XHTML og DOM/XSLT er at små og ubetydelige fejl kan være skyld i servicenedbrud. Blot enkelte tegn de forkerte steder kan være nok til at skabe parserfejl.

Hastigheden ved DOM/XSLT er meget høj, idet dokumentet laves om til et Objekt og pakkes ind inden det skal vises igen.

n = antal noder som skal læses/skrives
x = antal operationer igennem træet
l = antal links som skal "krypteres"

Hastighed[4]: O(2n + log(x*n) + log(l))

4.5.2. Links

Links findes i 2 varianter: absolutte og relative. Teknisk set bør alle absolutte links referere til en ekstern side som skal åbnes i et nyt vindue, hvorimod alle relative links referere til ressourcer på samme domæne. At blande de to sammen vil for portalens synspunkt virke forvirrende idet man derved er nødt til at se på om et absolut link peger på samme domæne, i stedet for blot at slippe det igennem uberørt.

Links består af 5 segmenter:

1. Scheme: som oftest er 'http://' og 'https://' men også findes i andre varianter. Scheme definerer protokollen som ressourcen tilgås med, et skift imellem https og http har normalt ikke den store betydning for cookies og sessioner. Derimod skal transaktionsdata krypteres ved https og porten skiftes mellem 443 og 80.

[4] Hastighederne er skønsmæssigt anslået og afhænger af implementeringen.

2. Domain: domænet hvor ressourcerne er lokaliseret, f.eks. www.google.com – Domæne-skift indikere at sessionsdata, cookies samt de resterende punkter, ikke længere er gyldige. Specifikt for Gadgets og deres sikkerhed, skal der være en klar definition om en Gadget må skifte domæne, eller være bundet til ét bestemt.
3. Path: den mappe-hierarkiske placering af dokumentet, f.eks. /debug/. Skal håndteres hver gang der klikket på et link i en Gadget.
4. File: Filen som skal hentes eller eksekveres
5. Querystring: En parametrisk liste af værdi-par (keys = values)

Path, File og Querystring behandles af serviceudbyderen og er selv ansvarlig for at fremstille dem korrekt. På portalens service side vil flere Gadgets fremstå side om side, det er derfor vigtigt at links fra Gadget A sendes til og behandles af Gadget A.

En måde at beskytte portalen mod URL-injektion og gøre links pænere, er at opsamle samtlige URL's fra et dokument. Opslag sker ved hjælp af et unikt id, som er det eneste synlige for brugeren. Eventuelle ekstra Querystring elementer som stammer fra form-get skal dog vedhæftes den originale cashede Querystring, inden et request sendes af sted.

4.5.3. Forms

En række emner skal håndteres når en form indtræder i en Gadget:

Action:

modtageren af formen, angives med et Link (som beskrevet tidligere) som behandles derefter. De regler der gælder for Links, gælder også for Action

Method:

Måden hvormed data skal sendes: GET eller POST – overførsel af tekst sker som regel via GET, hvor POST bruges til meget tekst og alt andet (deriblandt filer)

Accept-charset

Denne attribut respekteres ikke af visse browsere (læs: Internet Explorer), og bør derfor enten fjernes, eller ændres til at indeholde det samme charset som portalen. Charset'et bestemmer hvordan tekst og tegn skal encodes i Querystring.

Enctype

Kan have en af 4 værdier:

[Ingenting] - determineres af Method

text/plain - standard for GET (virker ikke med POST)

application/x-www-form-urlencoded - standard for POST (og GET)

multipart/form-data - bruges til overførsel af filer, kræver plugin på modtager.

Target

Definere om der sendes til en anden frame, iframe, vindue eller samme vindue. Med mindre der skal sendes filer, skal Target være tom, ved fil upload bør target være _blank for at åbne et nyt vindue. Det nye vindue bør åbne en adresse som kan modtage filen, hvorefter vinduet lukkes automatisk.

4.5.4. JavaScript – Applets – Objekts

JavaScript og til dels Applets og Objekts, introducere en række problemer i forbindelse med en portal med flere services på samme side. Dels kan der komme adresse konflikter: IDer på elementer som har samme navn, eller modificering af element-kollektioner som rammer nabo-Gadgets. Dels kan funktioner og objekter komme til at have samme navn som nabo-Gadgets.

Google Gadgets har til dels løst dette problem, ved at overlade en del af arbejdet til udvikleren, at navngive funktioner, variable, objekter, navne og ider med et præfiks som google derefter omdøber inden visning. Hvis udvikleren ikke overholder denne konvention er der således ingen konsekvens fra Google, hvilket tillader ondartede Gadgets at eksistere.

Det største problem ved at lade en parser sørge for præfiks generering, er at JavaScript kan eksekvere forskellige Eval-funktioner: bl.a. setTimeout(), Eval() og setInterval() samt inline scripts (tekst strenge som indsættes i htmlen som derefter eksekvere automatisk) – som gør det meget svært at styre om navnekonventionerne opfyldes!

Applets og Objekts kan på samme måde som JavaScript eksekvere kode, men er mere lukket. Ofte vil de søge adgang til JavaScript objekter eller elementer på siden – hvor man i værste fald kan komme ud for at flere Applets tilgår samme ressourcer.

Objekts findes bl.a. som WMV afspillere, Flash og lign. Ofte vedhæfter man et script som håndterer afspilleren, og her er det igen vigtigt at have en navne konvention, eller et regelsæt som alle kan følge.

4.5.5. AJAX

AJAX giver kun mening hvis udvikleren har adgang til både server og klientside, men på en portal vil man være afskåret fra serveren i et givent omfang. Såfremt der er beskeder som skal sendes frem og tilbage via AJAX, skal der skabes et Framework som Gadgets kan deles om.

Et stort problem med at indfører AJAX er bl.a. svartiden, på en hjemmeside hvor klienten kun skal snakke med serveren, er tiden 1 gang frem og tilbage, over en portal, bliver den således dobbelt så høj. Dette betyder at AJAX applikationer ikke må være lige så afhængige af server-ressourcer som man normalt ser. Den mest succesfulde brug af AJAX vil derfor blive at opdatere Gadgettens indhold, uden at opdatere resten af siden.

Problemet er ikke at implementere et AJAX Framework, men derimod at opretholde tilgængeligheden – regelsættet for tilgængelighed definere at siden skal kunne afvikles uden scripts. En mulighed for at gøre oplevelsen bedre for almindelige mennesker, kunne være at lade brugeren aktivere funktionaliteten via sin profil, eller ved et direkte link på siden. Problemstillingen skal tages op når den endelige implementering finder sted

4.5.6. Validering

Ved indtastning af data, er det ofte ønsket at værdierne valideres. Dette er oftest med henblik på at dataene skal passe ned i databasen, eller at værdien entydigt forstås af systemet – datoer, personnumre, Konti, E-mails osv. Skal alle sammen valideres. Serviceudbyderen SKAL håndtere validering af dataoverførsler, dette skal ske hos udbyderen selv, idet systemet ikke med sikkerhed kan sørge for det. Scripts som normalt validere input er ikke altid aktive, hvilket medføre at ukorrekte data kan slippe igennem. Desuden er det ikke alle brugere som har lige gode intentioner, hvilket igen kræver at der foregår validering så man steder i systemet som muligt.

Endnu en gang er det muligt at stille et Framework op, som serviceudbydere kan benytte sig af, simple valideringer kan ske i JavaScript, såvel på portalen inden de sendes til udbyderen.

4.5.7. Fejlhåndtering

Fejl kan ske mange steder i systemet og det er op til hver enkelt led i kæden at sørge for at fejlhåndtering finder sted.

Portalen kan komme ud for at en serviceudbyder ikke svare, timer ud, eller sender noget ugyldigt data – disse tilfælde skal håndteres så brugeren ved hvad der gik galt.

Program fejl skal ligeledes håndteres de retmæssige steder i systemet, det er ikke hensigtsmæssigt at brugeren ser en fejl 404, 500 eller andre fejlsider, som kun forstås af udviklere.

4.5.8. Styling og integration

Borger.dk indeholder et meget stort stylesheet samt medfølgende grafik, som Gadgets kan/skal benytte sig af. Det er meningen af Gadgets skal optræde som en del af borger.dk, hvilket er nemt når de indbyggede style-klasser benyttes.

Hvis en udbyder alligevel vælger at oprette nogle CSS stil-klasser som de vil benytte, skal disse håndteres af portalen. Dette skal gøres på en sådan måde at de ikke blander sig med resten af stylesheetet.

4.5.9. Tegnsæt

Et aspekt ved webprogrammering, som kan give anledning til mange frustrationer er tegnsæt. Tegnsæt definerer i alt sin enkelthed, hvordan server og klient snakker sammen rent datamæssigt. Grunden til at der findes flere tegnsæt kommer af kravet om, at 1 byte er det maksimale et tegn må fylde. Men da 1 byte kun rummer plads til 256 forskellige tegn, er der ikke nok til alle verdens skrifttegn.

Populært kendes ISO-8859-1 (Latin-1), indeholder alle ASCII tegn, som man kender dem fra C-Programmering. Samt UTF-8 (Unicode Transformation Format) som er den forkortede 8-bit encoding af Unicode, som findes i 16 og 32 bit.

Valget af tegnsæt på en hjemmeside har indflydelse på udformningen af querystrings samt visning af specialtegn som f.eks. æ, ø og å. Det er således ikke ligegyldigt om en Formular på

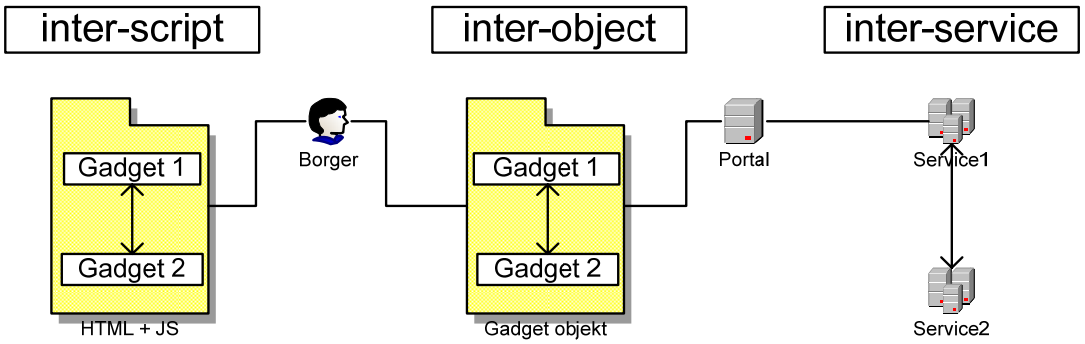
en UTF-8 side, poster sit svar til en Latin-1 modtager. En god håndtering af tegnsæt er derfor vigtig.

Endvidere skal det bemærkes at browsere imellem, har forskellige måder at behandle tegnsæt. Internet Explorer 6.0 læser tegnsættet fra http-hovedet, og respekterer derefter ikke attribut værdier længere nede i dokumentet. Firefox læser som IE6, tegnsættet fra http-hovedet, men til forskel bliver attributter respekteret.

Således vil en formular der er defineret:
<form method=”post” accept-charset=”utf-8”>
På en side, hvor http-hovedet angiver ISO-8859-1 – resultere i forskellige Querystrings, alt efter hvilken browser som benyttes.

4.6. Kommunikation mellem services / Gadgets

Et stort og vigtigt emne som en service orienteret portal bør understøtte, er kommunikation mellem services. Dette med henblik på at udføre opgaver, som i den ”virkelige verden” kræver en lang og bureaukratisk proces. Et eksempel er ved generhvervelse af vognmandstilladelse (beskrevet i PoC’en) – proceduren kræver samarbejde imellem flere myndigheder og en del papirarbejde: Bl.a. skal man have fat i CPR/CVR, politiet og skat.dk



Figur 10 - kommunikation mellem services

4.6.1. Kommunikation på service niveau

En serviceudbyder påtager sig ansvaret for at oprette en fælles-service. Dette kræver at udbyderen opretter en aftale med de myndigheder som er involveret, samt påtager sig ansvaret for at lave en præsentation som vises på borger.dk.

Løsningen forudsætter at alle parter indgår et samarbejde om vidensdeling, baseret på en brugers oplysninger (CPR/CVR). En løsning som ikke er nem at indfører, fordi myndighederne i dag har meget svært ved at blive enige om hvilke oplysninger der må sendes frem og tilbage. Man kan sige at løsningen ville gøre op med bureaukratiet, til fordel for borgerne.

Problemet er at samfundet og dets myndigheder er bygget op sådan, at borgeren er samlingspunktet for informationer og at borgeren er ansvarlig for at sende informationer fra A til B. Nogle tiltag til at gøre op med denne arkitektur findes dog flere og flere steder og ofte bliver informationer sendt på vegne af borgeren.

Ofte involveres myndigheder og instanser i samfundet, hvor det for brugeren er en fordel at viden ikke bliver delt. F.eks. er det ikke altid en fordel at pensions eller forsikrings firmaer har adgang til journaler, idet borgeren måske ville miste nogle rettigheder. Ikke nødvendigvis fordi borgeren ”fusker”, men fordi informationer altid bliver tolket forskelligt.

Vidensdeling på tværs af services kan ske over en XML eller JSON protokol, hvor en serviceudbyder har en "step-maskine" over processer som skal gennemføres for at en procedure gennemføres:

Eksempel på en procedure

Procedure: Generhvervelse af vognmandstilladelse (GAVMT)

1. Autentifikation / Autorisation
    a. CPR/CVR: Hent Oplysninger (Brugerens Certifikat)
2. Indsaml oplysninger
    a. Politiet: Hent Oplysninger (GAVMT, CPR, CVR)
    b. Skat: Hent Oplysninger (GAVMT, CPR, CVR)
3. Præsenter procedure for brugeren
    a. indsaml oplysninger fra bruger
    b. afvent godkendelse
4. Afslut procedure
    a. Send/Gem oplysninger de retmæssige modtagere

4.6.2. Kommunikation på portal niveau

Der er ikke stor forskel på hvordan mekanismen fungere på portal niveau i forhold til hos en serviceudbyder. Informationerne kan på samme måde, hentes i XML eller JSON format og blandes sammen til et layout som borgeren derefter skal acceptere. Dét som gør det sværere at lave procedure på portal niveau, vil være at lave et abstrakt Framework som kan udføre mere end bare 1 procedure. På denne måde behøver man kun lave ét system, som alle serviceudbydere kan benytte sig af. Alternativet ville være at lave en algoritme for hver procedure som skal kunne udføres på tværs af Gadgets, som det er tilfældet på service niveau.

En måde at lave et abstrakt Framework kunne være ”makroer”, af typen XML/XSLT eller JSON/Script. Makroen ligger på portalen, hvor den kan køres af brugere som har adgang til alle de ressourcer (service sites) som er listet i proceduren. Proceduren beskrives af en serviceudbyder som normalt er modtageren af den færdige formular og skal derefter implementeres som makro af borger.dk teamet. En mulighed er dog også at serviceudbyderen fremstiller makroen selv, og lader brugeren aktivere den igennem sin egen Gadget.

4.6.3. Kommunikation på Gadget niveau

En sidste mulighed er at lade en procedure ske hos brugeren selv. Dette eliminerer de aspekter der er ved at lade portalen handle på vegne af brugeren, men introducere nogle sikkerheds risici, samt tilgængelighedsproblemer. Data transporten vil foregå i JavaScript / Applet / Flash regi, hvilket gør dem sårbare overfor data-sniffere som brugeren kan have kommet til at installere.

Procedurer ville kunne foregå på nøjagtig samme måde som brugeren husker fra virkeligheden: *overfør formular B fra myndighed 1 til 2, underskriv formular A og send til myndighed 1.* - Bureaukratisk når det er bedst.
Brugeren kan endda få nogle små brikker at flytte rundt med, som kunne symbolisere formularer og cirkulærer.

Løsningen vil dog kun virke i en utopisk verden: uden handikappede, uden spyware og uden deaktivering af Scripts. Selv hvis data var krypteret, ville et krav om scripts sætte løsningen ud over kravspecifikationen.

4.7. Sikkerhedsaspekter

4.7.1. Sikker transport

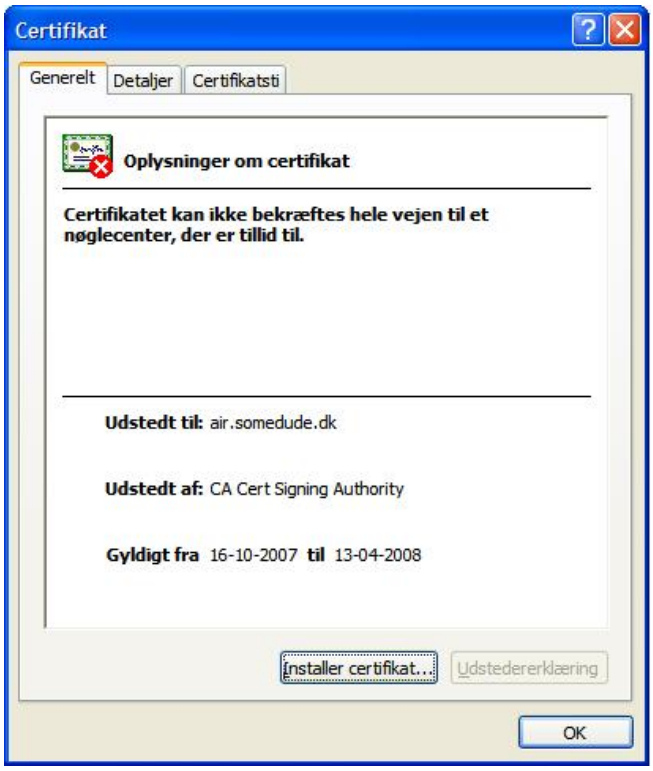
Da mange personlige data i dag bliver sendt frem og tilbage over nettet, er det vigtigt at sikre at elektronisk kommunikation foregår på betryggende vis.
Derudover er det vigtigt at sikre sig, at den webpart man kalder op til, er den, den udgiver sig som.
Ved at sætte en side op til at køre som https og bruge 128 bits kryptering, gør man at udefrakommende ikke kan opsnappe ens data, samt læse hvad der bliver sendt frem og tilbage.
Næsten alt data der sendes over nettet, sendes ved hjælp af kommunikationsprotokollen http. Http står for hypertext transfer protocol, og er en request/response protokol der står for kommunikation mellem klient og server. Typisk laver klienten et request ved at lave en TCP (Transmission Control Protocol) forbindelse til serveren, hvorefter serveren vil svare igen på requesten.
Når man forbinder til en https side i stedet for http, indikerer det at det er en sikker http forbindelse der bruges. Der er her lagt et ekstra krypterings- / dekrypterings-lag ind imellem http og tcp protokollen.
Forskellen på http og https er altså at http protokollen sender al data åbent, så enhver der opsnapper det, vil kunne læse det hele, mens https ikke kan aflæses. Https er ikke en separat protokol, men http protokollen der bruges over en SSL (Secure Sockets Layer) eller TLS (Transport Layer Security) forbindelse.
Alle netbanker bruger i dag https, og kører med en 128 bits kryptering. Steder der kræver cpr-numre benytter sig også af https. Det er kun modtager og afsender der kan læse data der sendes via https, så ved https undgår man at en lytter med på linjen, samt man-in-the-middle angreb. Når data er ankommet til bestemmelses målet, er der ikke længere en forskel på http og https, men data er kun så sikker som den computer, data bliver sendt til.
For at benytte sig af https kræver det at man har installeret et certifikat, til at bekræfte ens identitet. Det er ikke muligt at have flere server certifikater på en maskine, så det er ikke muligt at bruge flere domænenavne, på en ip adresse.

4.7.2. Validering af service udbyder

For at sikre sig at service udbyder er den, vedkommende udgiver sig for, kan man installere et servercertifikat. Et Server certifikat bekræfter en sammenhæng mellem en virksomhed eller organisation og en webserver. Videre bruges certifikatet til at etablere kryptering af kommunikationen imellem brugeren og hjemmesiden.
Certifikater udstedes af virksomheder kaldet nøglecentre. Ved hjælp af et webstedcertifikat, kan alle besøgende se, at et bestemt websted er ægte. Det sikrer, at der ikke er et websted, der overtager et andet websteds identitet. Når man kobler op til et websted, der har et certifikat, kan man få det vist, så det kan bekræftes, at data kommer fra en kendt pålidelig kilde, og at det ikke er blevet ændret undervejs.
For at få et troværdigt server certifikat, kan man ikke selv udstede sit certifikat. Da det koster penge at få udstedt et certifikat hos TDC, har vi valgt at bruge cacert.org til at lave et servercertifikat til os. Det er en gratis online tjeneste, der sælger sig selv på at der i flere år er blevet solgt en service, der ikke bør og skal koste noget.

Ca Cert Signing Authority sikrer sig, at de udsteder certifikater til de rigtige personer, ved hjælp af mail adresser. De vil kun udstede et servercertifikat, til personer, der har adgang til root eller admin mail adresser, til et domæne. Det betyder at for at få udstedt et certifikat, til air.somedude.dk, kræver det at man kan modtage mails på enten root@somedude.dk eller admin@somedude.dk. Kan man ikke modtage mails på en af de to adresser, kan man ikke få et certifikat dertil, da Ca Cert så ikke kan bekræfte ens identitet.
Begrænsningerne på servercertifikater fra Cacert er at de udløber efter 6 måneder, så de skal fornyes forholdsvis ofte, hvorimod en digital signatur fra tdc holder væsentlig længere. Derudover er det kun domæne navnet der står i certifikatet.

Et webstedcertifikat kan se således ud hvis det ikke er trustet:



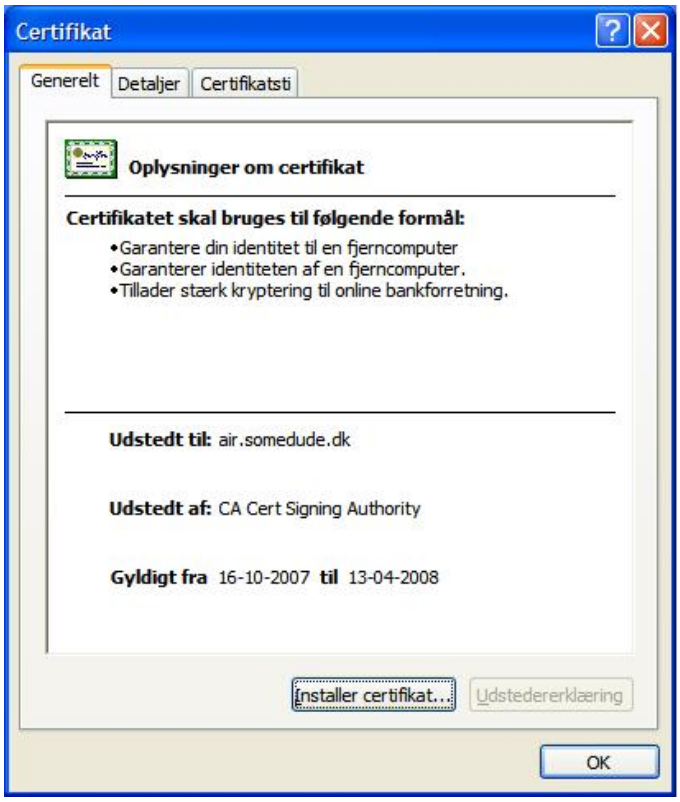
Figur 11 - Server certifikat der ikke kan bekræftes

Dette er et certifikat, der er udstedt af CA Cert Signing Authority, til air.somedude.dk. Gyldigheden af certifikatet løber fra d. 16-10-2007 til d. 13-04-2008.

Grunden til der står at certifikatet ikke kan bekræftes er, at Ca Cert Signing Authority endnu ikke er sat på listen som trustet authority til at udstede certifikater.

For at tilføje Ca Cert Signing Authority som en trustet udsteder, skal man tilføje deres root certifikat. Ved at tilføje deres root certifikat, og vælge at man truster dem, betyder det, at man tror på, at alle de certifikater der udstedes af Ca Cert er gyldige, og er tildelt de rigtige sites. Det vil sige det kun er dem der ejer domænet somedude.dk, der kan få udstedt et certifikat dertil.

Hvis man tilføjer Ca Cert som en trustet udbyder, stoler man på de certifikater de udsteder. Et trustet certifikat vil se ud som følgende:



Figur 12 - server certifikat der kan bekræftes

Ligesom et websted kan have et certifikat, kan privat personer også have et certifikat. Dette kaldes for et personligt certifikat. For at kunne logge på borger.dk skal der bruges en digital signatur. Digital signatur er et personcertifikat udstedt af TDC, som er den eneste der udbyder dette i Danmark. I fremtiden kan man godt risikere at det er en anden udbyder der står for det, men i øjeblikket har TDC og PBS vundet udbuddet. Vi har dog ikke tænkt os at bruge digital signatur, da vi ikke skal lave et login modul.

I vores tilfælde hvor vi skal sikre at det er borger.dk der kobler op til en serviceudbyder, skal vi bruge et klientcertifikat. Det skal bekræfte at vi er den vi giver os ud for. Ved at sætte serviceudbyderne op til at kræve certifikater af os, når vi vil hente en service, kan vi sikre os at der ikke er sider, der ikke burde have adgang til en service, der bruger den.

Overordnet kaldes certifikater for sikkerhedscertifikater. De fungerer ved, at et id knyttes elektronisk til en bestemt meddelelse eller et bestemt websted. Dette sikrer, at en meddelelse kun kan åbnes af den person, den er sendt til. Certifikatet indeholder en godkendelse af identiteten, der bekræftes af et nøglecenter, i forbindelse med den pågældende meddelelse eller det pågældende websted.

Det skal forstås sådan at et certifikat er et digitalt id-kort, og den private nøgle bruges til at bevise at man er indehaveren af certifikatet. En signatur er en matematisk beregning, der bruger den private nøgle, til at underskrive dokumentet der sendes.

Når vi opretter forbindelse til en service, der kører med et server certifikat, sender webstedet sit certifikat retur til os. Dette certifikat, skal vi derefter undersøge, for at se om det er gyldigt. Når man validerer et certifikat, kan det ende med to udslag. Enten er certifikatet gyldigt, og bliver accepteret, eller også er det ikke gyldigt, og skal så afvises.
Valideringen af server certifikatet foregår i vores applikation. Efter modtagelsen af certifikatet, skal certifikatet undersøges for følgende:
Udløbsdato
Policy errors
Er det en troværdig authority der har udstedt certifikatet
Er certifikatet udstedt til serviceudbyderen
Er certifikatet gyldigt/spærret

Policy errors tjekkes ved at se hvilken status SslPolicyErrors har. SslPolicyErrors er en enum, der kan få værdierne:
None
RemoteCertificateChainErrors (ChainStatus har returneret et tomt array)
RemoteCertificateNameMismatch (Certifikat navnet har uoverensstemmelser)
RemoteCertificateNotAvailable (Certifikatet er ikke tilgængeligt)

Servercertifikatet bliver kun godkendt, hvis den har en værdi der er lig med none errors.

Er et af punkterne under valideringen ikke opfyldt, kan vi ikke bekræfte serviceudbyderen, og vi skal derfor have afbrudt forbindelsen. Er alle punkterne derimod opfyldt, godkender vi certifikatet, og skal derefter sende klientcertifikatet af sted til serviceudbyderen.
Det er nu serviceudbyderens opgave at se om dette certifikat er gyldigt, og kun hvis det er gyldigt, skal vi have lov til at tilgå siden.

For at kalde en service og bruge et klientcertifikat, er man nødt til at opbevare certifikatet et sted, før man kan indlæse det. Der er to måder det kan opbevares på. Man kan vælge imellem

at have certifikaterne liggende i certificate store, eller at have dem liggende i en mappe på disken.
Der er ikke den store forskel på det sikkerhedsmæssigt, da man må gå ud fra, at der ikke er nogle udefra kommende, der har adgang til data på disken.

Vælger vi at have certifikaterne på disken, skal vi bruge stien til hvor certifikaterne ligger. Første gang vi kalder en service vil vi hente certifikatet dertil og indlæse dette. Når certifikatet er loadet vil vi gemme det i en X509CertificateCollection, som er en samling der kan opbevare certifikater. Ved at lave vores kollektion som static, slipper vi for at skulle indlæse det samme certifikat, hver eneste gang vi kalder en bestemt service.
Når vi loader en Gadget, skal den automatisk tage certifikatet med, men da den først skal vide hvor certifikatet ligger, skal vi starte med at kalde Gadgetten med stien til certifikatet. Ved at gemme stien til certifikatet, i web.config kan vi sikre os, at det kun er vores program der kender placeringen af certifikaterne. Den eneste parameter vi behøver at sende med, for at hente et certifikat, er navnet på certifikat filen.

Alternativ til at have certifikaterne liggende som en fil, kan vi vælge at have certifikaterne gemt i certificate store. Ved at have dem i certificate store, kan vi vælge mellem at loade et enkelt certifikat der kan bruges til den enkelte service, når servicen skal loades, eller vi kan indlæse alle certifikaterne fra certificate store og gemme dem i en X509CertificateCollection.

Hvad enten vi henter certifikatet fra disken eller fra certificate store, skal vi når certifikatet er blevet indlæst, gennemgå vores samling af certifikater, før vi tilføjer det nye certifikat, for at se om det allerede skulle være i vores samling. Er certifikatet ikke i samlingen, skal vi have tilføjet det, så vi ikke skal indlæse det næste gang. Skulle certifikatet være i samlingen i forvejen, fortsætter vi med indlæsningen af servicen, uden at indlæse det igen.

4.7.3. Sikker behandling af data

Vi står ikke for at lave services for serviceudbyderen, og det er derfor op til dem at sikre deres data. Selvom de stiller en service til rådighed skal de ikke lempe sikkerheden, den skal altså forblive den samme eller bedre. Portalen må ikke gemme oplysninger om brugerne, og har

derfor ingen gemte data at behandle. Al data der kommunikeres igennem portalen skal ske på en sikker forbindelse, så der således ikke vil være en sikkerhedsbrist på data.

4.8. Resume

I analysen har vi fortalt hvad der kan bruges til at genkende brugere i dag, samt hvordan man kan lave et single sign-on system.
Derudover har vi været inde på hvilke eksisterende løsninger der findes i dag, til at vise Gadgets på en portal.
Til sidst i vores analyse har vi beskrevet hvad systemet skal kunne. Vi har lavet en detaljeret beskrivelse af de forskellige handlinger, og lavet system sekvens diagrammer for hver usecase.
Ud fra usecases og sekvens diagrammer, vil vi lave et design, der gør at vi kan implementere disse funktioner.

Kapitel 5

5. Design

5.1. Valg af løsningsform

5.1.1. Dataoverførsel

Til overførsel af informationer, vælges brugen af WebRequest, med den begrundelse, at serviceudbyderne kan fortsætte med at bruge de teknologier de allerede benytter til deres webløsninger. At udbygge deres service til at have en Gadget-løsning er derfor en minimal opgave. Serviceudbydere behøver ikke at installere yderligere produkter på deres servere for at kunne tilbyde en Gadget-løsning.

5.1.2. Konvertering

For at en serviceudbyders løsning kan vises, skal siden konverteres. Dette gøres i første omgang med Regular Expressions, som er mindre striks overfor kilden. Regex er hurtig at udvikle i, hvilket gør at, man har en løsning ret hurtigt.

Ved at have en tidlig løsning, er det også muligt at bruge ressourcer, som ikke skal konstrueres i hånden, for derved at tilgå en større mængde test-sider. Google.com er et godt eksempel, kildekoden er meget specielt skrevet, med ingen klare retningslinjer for om f.eks. attributter skal have situationstegn eller ej, samt linjeskift midt i tags.

Resultatet ved at starte med Regex, er opbygning af viden omkring hvad man kan forvente af html sider. Samt en samling af Regex som kan bruges i andre konverterings motorer. Algoritmerne som bruges, er derfor nemme at portere til en DOM løsning.

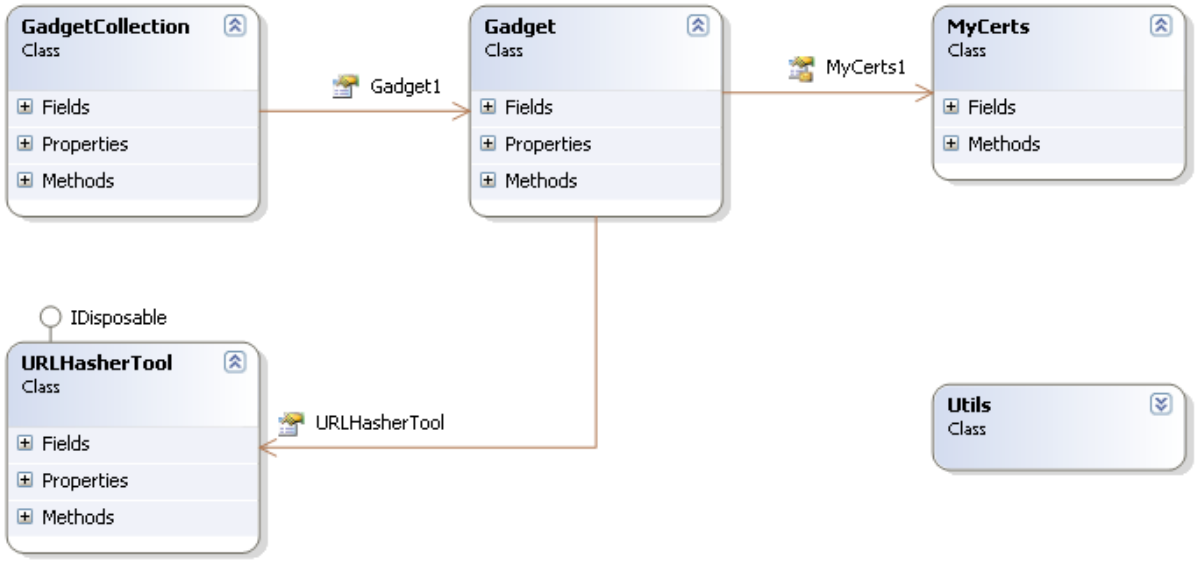
5.1.3. Sikkerhed og SSO

Da det ikke er muligt at afprøve eller benytte sig af en SSO løsning. Har vi valgt at bruge SSL imellem alle parter i systemet. Vores bud på en SSO løsning er at videresende brugerens certifikat til serviceudbyderen, som derved autentificere og autorisere brugeren.

Sikkerheden imellem Portal og Serviceudbyder, sker ved SSL - en 2-vejs autentifikation med certifikater.

5.2. Klassediagram

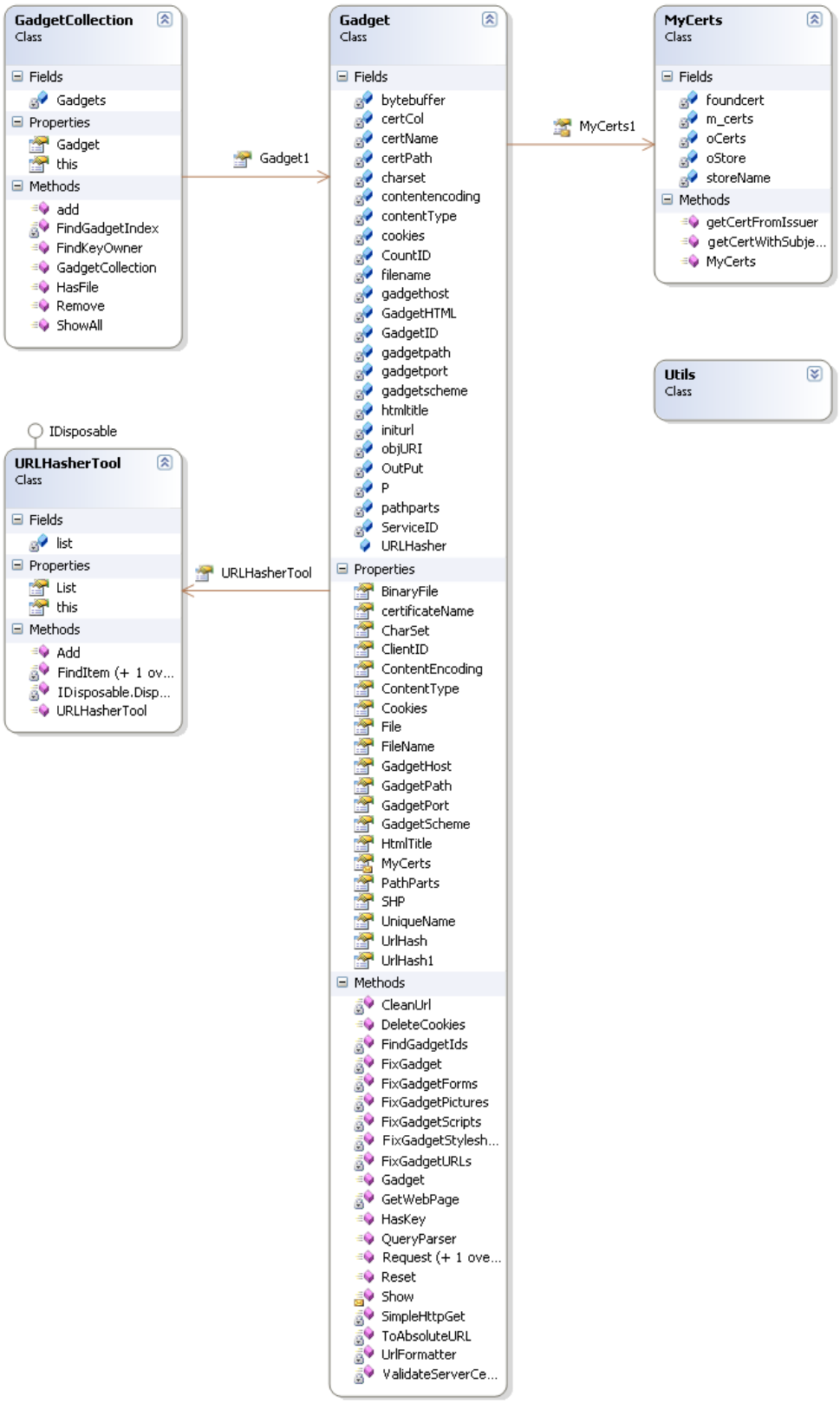
Klassediagrammet er lavet ud fra analysen med vores usecases. Vi har løbene rettet vores klassediagram til, efterhånden som der er blevet ændret på klasserne. Til at lave klasse diagrammet, har vi brugt Visual Studios Class Designer.



Figur 13 - simpelt klassediagram.

På Figur 13 ser vi et enkelt klassediagram hvor vi kan se associationerne mellem de forskellige klasser. Det kan bruges hvis man vil have et overblik over hvilke klasser der findes, og hvem der bruger klasserne.

Det er vores default.aspx side der kalder GadgetCollection, ligesom det er default.aspx der kalder Utils klassen. Som det ses er det vores GadgetCollection der benytter sig af Gadget klassen, og Gadget klassen der kalder MyCerts og URLHasherTool.



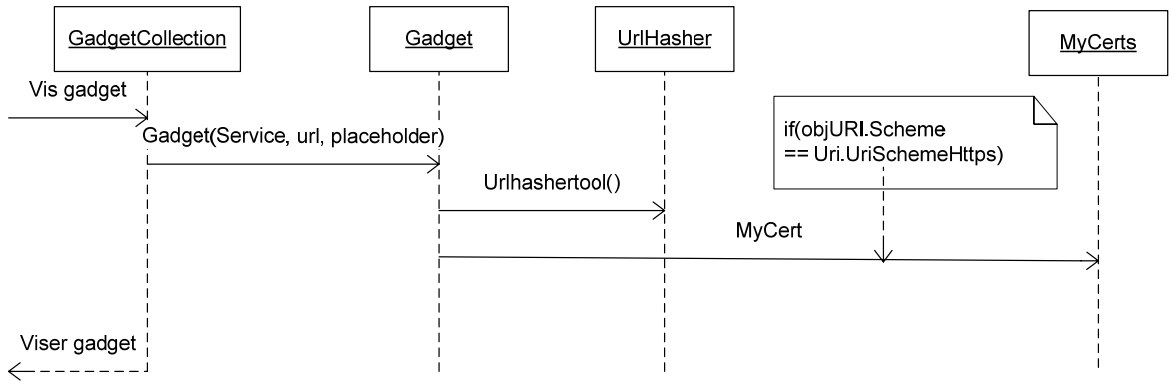
Figur 14 - detaljeret klassediagram.

Figur 14 viser et mere detaljeret klassediagram, hvor man kan se hvilke variabler der findes i de enkelte klasser, hvilke properties der er, samt hvilke metoder der findes.

5.3. Sekvens diagrammer for usecases

I analysen blev der lavet system sekvens diagrammer for vores usecases, der viste brugerens interaktion med systemet, men ikke hvordan selve systemets interaktion fungerede. Vi har derfor lavet nogle sekvens diagrammer i design fasen, der viser sammenhængen mellem vores klasser.

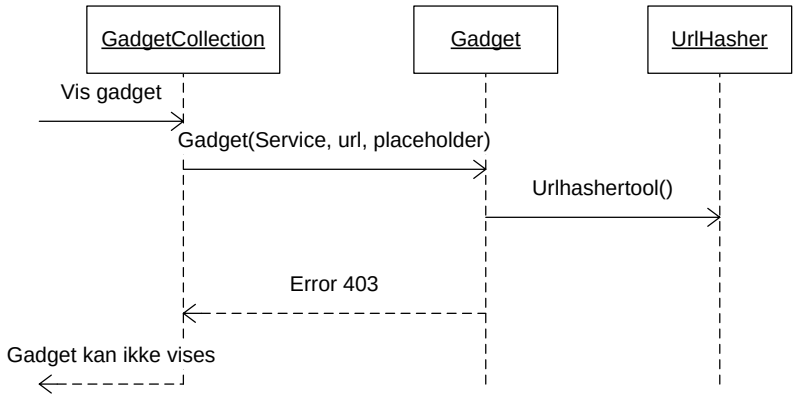
5.3.1. Sekvens diagram for usecase 1



Figur 15 - Sekvens diagram for usecase 1.

På Figur 15 ser vi interaktionen mellem klasserne, når brugeren får vist en Gadget. Vores default.aspx side opretter en GadgetCollection, der består af en Gadget. Gadget klassen laver derefter en UrlHasher, og kalder derefter MyCerts, hvis der er tale om en sikkerforbindelse. Til sidst vises Gadgetten på portalen.

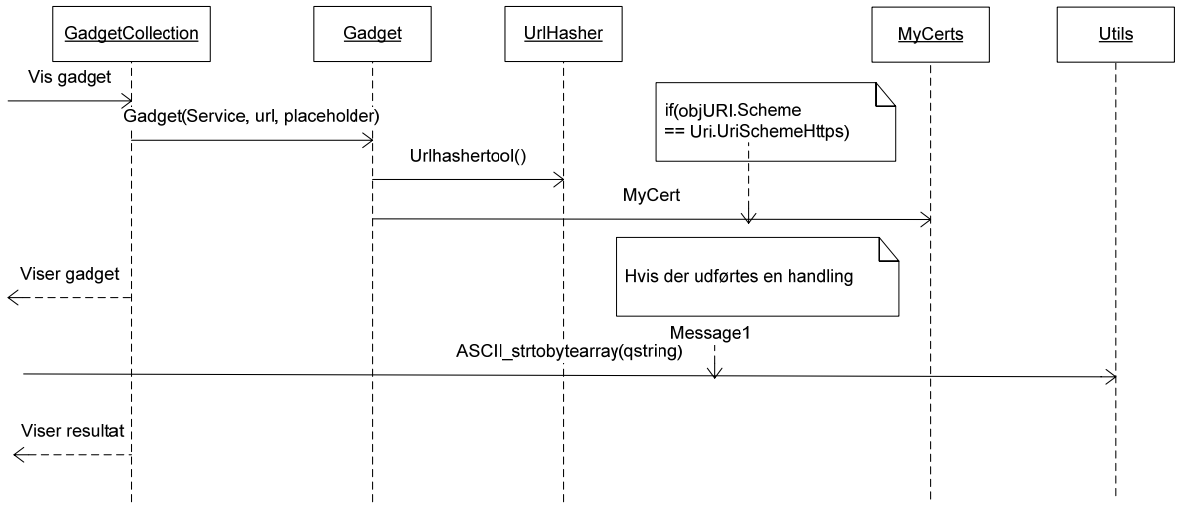
5.3.2. Sekvens diagram for usecase 2



Figur 16 - Sekvens diagram for usecase 2.

Sekvens diagrammet på Figur 16 viser interaktionen hvis der prøves at oprette forbindelse til en service, via en sikker forbindelse, hvor der ikke er adgang. Default.aspx siden laver en GadgetCollection, der indeholder en Gadget der skal køre på en sikker forbindelse. Gadgetten laver en Urlhashertool, hvorefter den får at vide at der er sket en fejl 403, i dette tilfælde fordi der ikke er brugt et certifikat. Brugeren får nu at vide at Gadgetten ikke kan vises.

5.3.3. Sekvens diagram for usecase 3



Figur 17 - Sekvens diagram for usecase 3.

Figur 17 viser interaktionen når der forsøges at udføre en handling i en Gadget. Ligesom med de andre usecases laves der en GadgetCollection, der indeholder en Gadget. Der oprettes en Urlhashertool, og derefter et MyCert objekt til at hente et certifikat med. Gadgetten vises derefter. Fra portalen udfører brugeren nu en handling, hvilket lavet et objekt af ASCII_strtobytearray, hvor vi derefter får resultatet fra handlingen.
MyCerts klassen bliver kun brugt, hvis der skal hentes et certifikat fra certificate store.

Der er ikke lavet sekvens diagrammer over usecase 3.1, 3.2 og 4, da usecase 3.1 og 3.2 fungerer på samme måde som usecase 3. Usecase 4 har vi ikke tænkt os at implementere, men er mere ment som en usecase, til en udvidelse i fremtiden.

5.4. Resume

I design fasen har vi fået lavet sekvensdiagrammer ud fra vores analyse fase, hvor vi har brugt de enkelte usecases. Vi har derudover lavet et klassediagram over hvordan vores program ser ud.

Kapitel 6

6. Implementering

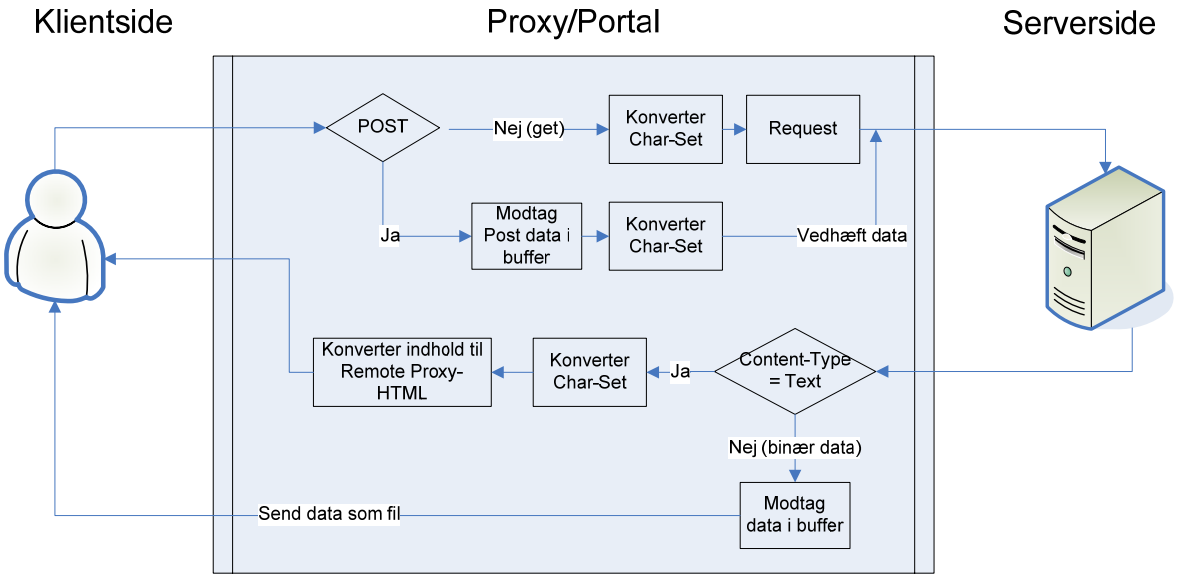
6.1. *Implementering af design*

I implementeringsfasen vil vi fortælle hvordan vi har lavet vores produkt.

6.2. *Proxy*

Idet der laves en aktiv Proxy med hhv. en klientside (den ende som brugerne vil opleve) og en serverside (den ende som snakker sammen med services), vil dataflows beskrives herefter.

Dataflowet igennem en Proxy, ser i sin kompakte form ud som på følgende tegning:



Figur 18 - Dataflow igennem proxy.

Det ses at flowet variere imellem at skulle håndtere en datamængde og ren tekst. Desuden ses det, at der foregår konvertering af tegnsæt under alle overførsler. Mere om dette vil beskrives i et senere afsnit. Konverteringsmotoren sørger for at alle relative URIer på den modtagne side, konverteres så de peger på portalen selv, absolutte links betragtes derimod som eksterne.

6.2.1. Tegnsæt håndtering

Idet en serviceudbyder kan vælge at udfærdige alle sider i UTF-8 mens andre vælger ISO-8859-1 (Latin-1), skal begge med flere understøttes. Dog vil klientsiden på portalen altid have samme tegnsæt, mens tegnsæt bliver behandlet op imod hver enkelt side fra serviceudbydere.

En Gadgets tegnsæt bliver initialiseret når den første side hentes. CharSet propertien på Gadgetten bliver sat hver gang der skiftes side og opdager derfor ændringer når de sker. Resten af siden/datastrømmen læses med samme tegnsæt:

```
Gadget.CharSet = (objWebResponse.CharacterSet.Length > 0) ? Encoding.GetEncoding(objWebResponse.CharacterSet) : Encoding.Default;

Stream objStream = objWebResponse.GetResponseStream();
StreamReader objStreamReader = new StreamReader(objStream, Gadget.CharSet);
```

Når der trykkes på et link, eller der postes en form på klientsiden, fanges handlingen af Page_Load. Den eventuelle Querystring omformateres med Gadgettens tegnsæt og bruges derefter til at hente en ny ressource fra serviceudbyderen. Her ses en GET mod portalen

```
if (Method.Equals("GET"))
{
   string QString = "";
   foreach (string Q in Request.QueryString)
   {
      string val = Request.QueryString[Q];
      if (!Q.Equals("LinkKey"))
         QString += Q + "=" + HttpUtility.UrlEncode(val, GadgetList[gid].CharSet) + "&";
   }

   GadgetList[gid].Request(path, QString);
}
```

6.2.2. Get og Post

I den nuværende version af Proxien understøttes 2 http metoder, Get og til dels Post. Get bruges når der klikkes på et link eller en form-get sendes, hvor Post bruges til at sende store beskeder. Grunden til at Post kun delvist er understøttet, forstås ved at multipart/form-data og text/plain encoding, kræver plugin til modtagelse af filer/data. ASP.NET platformen har dog sit eget system til upload af filer, men er i nuværende version af proxien, ikke supporteret. Get understøttes fuldt ud i forms, fordi dataene altid bliver lagt i querystring.

Fælles for begge metoder, gælder det at kommandoen modtages af proxien, hvorefter indholdet konverteres til modtagers tegnset, inden kommandoen sendes videre. Figuren: Dataflow igennem proxy viser hvordan Post og Get sendes igennem.

Dataene fra Post (normalt fra en Form) gemmes i en buffer, og sendes til serviceudbyderen når der laves et Request:

```
if (Gadget.BinaryFile != null){
   objWebRequest.Method = "POST";
   objWebRequest.ContentLength = Gadget.BinaryFile.Length;
   objWebRequest.ContentType = Gadget.ContentType;

   Stream requestStream = objWebRequest.GetRequestStream();
   requestStream.Write(Gadget.BinaryFile, 0, Gadget.BinaryFile.Length);
   requestStream.Close();

   //clear buffers
   Gadget.BinaryFile = null;
   Gadget.ContentType = "";

}else{
   objWebRequest.Method = "GET";
}
```

6.2.3. Regular Expressions

Der bliver oprettet 26 statiske Regex objekter, som bruges igennem konverterings-funktionerne. Fælles for dem alle er, at kompileringsflaget er sat, hvilket effektiviserer dem betydeligt. For at spare på antallet af operationer, er hver enkelt Regex konstrueret så kompleks som muligt, for derved at ramme bredest. Resten af Regex operationerne bliver sammensat af dynamiske data undervejs og kan ikke optimeres yderligere.

Grundet kilden ikke kan forventes at være formateret i henhold til XHTML standarden, bliver Regexene relativt avancerede. Ved <a> og <form> tags hvor URL'en skal hives ud, bliver man derfor nødt til at kunne omgås tilfælde, hvor URL ikke omringes af hverken gåseøjne eller plinger. Endvidere skal der differentieres imellem <form> tags med og uden action attribut.

```
private static Regex URLPattern = new Regex("(<a.+?href=)([\"'])?(((?<!http).)+?)(?(2)(\\2)|([>  ]))",
RegexOptions.IgnoreCase | RegexOptions.Compiled);

private static Regex FORMPattern = new Regex("(<form[\\s\\S.]+?action=)([\"'])?(((?<!http).)*?)(?(2)(\\2)|([>
]))([\\s\\S.]+?)(<)", RegexOptions.IgnoreCase | RegexOptions.Compiled);

private static Regex FORMPatternNoAction = new Regex(@"(<form (?![.\s\S]*?action)[\s\S.]*?>|<form>)",
RegexOptions.IgnoreCase | RegexOptions.Compiled);
```

Et eksempel på hvordan Regex bliver brugt til at gennemløbe matches og udføre noget kompleks på dem, gøres på denne måde: I et foreach loop findes et match, som via Regexen indeholder en række del-elementer. Et af elementerne i dette eksempel, er en URL som konverteres til en LinkKey, som indsættes i et hidden field. Mere specifikt sker dette:

<form action="submitpage.asp">

➔

<form action=”default.aspx”><input type=”hidden” name=”LinkKey” value=”[key]”>

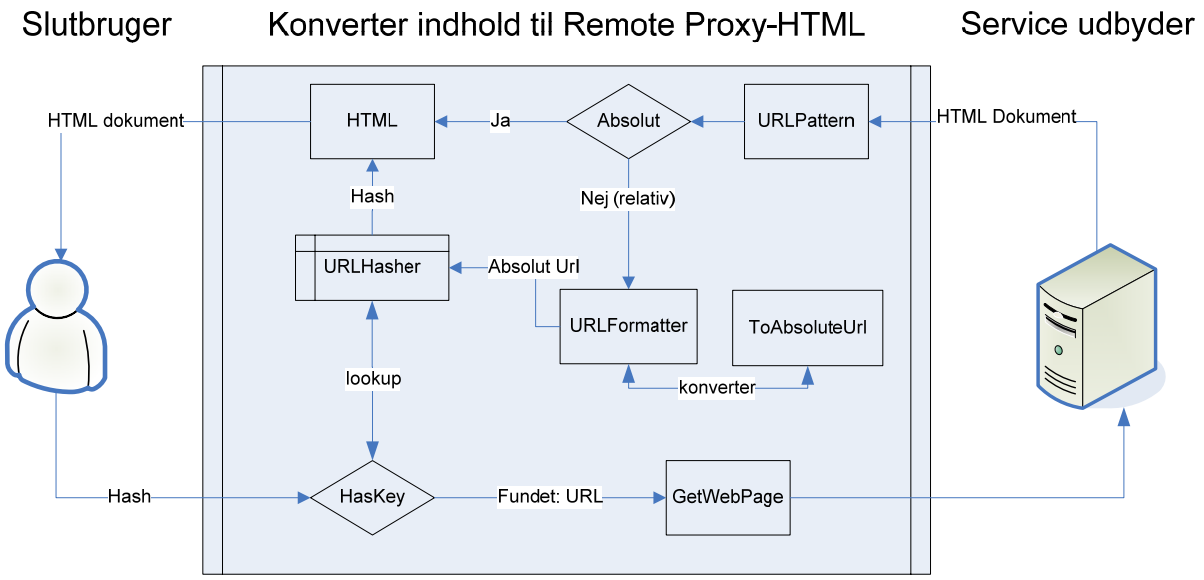
```
string hidden = "";
//WITH ACTION
foreach (Match m in FORMPattern.Matches(HTML))
{
	UrlHash UH = Gadget.URLHasher.Add(new
		UrlHash(UrlFormatter(m.Groups[3].ToString(), Gadget)));
	hidden = "<input type='hidden' name='LinkKey' value='" +
		UH.ID + "'>";
	HTML = HTML.Replace(m.ToString(), m.Groups[1].ToString() +
		m.Groups[2].ToString() + "default.aspx" +
		m.Groups[5].ToString() + m.Groups[6].ToString() +
		m.Groups[7].ToString() + hidden + m.Groups[8].ToString());
}
```

6.2.4. URL håndtering

Mekanismen fungerer ved at det hentede dokument fra en ekstern side, løbes igennem efter URI forekomster. Der søges på de mest almindelige forekomster, hvor det dog kun Hyperlink og Form action som får sløret deres destinationer:

Hyperlink	- <a href=”URL”>
Form action	- <form action=”URL”>
CSS link	- url(’URL’)
Billeder	- <img src=”URL”>
Baggrunde	- <? background=”URL”>

Absolutte links lades passere, mens relative bliver konverteret til absolutte links inden de gemmes i URL-hashen. Dette sker med henblik på at lave et nemmere tjek, inden og mens GetWebPage skal kaldes.



Figur 19 - Håndtering af html dokument.

URLpattern repræsenterer på Figur 19, alle de Regex patterns som fanger en URL i et HTML dokument. De fundne links sendes igennem en funktion som undersøger, om linket er absolut eller relativt. I tilfældes at linket er absolut, passeres det igennem uændret. Relative links bliver sendt til URLFormatter funktionen, som konvertere linket til et absolut link. Funktionen gemmer linket i en hash tabel, hvor kun ikke eksisterende URL's bliver lagret – 2 ens links får samme hash værdi. Hashen bliver derefter brugt til at substituere det oprindelige link med et relativt link:

Link.asp → ?LinkKey=f4f16b1e-df00-410c-8966-50d505932c67

Når brugeren derefter trykker på et link bruges hash-nøglen til at finde ud af, hvilken Gadget som skal opdateres, samt finde det link som er gemt i tabellen.

6.2.5. Indlejring på hovedside

For at holde styr på de forskellige Gadgets, lægges de gruppevis i asp:Placeholders. Designmæssigt er det fundet smartest at Gadgets er indrammet i et "blok" element som har float left. Dette medfører at man kan have et variabelt antal Gadgets i samme asp:Placeholder, idet Gadgets selv finder på plads. Gadgets får under oprettelse, angivet nogle egenskaber, så som maksimal bredde. Bredden angiver dels hvor bred Gadgetten er, men også hvor meget plads til det indre indhold der gives. Højden pr. Gadget er indtil videre ikke valgfri og skal besluttes i en endelig implementering.

```
GadgetList.add(new Gadget("G-Sim", "http://www.mdk-photo.com/Gadget",
        "GadgetHolder", 300));
```

```
GadgetList.add(new Gadget("G-Sim", "http://www.mdk-photo.com/dmi/pong.asp",
        "GadgetHolder", 300));
GadgetList.add(new Gadget("SSL-Site", "https://air.somedude.dk/sslsite",
        "GadgetHolder", 300));

GadgetList.add(new Gadget("G-Sim", "http:// www.mdk-photo.com/Gadget2",
        "GadgtHolderRight", 160));
```

Selve Gadgetten bliver udskrevet via **Gadget.Show(Placeholder PH)** Og indeholder en "Header" med 4 navigerings links (reload, remove, logout og reset) navn og titel, samt en "Body" med den konverterede HTML.

```
if (!PH.Controls.Contains(P)){
PH.Controls.Add(P);
}
P.Text = "<div style='margin:0px;width:100%;height:30px;background-color:#00AA00;color:white;font-
size:10pt;font-family:verdana;'>" +
   this.UniqueName + "  " +
   this.HtmlTitle + "<br />" +
   "<a href='?reload=" + this.ClientID +
   "' style='color:white;'>Reload</a> - " +
   "<a href='?remove=" + this.ClientID +
   "' style='color:white;'>Remove</a> - " +
   "<a href='?logout=" + this.ClientID +
   "' style='color:white;'>Log out</a> - " +
   "<a href='?reset=" + this.ClientID +
   "' style='color:white;'>Reset</a>" +
"</div>" +

"<div id=\"" + this.ClientID + "\" style='width:" +
   this.max_width +"px;height:270px;overflow:auto;'>" +
 GadgetHTML + "</div>";
```

6.2.1. Scripts og Styles

To aspekter ved hjemmesider som gør dem meget unikke, er deres brug af scripts og styles.

Stylesheets gør det muligt at style et helt HTML dokument efter nogle klassifikationsregler. Dokumentet defineres ved brug af attributterne Class og ID samt efter Tags, og kan derefter i stylesheetet styles hierarkisk derefter.

Eksempel:

** test **

Kan styles således:

.mydiv b{ color:#456; }

Hvilket medfører at kun indenfor mydiv klassen bliver stylet. I det tilfælde at en service-udbyder levere sit eget stylesheet, indsættes derfor en øvre klassificering på alle styles:

Eksempel:

** test **

Bliver til Gadget HMTL

```
<div id=”GADG1”><div class=”mydiv”><b>test</b></div></div>
```

Hvorefter stylesheetet bliver til

```
#GADG1 .mydiv b{ color:#456; }
```

Og vil derfor kun påvirke den HTML som findes i den pågældende Gadget.

På samme måde er det muligt at specificere Scripts til kun at påvirke én Gadget, men pga. tidsmangel var det ikke muligt at håndtere nok aspekter ved scripts, til at kunne få det med i projektet. Problemet med JavaScripts, er at elementer kan manipuleres efter ID, Name, Tag og Class samt DOM gennemgang. Ikke nok med at værdier skal passe sammen i Script og i HTML, så er det utrolig svært at fange de tilfælde hvor koden bliver dannet i strenge.

Løsningen i denne omgang, er defor blevet til at portalen henter scripts, og indsætter dem uændret i starten af Gadget’ens html. Det er derfor op til serviceudbyderen at navngive på en ansvarlig måde.

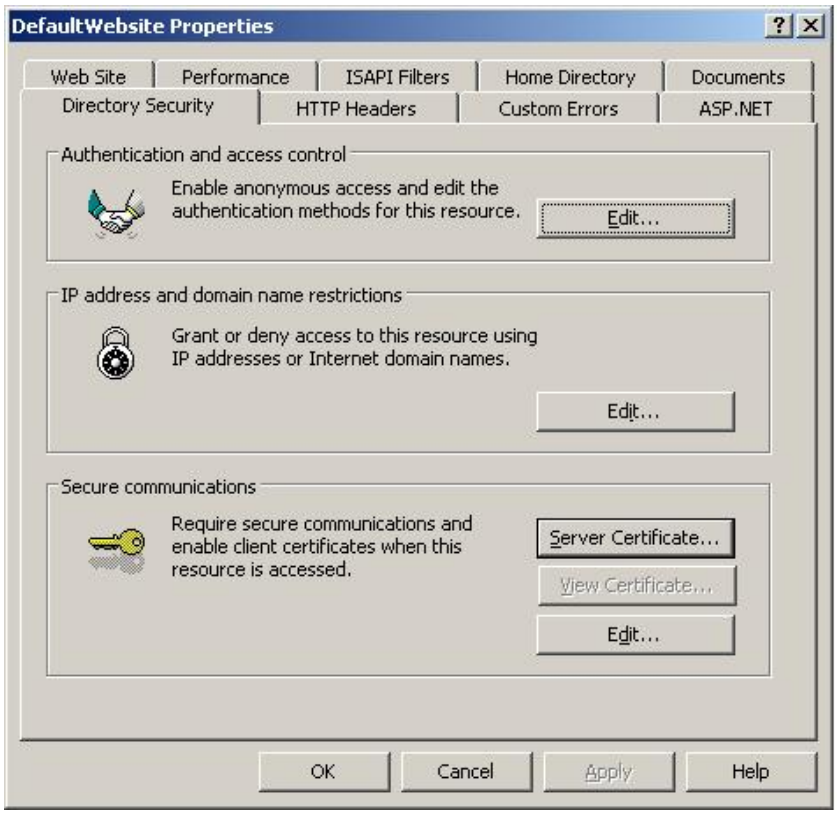
6.3. SSL

6.3.1. oprettelse og installation af servercertifikat

Følgende step skal udføres for at få et certifikat på serveren, og sætte den op til at kræve brug af certifikater:
Anmod om servercertifikatet
Få servercertifikatet
Slå anonym adgang fra
Kræv certifikat for adgang

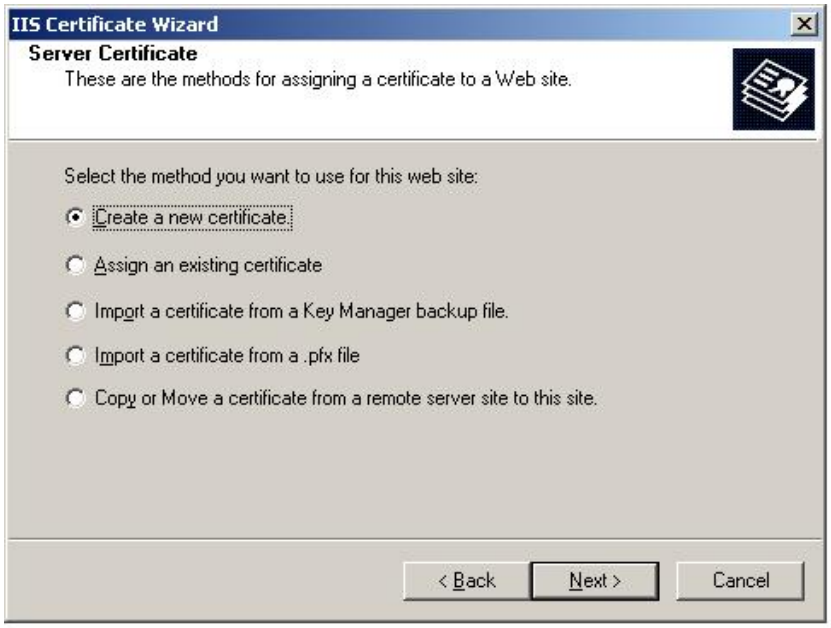
Anmod om servercertifikat

For at få et server certifikat, skal man starte med at requeste det. Dette gøres på IIS serveren ved at gå ind i egenskaber for sit default website, gå ind under directory security, og vælge server certificate.



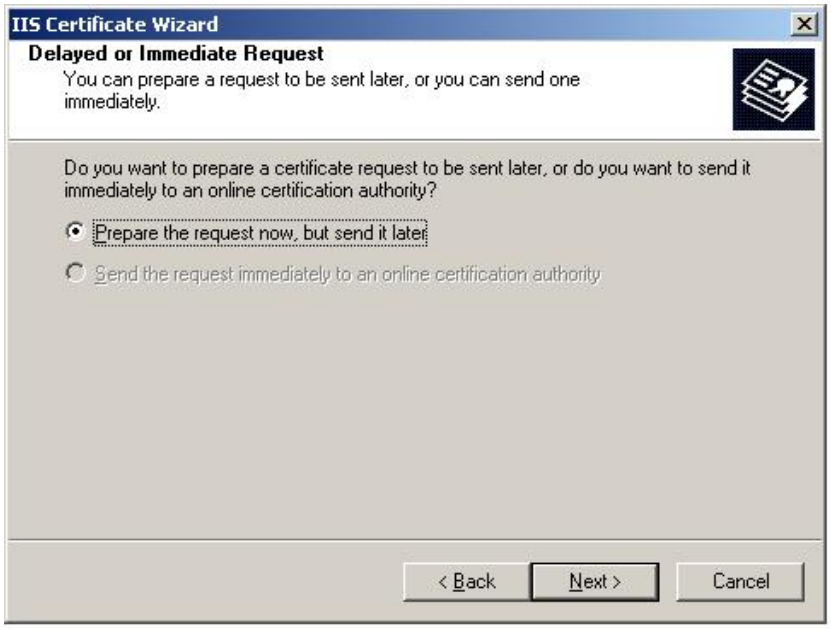
Figur 20 - egenskaber for website.

Der kommer en række valgmuligheder, og da vi ikke har et server certifikat i forvejen, vælger vi nu at oprette et nyt.



Figur 21 - oprettelse af nyt certifikat.

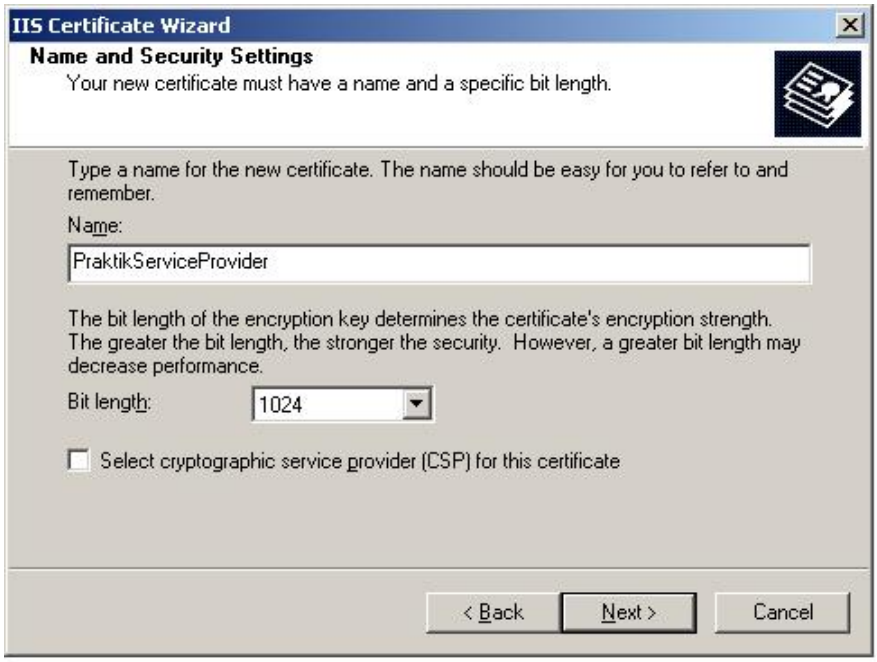
Her vælges der at vi skal gøre forespørgslen klar, men først sende den senere.



Figur 22 - klargør request af certifikat.

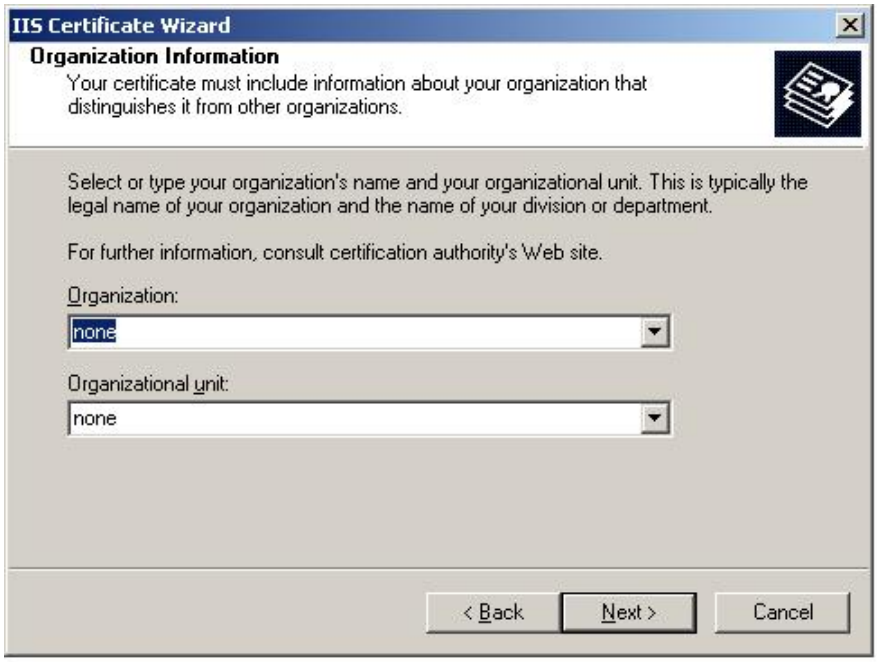
Der vil komme nogle vinduer op, hvor man skal udfylde noget information om det certifikat man er i gang med at lave.

Navnet på vores certifikat er ikke så vigtigt, men alle de andre felter der udfyldes er vigtige, da det er dem der testes på senere.



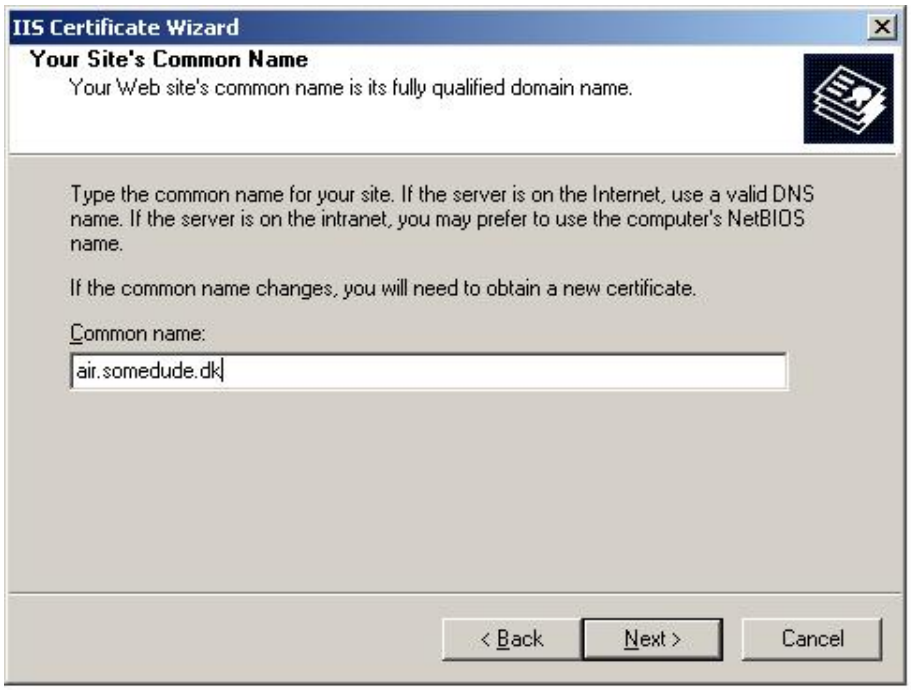
Figur 23 - navn på certifikatet.

Her udfyldes organisation og enhed.



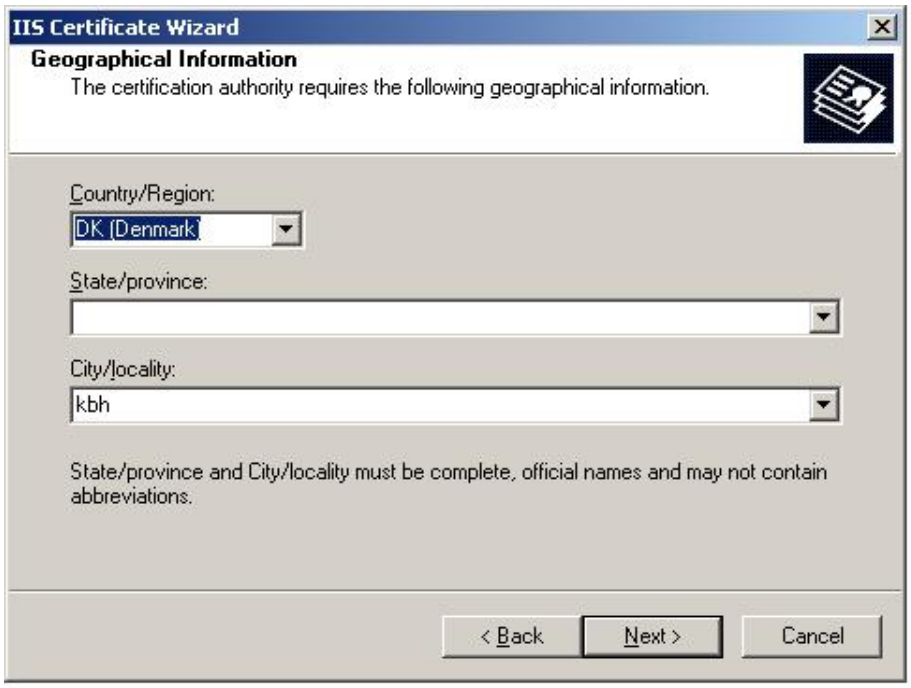
Figur 24 - navn på organisation og enhed.

Ved serverens almindelig navn, er det meget vigtigt at man skriver rigtigt. Hvis man f.eks. laver en stavefejl i dette navn, vil man ikke kunne få et gyldigt certifikat, og man vil derfor ikke have en troværdig side.



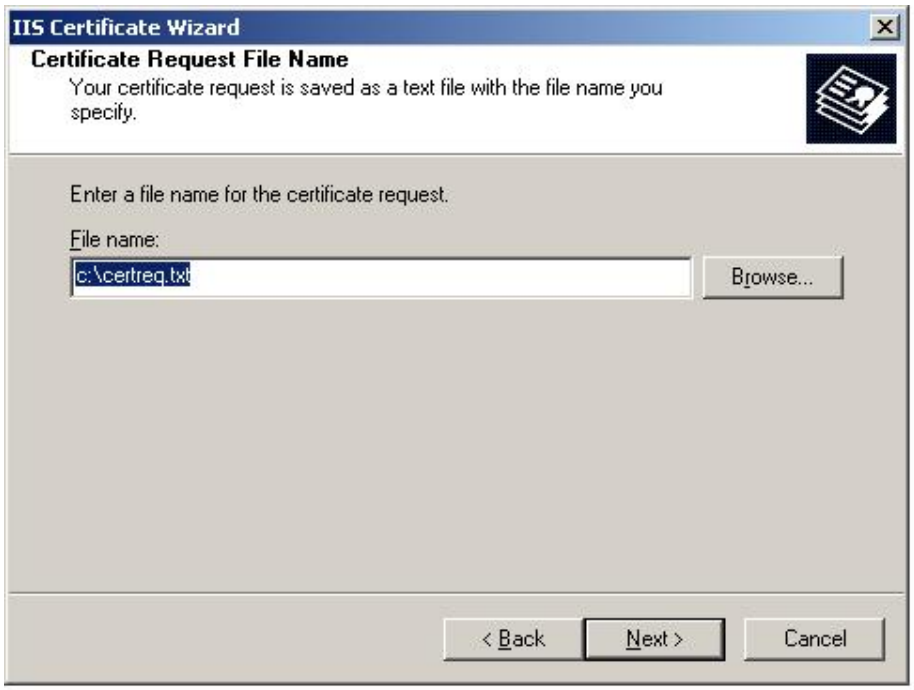
Figur 25 - domæne navn, til serveren hvorpå servicen er hostet.

Land, stat og lokalitet kan også udfyldes.



Figur 26 - land og by for serveren hvortil certifikatet skal bruges.

Her vælges der hvor certifikat anmodningen skal gemmes. Den skal bruges bagefter, når man skal have lavet sit certifikat.

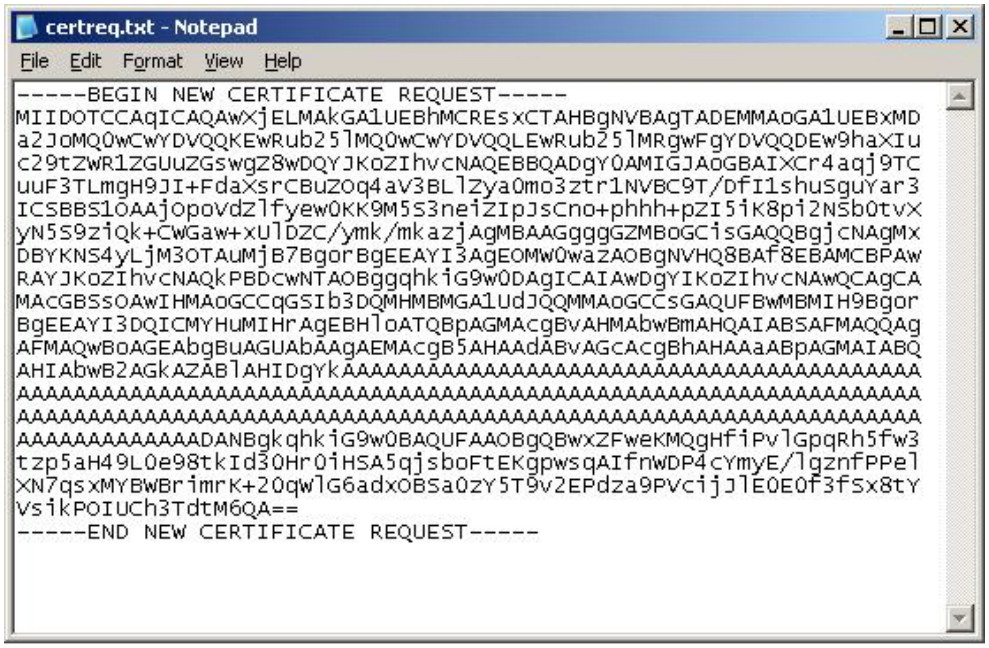


Figur 27 - placering til hvor certifikatet skal gemmes.

Efter at have udfyldt alle informationer, til det ønskede certifikat, får man en oversigt over oplysningerne omkring certifikat anmodningen.

Figur 28 - oversigt over indstillingerne til certifikatet.

Åbner man certifikat anmodningen, vil den se således ud:



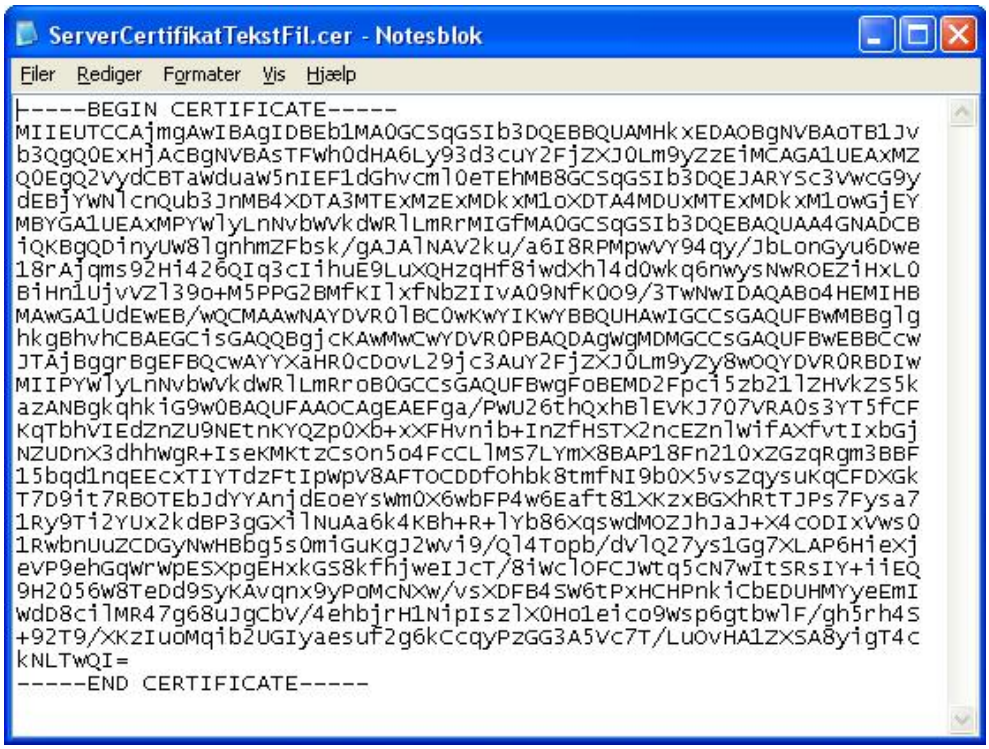
Figur 29 - Certifikat anmodningen som tekstfil.

Få server certifikatet

Efter at have lavet anmodningen til et certifikat, skal man have nogle til at udstede certifikatet, da det ikke er troværdigt hvis man selv udsteder det.

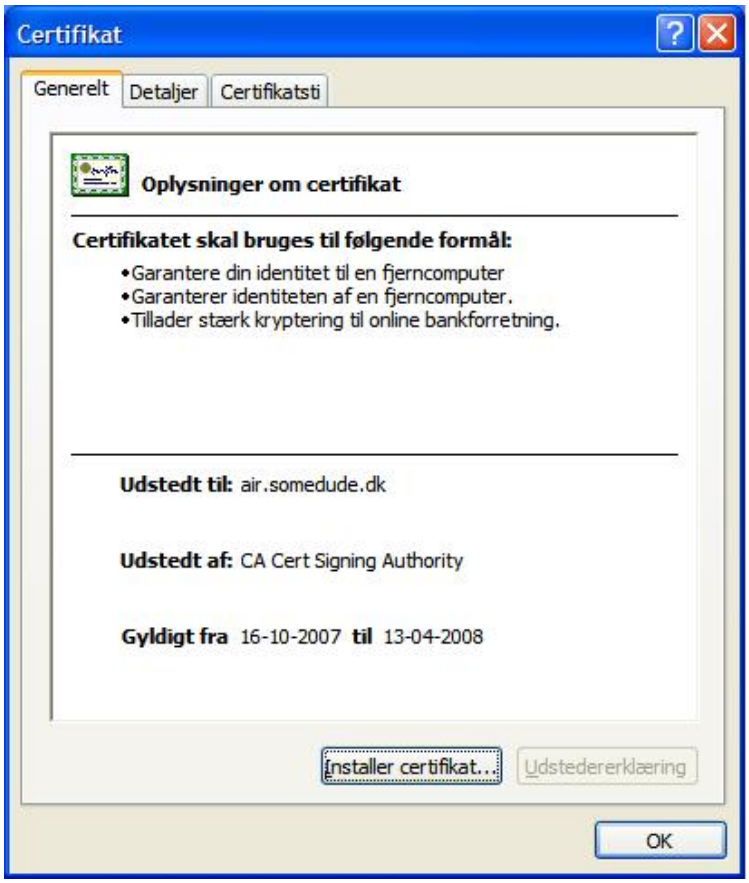
På Cacert.orgs side, kan man ved hjælp af certifikat anmodningen få udstedt certifikatet.

Ved at sætte teksten fra anmodningen ind på deres side, returnerer de med et certifikat til os. Hvis man åbner det med notesblok ser det ud på følgende måde:



Figur 30 - Det udstedte certifikat som tekstfil.

I stedet for at åbne certifikatet som en tekst fil, kan det nu åbnes som et almindeligt certifikat.



Figur 31 - det udstedte certifikat.

Vi kan nu gå ind og tilføje certifikatet til serveren der kører vores service. Alle forsøg på at tilgå den service vil nu blive præsenteret for vores certifikat, og det er så op til dem, at vurdere om de vil have tillid til det.

Slå anonym adgang fra

Efter at have fået tildelt et server certifikat, og tilføjet det til serveren, skal man sætte websitet op, så alle ikke kan få tilgang til siden. Da der som sagt kan være personfølsomme oplysninger i brug, skal en service køre som https, samt sørge for at det kun er betroede parter der har adgang til siden. Dette sættes siden op til ved redigere i indstillingerne for websitet på IIS serveren.

Figur 32 - administration af autentificerings metoder for websitet.

Vi er interesseret i at vide hvem det er der henter vores side, og har derfor slået anonym adgang fra. På billedet ovenover kan man se, at man har en række muligheder for at give adgang til siden. Man kan f.eks. vælge, at lade brugeren skrive brugernavn og adgangskode for at komme ind, ved at bruge Windows identifikation. Dette er vi selvfølgelig ikke interesseret i, da det hele gerne skulle ske automatisk.

Kræv certifikat for adgang

På secure communications under egenskaber for web sitet, kan man vælge hvad der skal ske med klientcertifikater, man kan vælge mellem følgende:

Ignorere klient certifikater

Acceptere klient certifikater

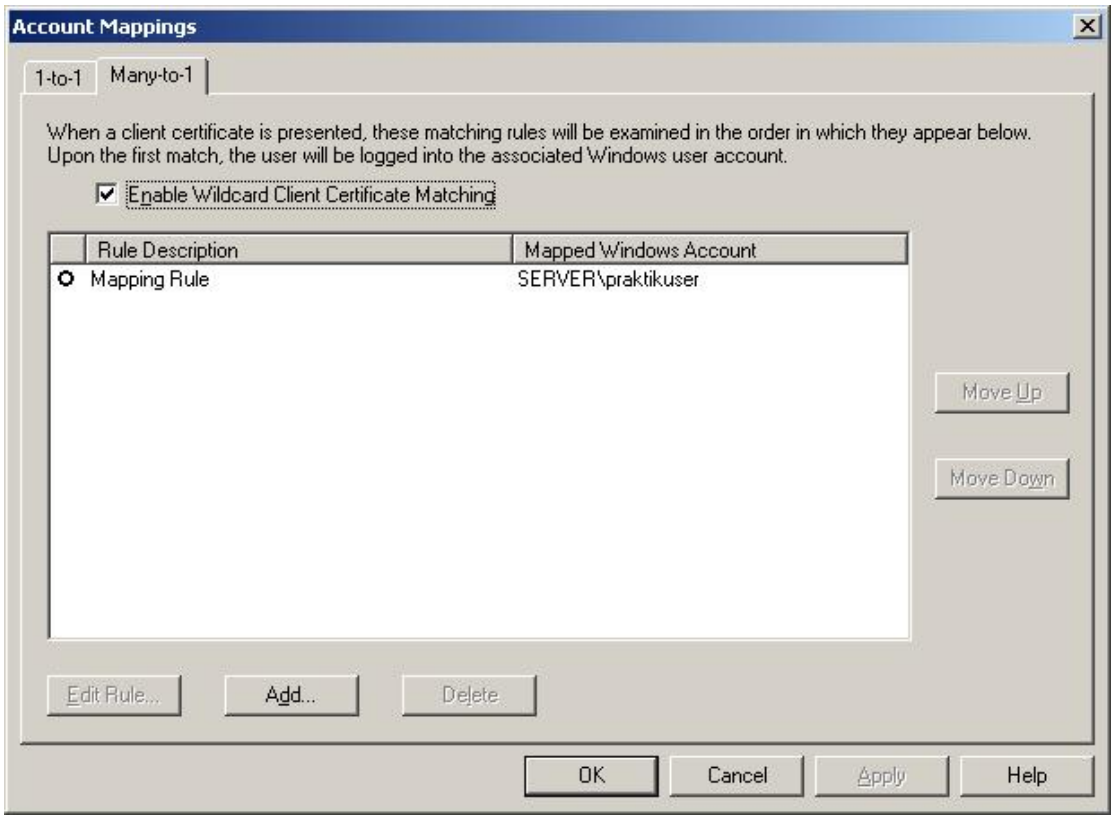
Kræve klient certifikater

For at vi ikke selv skal blive nægtet adgang til siden, sætter vi den op til at give os adgang hvis vi har et klientcertifikat. Vi har valgt at der skal kræves et certifikat, da der ikke skal være adgang, hvis man ikke er i besiddelse af certifikatet.

Figur 33 - indstillinger for sikkerheden af websitet.

Da vi har slået anonym adgang fra, og kræver et certifikat, skal vi give adgang til siden, hvis certifikatet vises.

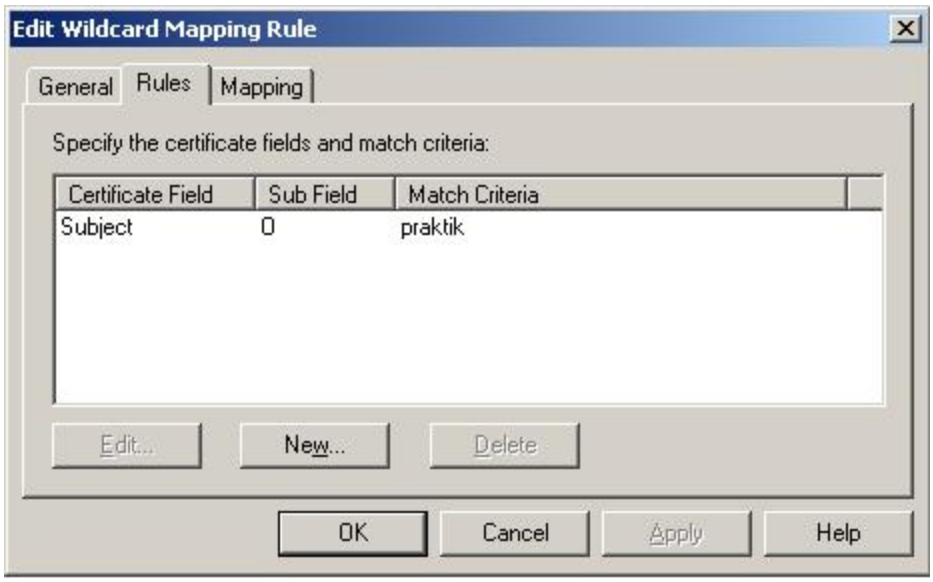
Det gør vi, ved at sætte serveren op til at overføre et certifikat til en Windows bruger konto.



Figur 34 - oversigt over mapping regler.

Til dette formål har vi lavet en bruger der hedder pratikuser. Ved at kalde siden, ved hjælp af et klientcertifikat, vil ens rettigheder blive overført til praktikuser, og man er identificeret som den bruger. Vi har valgt at lave reglen i many-to-1. I princippet kunne vi lige så godt have lavet overførslen i 1-to-1, da vi kun har en portal. Men hvis man nu havde skullet bruge servicen flere steder, og man ikke vil lave en regel til hver enkel bruger, kan man bruge many-to-1. Laver man nogle ændringer, kan man også nøjes med at rette det til et sted.

I vores mapping eksempel har vi valgt at se efter en enkelt ting på certifikatet.
Her ses den regel vi har valgt at lave:



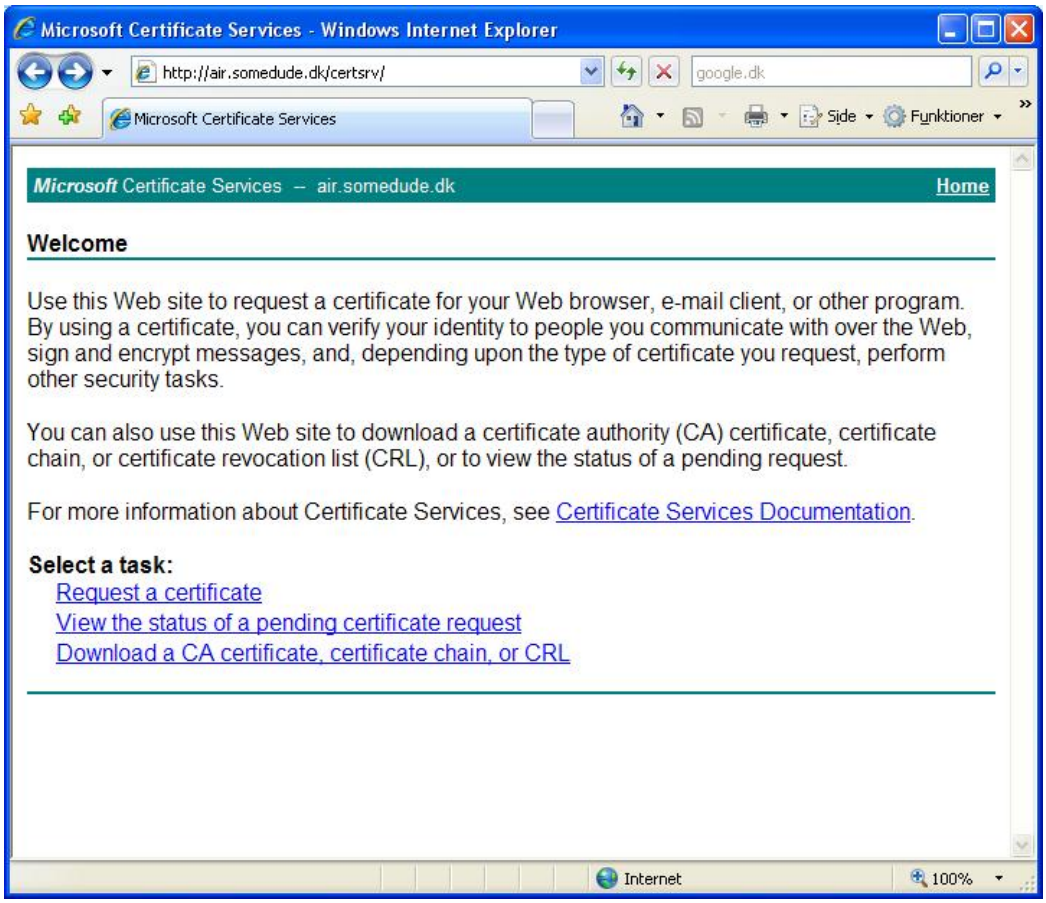
Figur 35 - mapping regel.

Hvis man tilgår siden med et certifikat, hvor der står praktik i subject, vil man blive overført til praktikuser, og få adgang, mens man med et certifikat hvor der ikke står praktik i subject vil blive nægtet adgang.
Det ville ikke være et problem at sætte flere krav op til certifikatet, men da vi kun har behov for at vise hvordan det virker, har vi valgt ikke at sætte flere regler op.

Det er også i secure communications at siden sættes op til at bruge SSL, så vi får en sikker forbindelse. Dette er muligt, netop fordi vi har lavet et servercertifikat.
Som det kan ses på FIGUR hend er vores side sat op til at køre med en SSL forbindelse der bruger 128 bit kryptering.

6.3.2. oprettelse og brug af klientcertifikat

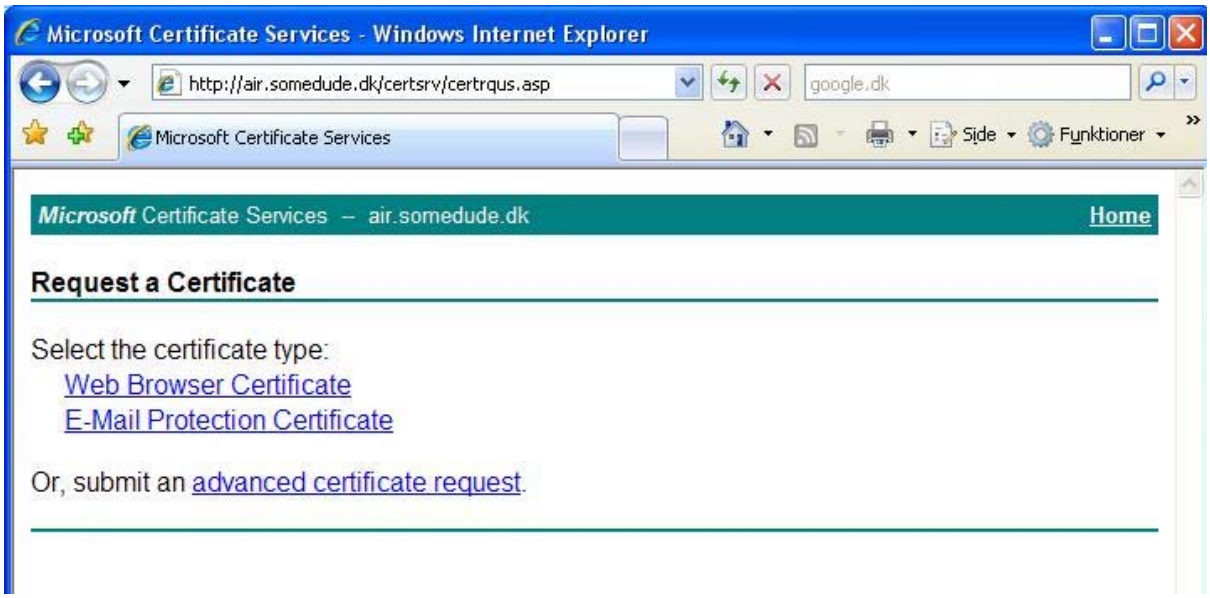
Efter at have sat serveren op, skal vi have tildelt et certifikat, så vores klient kan tilgå serveren, og derved bruge servicen der er hosted derpå.
For at få et klientcertifikat, har vi først sat serveren op til at køre som en certification authority. Det vil sige at serveren kan udstede vores klient certifikater.
Ved at tilgå certsrv, som er en service der kører på serveren, kan man ansøge om et certifikat.



Figur 36 - Microsofts Certificate Service hos service udbyderen.

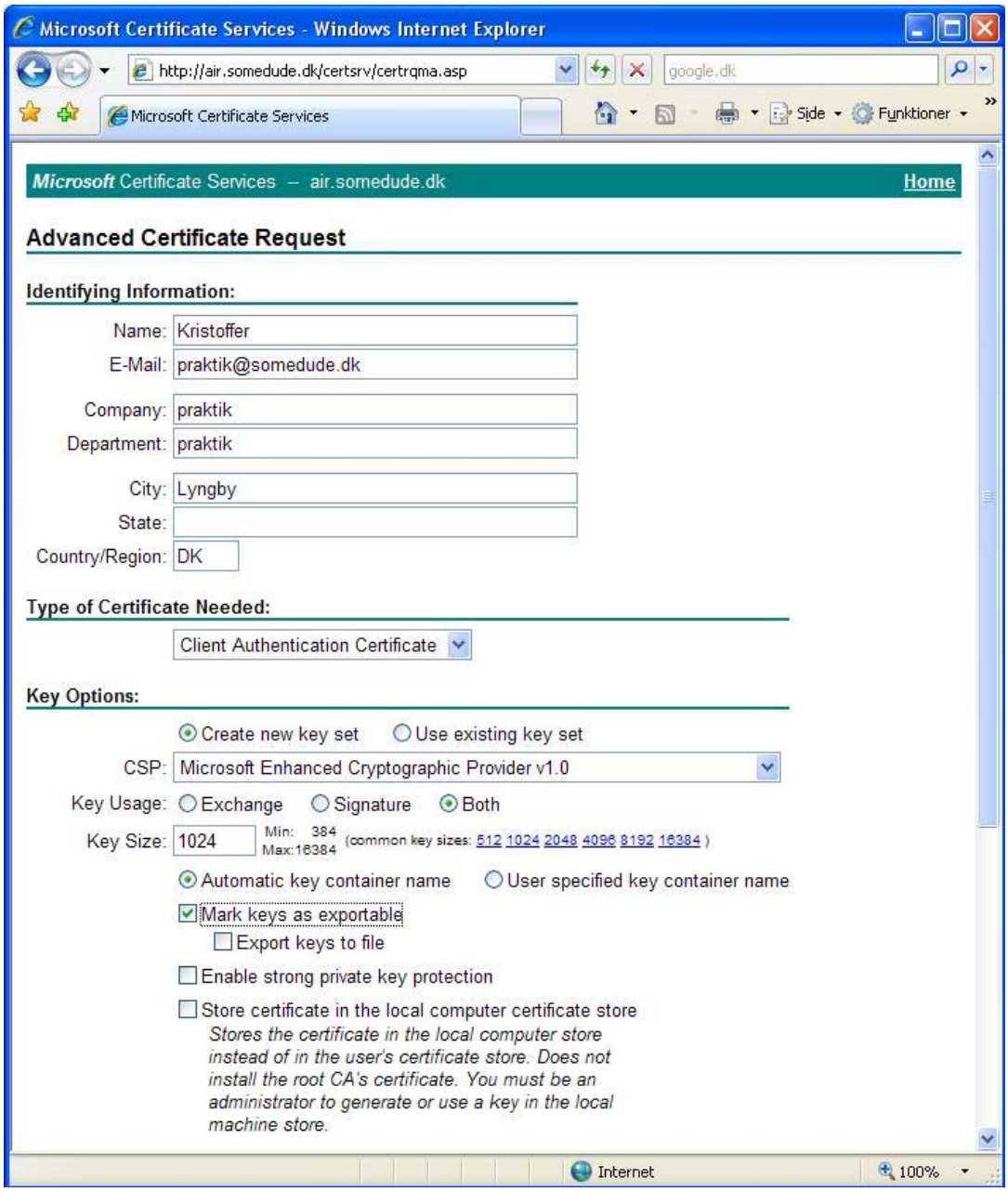
Man skal først vælge om man vil anmode om et certifikat, se status på en anmodning der er under behandling eller om man vil downloade serverens root certifikat.

Da vi skal bruge et certifikat til vores klient (portalen er klient i dette tilfælde), vælger vi at anmode om et nyt certifikat.



Figur 37 - anmodning af certifikat.

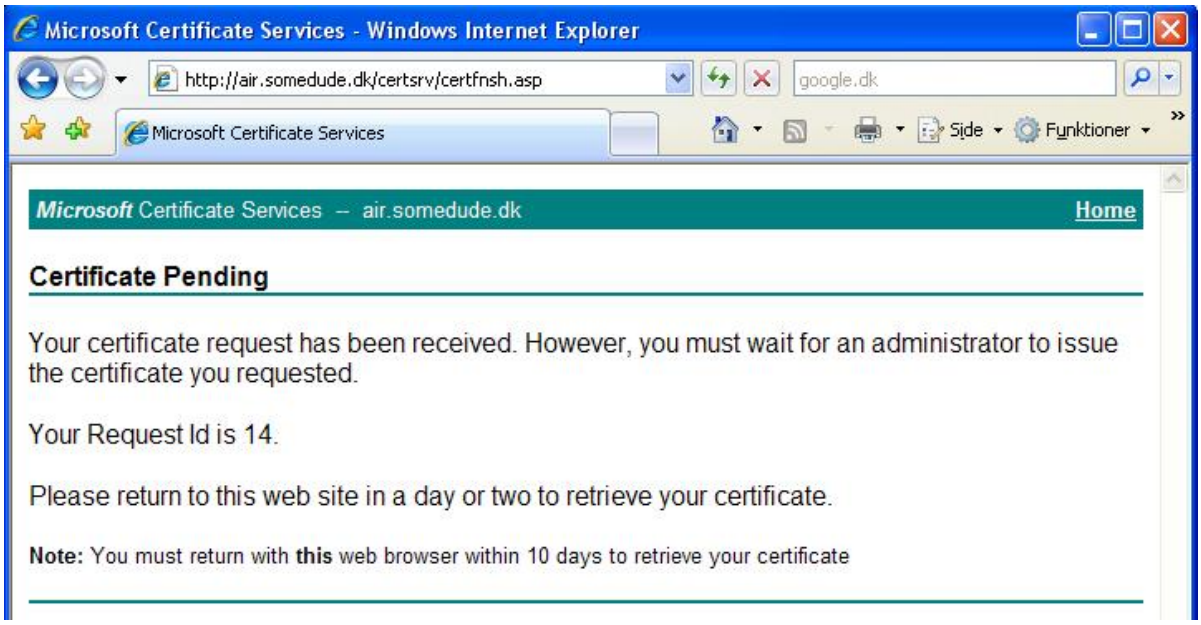
Vi vælger her en avanceret certifikatanmodning.



Figur 38 - avanceret certifikat anmodning.

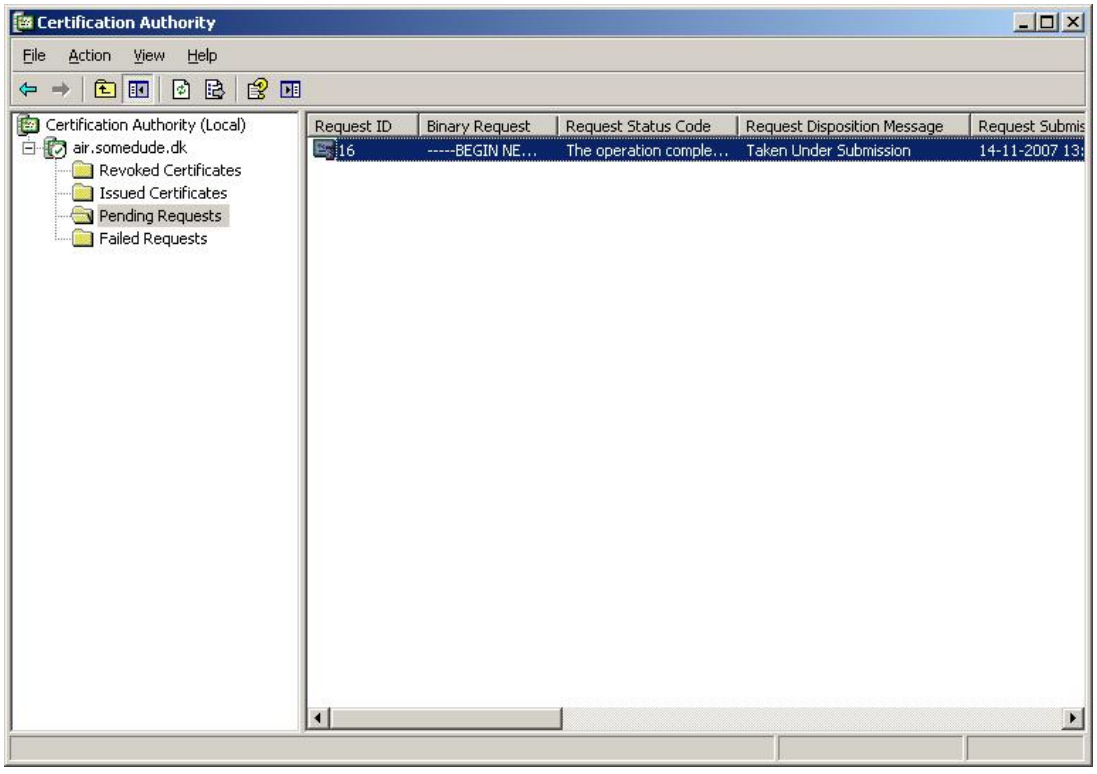
Efter udfyldning af identifikationsinformationerne, markerer vi nøglen som eksporterbar. Ellers kan vi kun få installeret certifikatet på en maskine, og ikke have certifikatet liggende som en fil. Vi har også her valgt at certifikatet skal være et klientcertifikat.

Man trykker derefter på submit, hvorefter man får en bekræftelse om at certifikat anmodningen er blevet modtaget af serveren.



Figur 39 - certifikat anmodning sendt af sted.

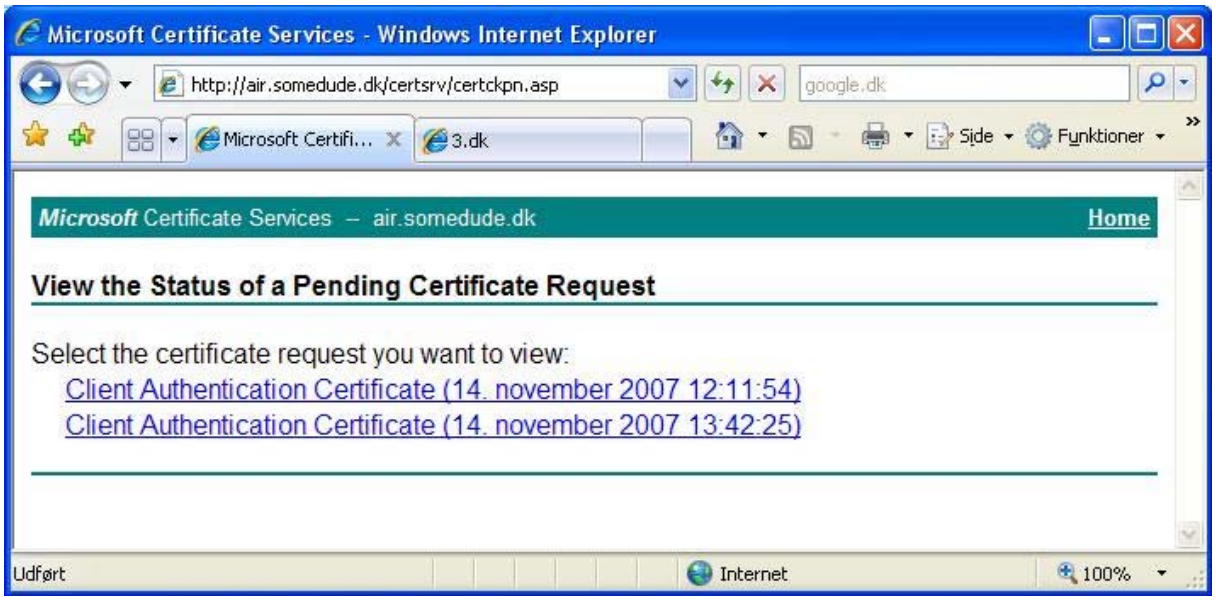
Efter at have sendt anmodningen af sted, skal serveren bekræfte at den vil udstede certifikatet.



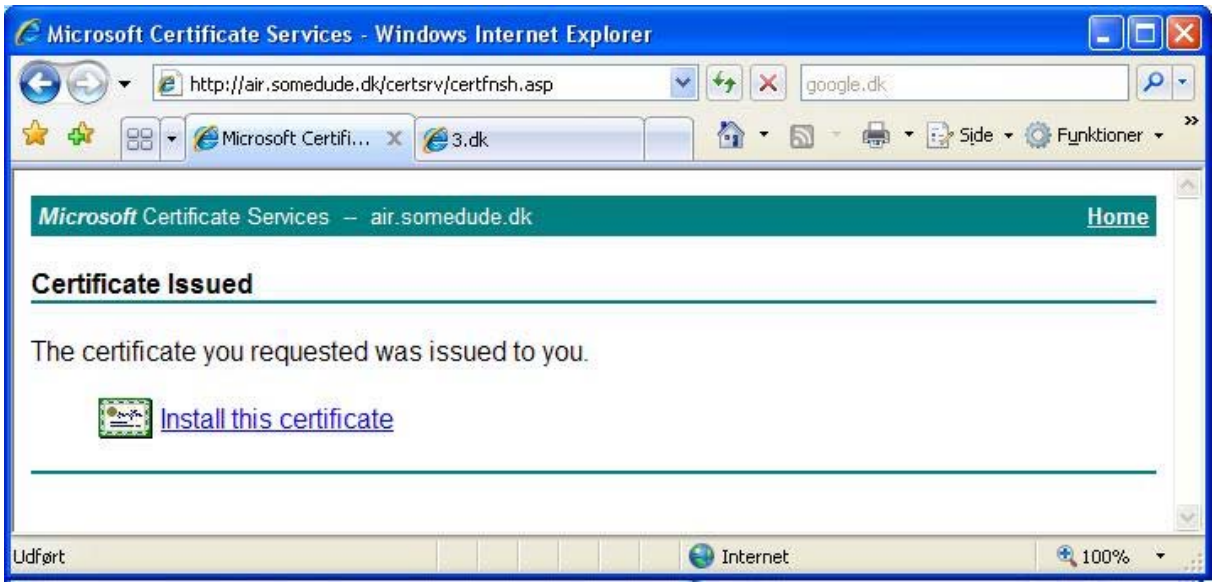
Figur 40 - certification authority.

På serveren er anmodningen til certifikatet kommet inde under pending request i Certification Authority. Vælger man her at udstede certifikatet, vil klienten kunne hente sit certifikat via nettet.

Hvis man på figur 39 havde valgt at se status på et certifikat der er under behandling, får man mulighed for at installere sit certifikat.



Figur 41 - oversigt over certifikat anmodninger.



Figur 42 - certifikatet er udstedt og kan installeres.

Efter installationen kan man nu forbinde til websitet, hvis man vel at mærke bruger den browser, man brugte til at installere certifikatet med. Da det ikke er nok at bruge browseren i vores tilfælde, er vi nødt til at eksportere certifikatet. Dette kan man gøre både fra browseren, og fra certificate store, men man får kun den private nøgle med, fordi vi har valgt at certifikatet skulle være eksporterbar.

Vælger vi ikke at nøglen skal være eksporter bar, vil certifikatet kun ligge i det certificate store som det bliver installeret i. Da man er logget på med en almindelig bruger i Windows, vil certifikatet, og derfor også den private nøgle dertil, blive installeret i den aktuelle brugers certificate store. Det vil der ikke være adgang til, når man hoster siden på IIS serveren, da certificate store så vil blive kaldt i form af en bruger der hedder ASPNET.
Alternativt skal man logge på Windows med ASPNET brugeren, og derefter installere certifikatet, men da ASPNET brugeren er styret af Windows, og dermed også passwordet, er det ikke noget man lige gør.

6.3.3. Adgang til privat nøgle

For at få adgang til den private nøgle til et certifikat kræves det, at ASPNET brugeren har adgang til den private nøgle til certifikatet. ASPNET brugeren kan få adgang til nøglen, hvis certifikatet er installeret i den lokale computers certifikat store, hvis man vel at mærke har givet adgang til det. Dette gøres med et værktøj fra Microsoft der hedder winhttpcertcfg.exe. Ved hjælp af dette program, kan man bestemme hvilke brugere der har adgang til et givent certifikats private nøgle.[2]
Her følger først en oversigt over hvilke ting man kan i programmet:

```
To list accounts which have access to the private key for
specified certificate:
    winhttpcertcfg -l -c CertLocation -s SubjectStr

To grant access to private key for an account with
specified certificate that is already installed:
    winhttpcertcfg -g -c CertLocation -s SubjectStr -a Account

To import a certificate plus private key from a PFX file:
    winhttpcertcfg -i PFXFile -c CertLocation

To remove access to private key for an account with
specified certificate:
    winhttpcertcfg -r -c CertLocation -s SubjectStr -a Account
```

Figur 43 - oversigt over kommandoer i winhttpcertcfg.

For at se hvem der har adgang til den private nøgle fra starten af, kan man vælge at få vist en liste over brugere.

```
C:\Programmer\Windows Resource Kits\Tools>winhttpcertcfg.exe -l -c local_machine
\my -s kristoffer
Microsoft (R) WinHTTP Certificate Configuration Tool
Copyright (C) Microsoft Corporation 2001.

Matching certificate:
E=praktik@somedude.dk
CN=Kristoffer
OU=praktik
O=praktik
L=Lyngby
C=DK

Additional accounts and groups with access to the private key include:
    SOMEDUDE\Kristoffer
    NT AUTHORITY\SYSTEM
```

Figur 44 - liste over kontoer der har adgang til det fundne certifikat.

Hvor –l (list) fortæller at vi vil have en liste frem over brugere, -c local_machine\my at vi skal se i den lokale computers certificate store og –s Kristoffer at den skal se efter alle certifikater der hedder noget med Kristoffer.

Vi kan nu se at det kun er brugeren der har indlæst certifikatet, og en system bruger der har adgang til certifikatet.

For at give ASPNET brugeren adgang, skal vi i stedet for –l bruge –g (grant), som giver adgang til et certifikat, efterfulgt af –a (account), der fortæller hvilken bruger der skal have adgang til certifikatet.

```
C:\Programmer\Windows Resource Kits\Tools>winhttpcertcfg.exe -g -c local_machine
\my -s kristoffer -a ASPNET
Microsoft (R) WinHTTP Certificate Configuration Tool
Copyright (C) Microsoft Corporation 2001.

Matching certificate:
E=praktik@somedude.dk
CN=Kristoffer
OU=praktik
O=praktik
L=Lyngby
C=DK

Granting private key access for account:
    SOMEDUDE\ASPNET
```

Figur 45 - ASPNET brugeren får adgang til den private nøgle.

Vælger vi at få vist listen igen, kan vi nu se at ASPNET brugeren er kommet på listen.

```
C:\Programmer\Windows Resource Kits\Tools>winhttpcertcfg.exe -l -c local_machine
\my -s kristoffer
Microsoft (R) WinHTTP Certificate Configuration Tool
Copyright (C) Microsoft Corporation 2001.

Matching certificate:
E=praktik@somedude.dk
CN=Kristoffer
OU=praktik
O=praktik
L=Lyngby
C=DK

Additional accounts and groups with access to the private key include:
    SOMEDUDE\Kristoffer
    NT AUTHORITY\SYSTEM
    SOMEDUDE\ASPNET
```

Figur 46 - liste over kontoer der har adgang til det fundne certifikat, ASPNET brugeren er nu med på listen.

Der vil nu være adgang til den private nøgle, for dette certifikat, når vi kører en webside, på IIS.

6.3.4. Hentning af klientcertifikat

Da de fleste serviceudbydere vil kræve et klientcertifikat, vil det være uhensigtsmæssigt at hente certifikaterne hver eneste gang, en bruger logger på portalen.
Vi har derfor oprettet en kollektion, der kan indeholde vores certifikater.

private static X509CertificateCollection certCol

Ved at lave den static, gør vi at alle de brugere der logger på portalen, bruger den samme kollektion, og vi skal derfor kun hente et certifikat til en given service, en gang for alle brugere.
Vi har gjort det muligt, både at hente certifikater fra filer og fra certificate store. Når en sikker service skal loades, ser vi først om certifikatet allerede ligger i vores kollektion.

```
if (certCol.Count != 0) //ser om vi har certifikatet i forvejen
{
        foreach (X509Certificate cert in certCol)
        {
                if (cert.Issuer == "CN=" + objURI.Host) //tilføjer certifikatet der passer til vores host
                {
                        objWebRequest.ClientCertificates.Add(cert);
                        found = true;
                        break;
                }
        }
}
```

Finder vi et brugbart certifikat, breaker vi ud, og fortsætter med at loade servicen. Ligger der ikke et certifikat vi kan bruge, eller er der slet ingen certifikater i vores kollektion endnu, prøver vi at hente et certifikat.
Vi ser først om det ønskes vi skal loade certifikatet fra en fil.

```
if (certName != "")
```

certName, er en string der indeholder navnet på et certifikat. Er denne string sat til et navn, henter vi certifikatet fra den pågældende fil. Det eneste krav vi har sat, for at hente et certifikat fra en fil, er at certifikatet skal gemmes et bestemt sted. I vores tilfælde har vi valgt at det skal gemmes i en mappe der hedder cert i roden af c-drevet. Dette er sat op i web.config.

```
<add key="CertPath" value="C:\\Cert\\"/>
```

Ved at sætte stien til certifikaterne i web.config, sikrer vi os, at der ikke er en hacker, der kan se I koden på siden, hvor certifikaterne er gemt, og derved hente dem.
Samtidig er det let at ændre stien, hvis det bliver besluttet at certifikaterne skal gemmes et andet sted.
Efter at have indlæst navnet på certifikatet, prøver vi at hente certifikatet.

```
if (certName != "") //hvis man vil bruge et certifikat der er gemt på disken
            {
            try
            {
            cert = X509Certificate.CreateFromCertFile(certPath + certName);

            if (DateTime.Parse(cert.GetExpirationDateString()).CompareTo(DateTime.Now) <= 0)
            {
                        Exception ex = new Exception("Certifikatet er udløbet");
                        throw ex;
            }

            //certifikat skal kun tilføjes hvis det er gyldigt
            // og ikke findes i certCol i forvejen
            if (!certCol.Contains(cert))
            {
                        certCol.Add(cert);
            }
   certName = "";
            }
   catch (Exception)
   {
                        Exception ex = new Exception("Der blev ikke fundet et gyldigt certifikat");
      throw ex;
   }
}
```

Vi har valgt kun at tjekke om datoen er udløbet på klient certifikatet, og overlader resten til serviceudbyderen, da det er deres ansvar at tjekke certifikaterne der kommer ind hos dem. Skulle vi komme ud for at det ikke er et certifikat vi får indlæst, bliver der smidt en exception, der fortæller at det ikke er et gyldigt certifikat vi har fundet.
Er datoen ikke udløbet på et indlæst certifikat, løber vi vores certifikat kollektion igennem, og ser om det skulle være der i forvejen, ellers bliver det tilføjet.

Er der ikke angivet navnet på et certifikat, henter vi certifikatet fra certificate store.
For at få adgang til certificate store, gør vi brug af CAPICOM librariet [3]. CAPICOM er et library udgivet af microsoft, der gør at man kan få adgang til certificate store, via .net.
Bruger man ikke CAPICOM, men prøver at læse fra certificate store vil man få en ”Attempted to read or write protected memory” fejl.
Capicom kan hentes hos Microsoft [4] og ved at lave en .dll af CAPICOM kan man inkludere .dll filen i sit projekt, og igennem librariet få adgang til certificate store.

Vi har lavet to metoder, vi kan bruge når vi vil hente fra certificate store. En der søger local_machine_store igennem for certifikater udstedt af en bestemt udsteder, og en metode der søger med et bestemt filter, hvor filteret er en string i certifikatet.

Metoden der finder et certifikat med en given udsteder fungerer således:

```
public X509Certificate getCertFromIssuer(string issuer)
   {
      oStore = new StoreClass();

      oStore.Open(
      CAPICOM_STORE_LOCATION.CAPICOM_LOCAL_MACHINE_STORE,
      storeName, CAPICOM_STORE_OPEN_MODE.CAPICOM_STORE_OPEN_EXISTING_ONLY |
      CAPICOM_STORE_OPEN_MODE.CAPICOM_STORE_OPEN_READ_ONLY);

      oCerts = (Certificates)oStore.Certificates;
      oCerts = (Certificates)oCerts.Find(
        CAPICOM_CERTIFICATE_FIND_TYPE.CAPICOM_CERTIFICATE_FIND_ISSUER_NAME,
issuer, false);

      foreach (Certificate ocert in oCerts)  //Certificates is IEnumerable
      {
         Certificate firstcert = (Certificate)oCerts[1];

         ICertContext iCertCntxt = (ICertContext)firstcert;
         int certcntxt = iCertCntxt.CertContext;
         IntPtr hCertCntxt = new IntPtr(certcntxt);
         if (hCertCntxt != IntPtr.Zero)
         {     //use certcontext from managed code
            return foundcert = new X509Certificate(hCertCntxt);
         }
      }
      return foundcert;
   }
```

Vi starter med at oprette en StoreClass, som vi sætter til at åbne local_machine_store.
Vi vælger nu at søge disse certifikater igennem, og finder alle dem der er udstedt af den udbyder, hvis navn metoden bliver kaldt med. Bliver der fundet et certifikat, vælger vi at returnerer det første der bliver fundet, og bruger dette til at hente servicen med.

Vi havde kunnet vælge at hente samtlige certifikater fra certificate store, og gemme dem i vores X509CertificateCollection, men da det kan være at der kører andre applikationer på samme IIS server, der gør brug af klientcertifikater, eller at der af en anden grund er installeret certifikater, som vi ikke skal bruge for at hente de forskellige services, har vi valgt ikke at gøre dette.
Grunden til vi først søger vores kollektion af certifikater igennem, er at alle certifikater, der skal bruges, vil være at finde her, efter kort tid.

6.3.5. Validering af servercertifikat

Når vi kalder op til en service der kører med en SSL forbindelse, vil vi blive mødt med et servercertifikat, hvor det er op til os at vurdere om det er gyldigt.
Ved hjælp af

```
ServicePointManager.ServerCertificateValidationCallback = new
RemoteCertificateValidationCallback(ValidateServerCertificate);
```

Kan vi lave en validation af servercertifikatet.

```
  private static bool ValidateServerCertificate(
 object sender,  X509Certificate certificate,
 X509Chain chain, SslPolicyErrors sslPolicyErrors)
  {
     string issuerString = certificate.Issuer;
     //CaCert issuer string
     //E=support@cacert.org, CN=CA Cert Signing Authority, OU=http://www.cacert.org, O=Root CA
     string cn = Regex.Match(issuerString, "(.+?)(CN=.+?)(,|$)").Groups[2].ToString();
     string ou = Regex.Match(issuerString, "(.+?)(OU=.+?)(,|$)").Groups[2].ToString();
     string o = Regex.Match(issuerString, "(.+?)(O=.+?)(,|$)").Groups[2].ToString();

     //ser om issuer er den vi regner med
     if (!"CN=CA Cert Signing Authority".Equals(cn))
     {
        return false;
     }
     //if (!"TDC OCES CA".equals(cn) ||!"TDC".equals(o) ||!"DK".equals(c)) {throw new
ServletException("Unknown issuer: " + issuerString);

     if (DateTime.Parse(certificate.GetExpirationDateString()).CompareTo(DateTime.Now) <= 0 &&
sslPolicyErrors != SslPolicyErrors.None)
     {
        return false;
     }
     if(!certificate.Subject.Equals("CN=" + objURI.Host))
     {
        return false;
     }
     // returnerer true hvis server certifikatet er gyldigt
     return true;
  }
```

Vi starter med at lave tre strenge ud fra issueren af server certifikatet. Dette gør vi for at tjekke om certifikatet er udstedt af den authority, som vi forventer. I vores tilfælde forventer vi at det er Cacert der har udstedt certifikatet.
Derefter undersøger vi om udløbsdatoen på certifikatet er udløbet, samt om der er nogle SslPolicyErrors på certifikatet.

Til sidst ser vi om servercertifikatet er udstedt til det domæne, hvorpå certifikatet er installeret og vi er ved at hente servicen fra.
Er alle tingene opfyldt, vil vi returnere en boolean der er true, og vi vil således fortsætte med at loade servicen. Er certifikatet ikke valideret korrekt, vil vi returnere false, og ikke tillade servicen at blive vist på ”Min Side”.

6.3.6. Sikkerhed på service

Selvom at der skulle ske en fejl, med hensyn til at der ikke må logges på uden certifikat, vil det stadigvæk blive fanget af servicen.
Dette gør vi ved at tjekke i koden, om der bliver præsenteret et certifikat.

```
if (cert.IsPresent)
        certDataLabel.Text = cert.Get("SUBJECT O");
else
        Server.Transfer("NoCert.aspx");
```

Ved at se om der bliver brugt et certifikat på siden, kan vi på den måde redirecte til en anden side, hvis der ikke blev brugt et certifikat.

6.4. Gadget udvikling

Dette kapitel skrives dels som en vejledning til udviklere, som skal skabe Gadgets og dels som dokumentation for den mængde af features, som Proxien understøtter i den nuværende version.
Undervejs i projektet blev der udviklet nogle simple Gadgets, som skulle bruges til at, teste om systemet virkede. Disse bygger på de regelsæt som hermed gives.

6.4.1. Grundregler

Når man skal udvikle en Gadget, skal den udvikles med udgangspunkt i at være begrænset i størrelse. Det er derfor en god ide enten at udvikle til en iFrame med en given højde og bredde, eller direkte igennem en fremtidig testside. Med det nuværende design, kan Gadgets laves

i 3 bredder: **Wide 610px**, **Normal 300px** og **Slim 160px** – deres maksimale indholdshøjde er **270px**. Selvom dette 'vindue' ikke er ret stort, vil overflow give anledning til scroll.

Gadget'en skal skrives som om den var en almindelig hjemmeside, med **<html>, <head>** og **<body>**, men det er indholdet fra **<body>** som bruges af portalen. Alt andet bliver klippet væk.

Grundtanken er at lave en side, der fungere på samme måde som PHP/ASP. Dvs. en simpel webside uden specielle Framework funktioner, som man kender dem i ASP.NET (postback, Ajax).

Der er ikke nogen krav til HTML-version, men tags skal være afsluttet. Attributter bør have enkelt eller dobbelt pling rundt om værdierne.

6.4.2. Scripts og Styles

Scripts og Stylesheets bør angives i <head> med følgende tags:

```
<script src="URL"></src>
<link rel="stylesheet" type="text/css" src="URL" />
```

eller inline

```
<script>/* Javascript */</script>
<style> /* Stylesheet */ </style>
```

I begge tilfælde hentes hele indholdet fra alle funde forekomster, ind i 2 strenge: **scripts** og **styles** som derefter processers.

Styles får tilføjet et 'namespace' – via css class, hvilket betyder at man ikke behøver tænke på om ens Stylesheet overskriver allerede definerede klasser og lign. Til udvikling bør stylesheet'et fra borger.dk benyttes!

Scripts og tilførende ID-felter skal derimod oprettes med omtanke, ID-felter vil give et problem hvis der skal eksistere flere kopiere af samme Gadget på samme tid. Funktions navne og variable skal skrives med et præfix som er unikt for producenten. Scripts bliver ikke behandlet, hvilket medføre at 2 ens Gadgets, vil medføre 2 kopier af samme script!

6.4.3. Session, Cookies og State

Når en side er hentet fra en service, bliver den behandlede HTML gemt i Gadget'ens cache, det er derfor op til udvikleren at sørge for navigeringslinks på hver eneste side. I den nuværende version, er der påsat en header som gør det muligt for brugeren at: **Reset, Refresh, Logout og Remove** – Logout sletter cookies (og session), Remove fjerner Gadget'en fra portalen, Refresh henter siden med de samme parametre som sidst og Reset henter den initiale side som Gadget'en blev oprettet med, samt sletter cookies.

6.4.4. PostBack, AJAX, Objects og Applets

PostBack og AJAX er pt. ikke understøttet.

Objects og Applets bliver ikke behandlet af portalen, hvilket ofte vil betyde, at de ikke virker.

6.4.5. Forms og inputs

Alle almindelige forms er understøttet til at gå igennem portalen, det nemmeste er blot at angive action attributten og lade resten ske automatisk. Hvis der skal sendes filer med enctype=multipart/form-data, skal action være et absolut link til en side som kan modtage filen, samt target=_blank.

6.4.6. Registrering af Gadget

Registrering foregår ved at tage kontakt til Borger.dk udviklingsteamet. For at en Gadget kan optages på portalen, skal der udstedes et klientcertifikat til borger.dk så portalen kan benytte sig af servicen. Gadget'en registreres i samlingen med følgende parametre:

Start link: det link som alle brugere vil starte på når Gadget'en initialiseres.
Bredde: én af de 3 bredder samt en præference om hvor Gadget'en skal placeres på siden.
Navn: en titel som altid står i Gadget headeren, til hurtig identifikation: f.eks. 'skat.dk'

Kapitel 7

7. Test

7.1. Indledning

Test kapitlet sigter mod at teste løsningen mod henholdsvis kravspecifikationen og den programspecifikke funktionalitet.

Konverteringstesten, viser at programmet udfører handlinger efter hensigten.

Kravspecifikationstesten viser at programmet løser opgaven.

7.2. Manuelle tests

7.2.1. Konvertering

Succeskriteriet for denne række test, er at input HTML bliver konverteret af portalen og får den rette output HTML.

Input	Forventet output	Resultat
LINKS		
?	?	succes
?test=123	?test=123	succes
pong.asp?test=abc	pong.asp?test=abc	succes
pong.asp	pong.asp	succes
../	../	Succes
/../	/../	Succes
mappe/pong.asp	mappe/pong.asp	Succes
mappe/pong.asp	/mappe/pong.asp	Succes
www.link.com	www.link.com	Succes
http://www.link.asp	http://www.link.asp	Succes
http://www.link.asp?	http://www.link.asp?	Succes
http://www.link.asp?asd=123	http://www.link.asp?asd=123	Succes
FORMS		
<form></form>	<form action="default.aspx"><input name="LinkKey" value="[GUID]" type="hidden"></form>	Succes
<form> <input type="text"> </form>	<form action="default.aspx"><input name="LinkKey" value="[GUID]" type="hidden"> <input type="text"> </form>	Succes
<form method="post"> <input type="text"> </form>	<form method="post" action="default.aspx"> <input name="LinkKey" value="[GUID]" type="hidden"><input type="text"> </form>	Succes
<form method="post" action="link.asp"> <input type="text"> </form>	<form method="post" action="default.aspx"> <input name="LinkKey" value="[GUID]" type="hidden"><input type="text"> </form>	Succes

<form method="post" action="link.asp" accept-charset="utf-8"> <input type="text"> </form>	<form method="post" action="default.aspx" accept-charset="ISO-8859-1"> <input name="LinkKey" value="[GUID]" type="hidden"><input type="text"> </form>	Succes
SCRIPTS		
Scripts placeret forskellige steder på en side, endten inline eller med <script src=””>	Alle scripts-kilder findes sammensat i starten af Gadget'ens html, ingen andre steder i Gadget'en ses et script tag	succes
STYLES		
Styles placeret forskellige steder på en side, endten inline eller med <link href=””>	Styles tilføjet et namespace (Class) som er Gadget-specifik. Alle style-data fra style-tags findes sammensat i Gadget'ens html, ingen andre steder ses et style-tag	succes
BILLEDER		
IMG-tags med link til billede	Alle img tags har et absolut link	succes
CSS url('') links	Alle css url's har et absolut link	succes
Background=”” attributter	Alle background attributters værdi er et absolut link	succes

7.2.1. Test af krav

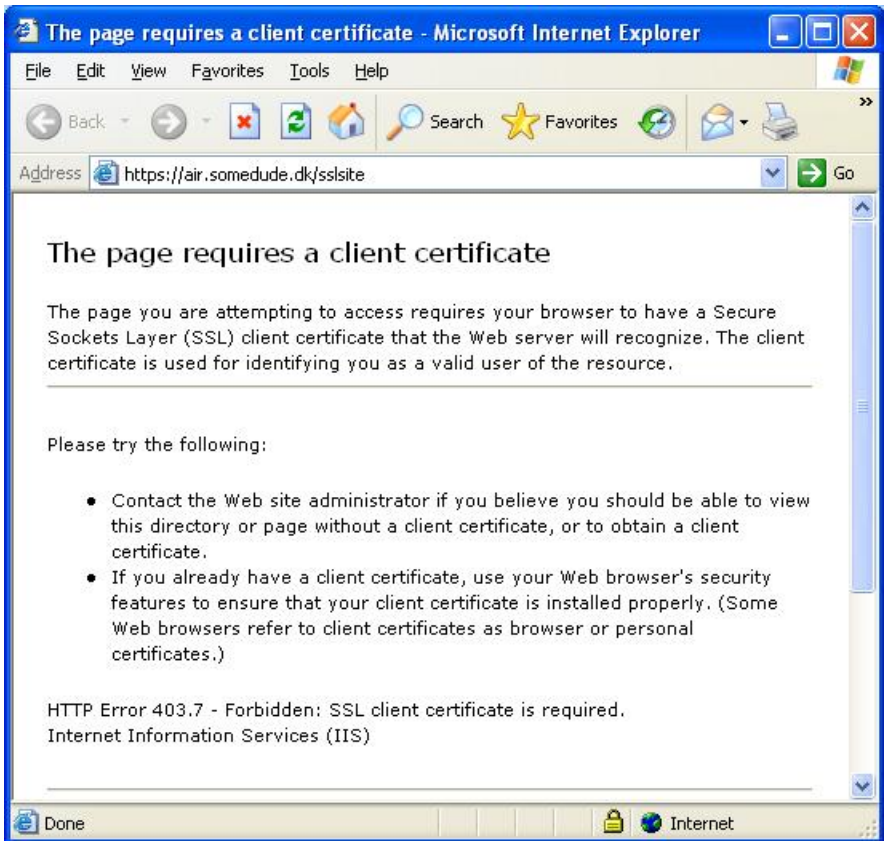
Kravene fra kravspecifikationen

Krav	Forventet resultat	Resultat
2.1.1 Visuel integration 1 – Gadgets præsenteres som en del af borger.dk	Gadgets er intergreret i designet i matchende stil	fejlet
2.1.1 Visuel integration 1 – Gadgets præsenteres som en del af borger.dk	Gadgets fremtræder som elementer i designet af borger.dk	succes
2.1.1. Visuel integration 2 – Gadgets er ikke loadet asynkront	Gadgets loader samtidig med borger.dk	succes
2.1.1 Visuel integration 2 – Gadgets er ikke iframes	Gadgets er indlejret i html'en fra borger.dk	succes
2.1.1 Visuel integration 3 – automatisk visning af borgerspecifikke Gadgets	Gadgets som brugeren er tilmeldt er vist fra starten	Fejlet – ikke implementeret
2.1.1 Visuel integration 4 – Brugeren logges ind via SSO	Brugeren er logget ind på Gadgets via SSO	Fejlet – ikke implementeret
2.1.1 Visuel integration 5 – Formularer kan postes i Gadgets	Formularer kan sendes i en Gadget, og resultatet vises i samme Gadget	succes
2.1.2 Valgfri arkitektur 1 – udbydere kan selv vælge arkitektur	Hjemmesider som opfylder grundreglerne for Gadget udvikling, kan vises korrekt.	succes
2.1.3 Tilgængelighed 1 & 2 – Gadgets tillader udvikling af WAI-opfyldte services	Gadgets virker uden brug af Scripts og iFrames, som er grundkravene i WAI	succes
2.1.4 Fortrolighed og Sikkerhed 1 – Gadgets yder høj sikkerhed	Der kan oprettes SSL forbindelse fra brugeren igennem portalen og hele vejen til en serviceside igennem Gadgets.	succes
2.1.4 Fortrolighed og Sikkerhed 2 – Portalen gemmer ikke fortrolige informationer	Data fra brugeren og fra Gadgets gemmes i session variable, men ikke andre steder.	succes

2.1.4 Fortrolighed og Sikkerhed 3 – Portal og Serviceudbydere har stærke relationer	Login på service kan kræve certifikat, kun givet til Portalen – hvilket holder andre ude.	succes
2.1.5 Performance 1- krav om flere brugere på samme tid	Tiden hvormed en Gadget loades er meget kort og flere brugere kan være online på samme tid.	succes
2.1.5 Performance 2 – krav om datasikkerhed på trods af spidsbelastning	Datavejene fra bruger til serviceudbyder er ikke afhængig af tidsspecifikke variable	succes

7.2.2. Test af certifikat

Prøver man at åbne en service sat op med certifikat direkte i browseren, uden at have klientcertifikatet, får man følgende fejl:



Figur 47- Siden kræver et klientcertifikat.

Hvis man ikke har et gyldigt certifikat, er vi derfor sikre på at en service ikke bliver misbrugt. Hvis vi tillader anonym adgang til servicen, og sætter indstillingen med certifikater til at acceptere certifikater, kan vi logge på uden at have et certifikat. På vores test service, vil vi komme frem til følgende side:

Figur 48 - Bliver henvist til en anden side, fordi der ikke blev fundet et certifikat.

Vi ser altså at servicen registrerer om der er brugt et certifikat, og er der ikke et gyldigt klient-certifikat til stede, vil brugeren blive redirected til en anden side, hvor der står at der ikke blev fundet et certifikat.

Prøver vi at logge på med et certifikat, mens det stadigvæk er muligt at logge på anonymt, får vi denne side:

Figur 49 - Der blev fundet et gyldigt klientcertifikat.

Er certifikatet gyldigt, bliver det som vist på Figur 49 at certifikatet er godkendt, og man vil blive budt velkommen, som den testbruger vi har oprettet, til test af certifikater.

Selvom at websitet på IIS serveren er sat op til at kræve certifikat, og ikke tillade anonym adgang, vil en request af siden, stadigvæk kunne blive afbrudt, og redirected til en anden side, så brugeren således vil blive oplyst om at certifikatet ikke er gyldigt.

For at se om vi vil blive præsenteret for en fejl, eller om en service stadigvæk vil blive loadet, hvis et servercertifikat ikke er gyldigt, prøver vi at lade valideringen af servercertifikatet fejle. Dette gør vi ved at indsætte

```
return false;
```

under valideringen af vores servercertifikat. Vi får derfor at vide at servercertifikatet ikke er gyldigt, og forventer at servicen ikke vil blive vist.

```
last request returned error :The underlying connection was closed: Could not establish trust relationship for the SSL/TLS secure channel.
```

Som forventet bliver servicen ikke loadet, og vi får i stedet for en fejlbesked om at der ikke kan oprettes en sikker SSL forbindelse til servicen.

7.3. Resume

Testene viser at 2 funktioner fejler, fordi de ikke er implementeret. Begge funktioner ligger uden for opgavens rammer, hvilket betyder at løsningen løser opgaven indenfor rammerne. Derudover har vi vist at man ikke kan få lov at hente en service der kører på en sikker forbindelse, hvis man ikke er i besiddelse af det rigtige certifikat.

Kapitel 8

8. Udvidelser

8.1. Bedre fejlhåndtering

For at brugeren ikke skal få en fejl om, at der ikke er et klientcertifikat til rådighed, eller en anden fejl, som brugeren ikke kan gøre noget ved, skal der laves et tjek på, hvad serviceudbyderen svarer portalen med. Får portalen ikke noget indhold retur, der kan bruges til en Gadget, skal portalen ikke udskrive fejlen der kommer, på portalen så brugeren kan læse den, men i stedet fortolke fejlen, og give brugeren en besked om at servicen midlertidig er ude af drift. Tolker man ikke fejlen, men i stedet for sender den direkte til brugeren, får brugeren indtryk af at der ikke er styr på behandlingen af data, og vælger måske ikke at bruge portalen en anden gang. Det er heller ikke brugbare oplysninger for brugeren, da han alligevel ikke vil kunne stille noget op ved problemet.

8.2. Skift af transport protokol

For at finde den mest optimale løsning på at loade en Gadget til borger.dk, vil det være hensigtsmæssigt at forsøge at lave et forsøg med sockets. Ved at have to eksempler, hvor det ene eksempel bruger HttpWebRequest som i vores eksempel, og et andet eksempel hvor man bruger socket, kan man stille dem op imod hinanden, og se hvilken der giver det bedste performance resultat. Da sikkerheden er den samme ved de to former for request, må metoden med den bedste performance test, altså den der loader hurtigst være den mest brugbare metode. Load tiden, vil blive forøget ydermere, når der kommer flere services til, og derfor må den hurtigste metode være mest oplagt. Ved sockets skal der dog implementeres tråde, hvilket allerede er en del af HttpWebRequest implementeringen i .NET.

8.3. Gadget kommunikation

I den Offentlige Erfaringsrapport [1] er der et eksempel med en vognmandstilladelse. Denne proces foregår mellem flere offentlige myndigheder, og består ifølge rapporten af flere end 7000 breve og e-mails om året. Ved at digitalisere denne proces, kunne man spare alle disse henvendelser, og spare en masse arbejde. I usecase 4 har vi lavet et eksempel hvor to serviceudbydere arbejder sammen. Det kræver dog en del arbejde fra portalens side, hvis to services skal snakke sammen.

For det første har services ikke adgang til hinandens metoder og data, og de kan derfor ikke sende data, som for eksempel en udfyldt form, til en anden service.

For det andet kan portalen ikke uden videre sende data til en anden serviceudbyder, på baggrund af udfyldt data i en service. For at portalen kan sende data mellem services, er den nødt

til at vide på forhånd om der er et samarbejde mellem to services, og hvornår den har lov til at sende data videre.

8.4. Personalisering

For at gøre ”Min Side” mere personlig, kan man lade brugeren vælge hvor services skal placeres på siden. Det kan være at en bruger foretrækker at servicen fra Told og Skat skal være øverst, fordi det er den der bliver brugt oftest. Derudover kunne man lade brugeren vælge hvilke services der skulle vises på ”Min Side”. Hvis der er en service som brugeren ved med sikkerhed ikke skal bruges, eller som først skal bruges flere år ud i fremtiden, hvis det f.eks. er en service der viser en pensions opsparing, kunne brugeren få lov til at fjerne disse services fra oversigten på ”Min Side”. For at lade brugeren sætte disse indstillinger, kræver det dog at man gemmer oplysninger om den enkelte bruger på portalen, så opsætningen kan huskes. Dette bryder dog mod glasplade princippet, der siger at oplysninger om brugeren ikke må gemmes på portalen. Alternativt kan indstillingerne gemmes i en cookie. Ulempen ved dette er at alle brugere ikke tillader cookies på deres pc, samt at indstillingerne således kun vil være gemt på lige netop den pc brugeren sidder ved. De ville således ikke blive husket alle steder, hvis bruger har flere computere i hjemmet.
Ved at lave en menu med hvilke services en bruger har adgang til, kan dette dog undgås. Brugeren vil dermed kun få vist en service af gangen, hvilket også vil øge loadtiden, i modsætning til vores test scenarie, hvor vi viser flere test services samtidig.

8.5. Udvidet sikkerhed

For at sikre portalen endnu mere, bør man se om servercertifikatet hos den enkelte serviceudbyder er gyldigt. Ved at gøre dette jævnligt, sikrer man sig at det ikke er blevet spærret. I vores projekt tester vi certifikatet på en række punkter, men vi validerer det ikke online, op imod certificate authority. Ved at tjekke den, kan man se om certifikatet er kommet på en spærreliste, og dermed ikke gyldigt mere. Hos cacert.org som vi har brugt til at udstede certifikater, har de en Online Certificate Status Protocol (OCSP) hvor man kan validerer certifikatet med det samme.
For et scenarie med en rigtig service udbyder, ville det dog være TDC der skulle udstede certifikatet, og derfor TDC man skulle tjekke gyldigheden af certifikatet hos. TDC har en certificate revocation list (CRL), der indeholder serienumrene på alle de certifikater, der er blevet spærret.

OCSP og CRL har ikke noget at gøre med udløbsdatoen på certifikatet, da et certifikat sagtens kan blive spærret i gyldighedsperioden.

8.6. AJAX

Som portalen og vores løsning er designet i dag, er interaktionen meget sløv og gammeldags. Hele siden hentes på ny, når man klikker på et link i en Gadget, hvor de fleste ville forvente at det kun er Gadget'en som opdateres. AJAX er en god og nem løsning på dette problem og kan implementeres sådan at browsere med script blokering stadig vil virke, samt at kravene om tilgængelighed overholdes.

AJAX er også nyttig når formularer skal valideres og kan i denne sammenhæng laves som et færdigt Framework, som serviceudbyderne kan benytte sig af.

Det største problem med at implementere AJAX optimalt er, at data skal sendes over portalen, hvilket vil føles langsomt.

Kapitel 9

9. Konklusion

9.1. Formål

Formålet med projektet var at vise hvordan man kunne tilføje services på borger.dk ved at lave sin egen standard. Det skulle kunne lade sig gøre at tilføje, en eller flere services til borger.dk, samtidig med at man kunne benytte sig af funktionaliteten på hver enkelt service. Standarden skulle definere aspekterne ved design, interaktive handlinger, sikkerhed og identifikation.

9.2. Resume

Der er lavet en analyse af de eksisterende metoder til service integration, samt en analyse af hvad der skal til, for at indlejre services på en portal. Vi har i analysen fortalt hvordan man kan identificere brugere og hvordan der kan laves et ”Single Sign On” system. Udover at vise hvilke eksisterende løsninger der findes, har vi beskrevet hvordan datatransporten mellem portalen og de tilføjede services kan bygges op.

Det er lykkedes at beskrive en standard for sikker transport over en SSL forbindelse, fra en serviceudbyder til en portal, hvor kilden bliver konverteret og fremvist til brugeren. Aspekterne ved at lave interaktive services, som kan integreres på en portal er gennemgået.

Vi har lavet en implementering af en service, der kører over en sikker forbindelse, samt eksempler på sikkerhedsløse services. Det er lykkedes at lave en komponent, som kan håndtere mange services pr bruger, robust nok til at der kan eksistere flere instanser af den samme service hos én bruger.

Vores tests viser at komponenten lever op til kravene og at der er aspekter som ikke har været mulige at implementere i dette projekt.

9.3. Evaluering

9.3.1. Problemer under projektet

Inden projektets start var det lidt usikkert, hvilke aspekter ved borger.dk vi skulle undersøge. Vores vejleder havde i første omgang foreslået, at vi burde kigge på en SSO løsning, men efter starten på projektet, fandt vi ud af, at en SSO løsning til borger.dk allerede var valgt. SSO løsningen var dog ikke implementeret, hvilket gjorde at vi måtte vælge et andet aspekt.

Brugen af certifikater skabte en del problemer. Til at starte med fungerede det hele fint, men ved skift til IIS havde vi ikke længere adgang til ”certificate store”. Efter en del søgning på nettet, fandt vi ud af at mange havde haft samme problem. Vi kunne konkludere at fejlen skyldtes at ASP.NET brugeren ikke havde tilladelse til at læse i certificate store, men der var ingen forklaring på hvordan der skulle gives adgang. Ved at bruge en komponent fra CAPI-COM, lykkedes det at få adgang til certificate store.

Et andet problem som tog en del tid at forstå, var håndtering af tegnsæt. At forstå hvordan tegnsæt bliver brugt og konverteret af ASP.NET var ikke nemt, hvilket førte til en del forbrug af tid.

9.3.2. Produktevaluering

Ved at vælge den simpleste måde at indlejre services på portalen, er det blevet utroligt nemt for serviceudbydere at skabe deres løsninger. Serviceudbyderen er fri for at skulle installere et program som skal vedligeholdes og kan i stedet koncentrere sig om vedligeholdelse af indholdet. Serviceudbyderen har også mulighed for at præsentere sin service andre steder, idet den vil fungere i en iFrame.

WSRP som er det eneste kendte alternativ til vores Proxy løsning kan sammenlignes således

1. WSRP Kræver installation af produkter hos alle serviceudbydere – dette er de frie for med vores komponent.
2. Microsoft er tilbageholdende med at understøtte WSRP fordi Microsofts Webparts, har højere prioritet.
3. WSRP 2.0 er på vej og vil kunne løse mange aspekter, men det er usikkert hvornår den er klar og i hvilket omfang.

Produktet er i den nuværende version, baseret på en Regex parser, som ikke er den mest ideelle løsning, en opgradering til DOM parsing, vil være optimalt.

Kapitel 10

10. Vejledning

For at kunne køre vores forslag, til hvordan man integrere services på portalen, kræver det lidt opsætning først.

Vi har vedlagt en cd med følgende indhold:

- Inetpub\wwwroot\GadgetPortal – kildekode
- Cert – certifikater
- Install – værktøj til at give adgang til private nøgler
- Udvikling af Gadget til Borger.doc – rapporten i Word format
- Udvikling af Gadget til Borger.pdf – rapporten som pdf

Inetpub mappen med kildekoden skal kopieres til roden af c-drevet. Derefter skal der oprettes en webside på IIS serveren, der henviser til GadgetPortal mappen, og som kører med ASP.Net version 2.0.50727.

Cert mappen indeholder certifikaterne. De skal installeres i local_machine\my. Dette kan gøres ved hjælp af Microsoft Management Console, der kan startes ved at trykke start og køre mmc.
Tryk her på filer – tilføj/fjern snap-in og tryk på tilføj.
Vælg at tilføje certifikater og vælg at styre Computerkonto certifikaterne. Ved at lukke vinduerne så man kommer tilbage til konsol programmet, kan man nu gå ind i det personlige certifikat store, hvor certifikatet skal installeres. Vælg at installere certifikatet her.
Vil man hente certifikaterne fra disken, skal Cert mappen kopieres til roden af c-drevet.
For at kunne bruge certifikaterne skal der gives adgang til de private nøgler. Dette gøres ved hjælp af winhttpcertcfg, som ligger i Install mappen. Det kan også hentes direkte fra Microsofts hjemmeside [5]
Se under ”Adgang til privat nøgle” under implementering af certifikat, for at se hvordan der gives adgang til den private nøgle.
Der er 3 forskellige certifikater til rådighed. ClientCertUdenPass og TestCert er begge certifikater der er gyldige, men serverCert er serverens certifikat. Serverens certifikat kan bruges som klient certifikat, hvis man vil teste at der ikke er adgang med et ikke gyldigt certifikat.

Til at vise at vores program kan indlæse andre sider, har vi lavet to test services. En der kører med certifikat, og en der kører uden. Servicen med certifikat findes på https://air.somedude.dk/sslsite mens den uden certifikat findes på linkhenvisd.

Begge services kan åbnes direkte fra en browser, man er dog nødt til at installere et certifikat, før man for adgang til sslsitet, ellers vil man blive afvist.
Vælger man at teste sslsitet direkte fra browseren, kræves det at man truster udstederen af serverens certifikat. Dette gøres ved at godkende deres root certifikat. Root certifikatet findes her: http://www.cacert.org/index.php?id=3.
Bruger man en anden browser end Internet Explorer kan det være nødvendigt at installere certifikatet igennem browseren.
Hvis man vil teste servicen uden certifikat, kan den indlæses via en browser, uden at skulle ændre noget. Sådan en service ville kunne bruges, hvis der kun skulle overføres ikke person-følsomme oplysninger.

Kapitel 11

11. Litteraturliste

1. PoC - http://oim.modernisering.dk/ProofOfConceptErfaringer#Composite
2. Certifikat - http://support.microsoft.com/kb/901183 - henvis fra adgang til privat nøgle
3. Capicom - http://msdn2.microsoft.com/en-us/library/ms867087.aspx
4. Capicom - http://www.microsoft.com/downloads/details.aspx?FamilyID=860EE43A-A843-462F-ABB5-FF88EA5896F6&displaylang=en
5. Winhttpcertcfg - http://www.microsoft.com/downloads/details.aspx?familyid=c42e27ac-3409-40e9-8667-c748e422833f&displaylang=en
6. IFrames - http://html-faq.dk/2009.asp
7. IFrames - http://praxisworks.net/archives/2005/12/21/45.htm
8. SAML - http://en.wikipedia.org/wiki/SAML
9. SAML - http://xml.coverpages.org/saml.html
10. Dk-SAML - http://borger.dk/j2eebdk/app_hoering_showdoc/showDocument.jsp?p_docid=861948
11. Unified Process - http://da.wikipedia.org/wiki/Unified_Process
12. Unified Process –
    Applying UML and Patterns
    An introduction to Object-Oriented Analysis and
    Design and Iterative Development
    Third Edition
    Craig Larman
13. Smoke test - http://en.wikipedia.org/wiki/Smoke_testing
14. Web Accessibility Initiative - http://www.w3.org/WAI/
15. JSON - http://en.wikipedia.org/wiki/JSON
16. SOAP - http://en.wikipedia.org/wiki/SOAP
17. WSRP – http://oim.modernisering.dk/filer/WSRP-analyseITST.pdf
18. Fællesofentlige integrationsmodel for Borgerportalen og Virksomhedsportalen (version 1.0) - http://oim.modernisering.dk/filer/TF_OIM_1.0.pdf