voresjazzklub.dk  
en studieopgave i webteknologier

Indholdsfortegnelse

[Voresjazzklub.dk 1](#_Toc480993657)

[Indledning 1](#_Toc480993658)

[Problemformulering 1](#_Toc480993659)

[Afgrænsning 1](#_Toc480993660)

[Teori 1](#_Toc480993661)

[Teknologivalg 2](#_Toc480993662)

[MVC som design pattern 2](#_Toc480993663)

[Et dynamisk website 3](#_Toc480993664)

[Betragtninger omkring sikkerhed 3](#_Toc480993665)

[Databasemodel 3](#_Toc480993666)

[Praktisk opgave 8](#_Toc480993667)

[I gang med dette konkrete setup 8](#_Toc480993668)

[Migrering af adgangskontrol fra SQLserver til MySQL 18](#_Toc480993669)

[Bygning af Voresjazzklub.dk – de første skridt 20](#_Toc480993670)

[Test/Validering 24](#_Toc480993671)

[Det videre forløb 24](#_Toc480993672)

[Konklusion 24](#_Toc480993673)

[Litterarurliste 25](#_Toc480993674)

[Appendix 26](#_Toc480993675)

[Lidt om, hvordan programkoden bliver til 26](#_Toc480993676)

[Noget hurtig PHP 27](#_Toc480993677)

[Noget hurtig ASP.NET 29](#_Toc480993678)

[Script til migrering af adgangskontrol 30](#_Toc480993679)

**A great many thanks to Google.  
Without Google, this wouldn’t have happened.**

**Forord**

Engang i en fjern fortid, da jeg var på sidste år i gymnasiet, stod der et fjernsyn og et skrivemaskinetastatur på fysikgangen. Det viste sig at være en adgang til noget edb; ikke noget for mig. Min daværende interesse for elektronik var primært ovre i den analoge afdeling. Det ændredes hurtigt, og blot to år senere havde jeg en hjemmedatamat – sådan hed det altså i gamle dage. Det var en CP/M-maskine, en NewBrain.

Operativsystemet var så begrænset i omfang, at det både kunne læses og forstås. Al kodning var i assembler; og maskinkode, når assembleren ikke kunne følge med længere. Det var sjovt at pille i operativsystemet. Alting var småt, nemt og simpelt.

At begynde fra scratch, hver gang, virkede naturligt. Henad vejen indså jeg fordelene ved genbrug. Genbrug kræver dog, at koden er velfungerende, gennemtest og modstandsdygtig overfor fejlagtig anvendelse.

Men Verden er ikke konstant. Specielt er IT-verdenen ikke konstant.

Jeg tror på, at ting skal testes, og, at det skal være nemt at teste.

Jeg tror på, at det er værd at bygge videre på, hvad andre har lavet.

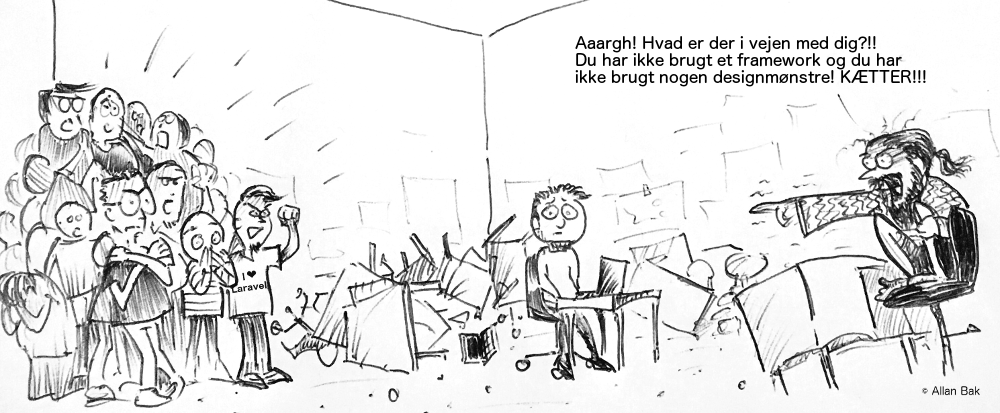
Jeg tro på, at it-systemer skal bygges, for at blive forandret.

Nye udviklingsværktøjer, gør det muligt at komme i gang på ingen tid. Det går faktisk hurtigere at fremstille en ”Hallo World”-webapplikation, end at udtale det.

Gode, stabile IDE’er, færdige komponenter og frameworks gør it-udvikling nemmere og hurtigere. Ulempen er, at der er så meget mere, der skal læres, før man rigtigt mestrer produkterne. Tillige skal der stadigt tilegnes ny viden inden for samme rammer, da produkterne udvikles hen over tid, og der kommer nye svar på eksisterende problemstillinger.

Baggrunden for dette projekt er netop, at der er kommet nye produkter og andre er forandrede, og at jeg ikke helt har fulgt med.

Her er en, der ikke har fulgt med tiden. Don’t be him!



Claus Tetens, Hillerød, maj 2017.

# Voresjazzklub.dk

## Indledning

Blandt en gruppe mennesker, der alle havde lyst til at ses til noget spisning og musik i byen, har der været et behov for at kunne foreslå arrangementer, og for at kunne melde sig til de enkelte begivenheder.

Det har affødt en ide om at konstruere et mindre website til netop det formål.

Projektet er rent faktisk påbegyndt i starten af 2010, men på grund af noget teknisk bøvl, for lidt tid og anvendelse af en alternativ løsning [1], blev projektet droppet efter kun et par dages arbejde.

Siden 2010 er der gået syv år og i IT-sammenhæng er det lang tid, bl.a. eksisterede HTML5 ikke [2], så de tanker, der blev gjort dengang er kasseret og skal up to date.

Projektet er i en størrelse, hvor der kan laves en demo inden for web-teknologier og databaser, men næppe et helt færdigt website.

## Problemformulering

Der skal fremstilles et website til administration af arrangementstilmeldinger bland personer, der er tilmeldt som bruger.

Der skal kunne oprettes arrangementer, og efter afholdelse, skal det være muligt at uploade billeder.

Brugerne skal selv kunne tilmelde sig og kunne administrere deres konti.

En administrator skal have udvidet adgang til brugerdata og skal bl.a. kunne nulstille passwords.

Brugeradgange og administratorbeføjelser skal opbevares i en database.

Der skal anvendes nyere teknologier i det omfang det er praktisk muligt.

## Afgrænsning

Sikkerhed er en væsentlig del af nutidigt web. Derfor ville det være på sin plads at gennemføre den praktiske implementering underanvendelse af sikret tilgang via HTTPS (SSL eller TLS), men da ikke er gratis [3] at få lagt SSL på sitet, vil sikkerhed kun berøres teoretisk indtil en eventuel go-live. Det vil ikke have nogen implikationer for koden på server-siden, da kryptering foregår på socket-niveau

Der er på forhånd valgt et hostingfirma, web10.dk, og en pakke (basic med windowsserver) [6]. Det giver nogle afgrænsninger af de muligheder, der er for at vælge programmeringssprog mv.

Da der er tale om en studieopgave, er processen og læring væsentligere end produktet. En afdækning af, hvordan et delproblem løses, vægtes højere end at få en lækker applikation.

## Teori

*udvikling af dynamiske websider*

*principper for eller design og administration af databaser*

*netværksapplikationer*

*kommunikationsprotokoller*

*netværkskomponenter (routere, firewalls, etc.)*

*metoder til test eller validering af ovennævnte*

### Teknologivalg

Som et forprojekt, skulle det undersøges, hvilke teknologier, der skulle anvendes under fremstilling af websitet. Forfatteren har allerede arbejdet med Java/JSP og Java/Wicket, så der ville ikke være meget sport i at anvende disse i forbindelse med en studieopgave.

Tilbage stod så PHP, ASP.NET og ASP.NET MVC som de mest oplagte kandidater.

PHP er relativt let tilgængeligt og med en pakke som xampp [15] får man et fuldt integreret og kørende miljø. Det er nemt at komme i gang med, og det tager kun få minutter at lave en dynamisk udgave af ”Hallo World”. Det er illustreret i appendix ”Noget hurtig PHP”

Et af webhostingsfirmaets begrænsninger er, at man kan enten vælge Linux/PHP eller Windows/ASP.NET. Valget var i forvejen faldet på Windows, så PHP er ude.

Forfatteren har et beskedent kendskab til aps.net fra tidligere; dog må det konstateres, at der er sket en udvikling de sidste omtrent syv år, hvor asp.net sidst blev anvendt.

MVC som design-pattern er kendt, men asp.net mvc er nyt. Da en væsentlig del af projektet er læring, vil det være naturligt, at kaste sig over dette framework. Første version kom frem for otte, ni år siden, og har undergået en del forandring undervejs.

Google er en kammarat, når det drejer sig om it-udvikling. Der er altid svar på problemstillinger, men i forbindelse med asp.net mvc har der været mange fejlskud, fordi produktet har undergået så store forandringer. Der har flere gange været henvisninger til metodikker, der ganske simpelt ikke anvendes, og kan anvendes, længere.

Betydningen af denne udvikling er, at der har været bruge for mange små eksperimenter og undersøgelse af, hvad koden rent faktisk gør, for at kunne fremstille en brugbar kode.

Undervejs i projektet har det været nødvendigt at skifte maskine, og da det oprindelige projekt, på grund af de mange eksperimenter, var blevet noget saneringsmodent, var det oplagt at begynde forfra. I rapporten er der både henvisninger til det gamle projekt – VoresJazzklubSomMvc – og det ny, med navnet Voresjazzklub.

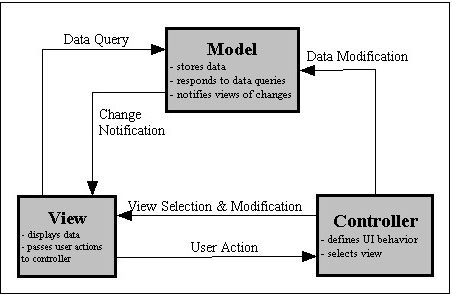
### MVC som design pattern

Design patterns er mønstre, der opdages i gode løsninger af forskellige problemstillinger. Hvis man i flere it-systemer har fundet frem til samme mønster, og det fungerer godt, så kunne der være en anledning til, at det næste projekt, med samme type opgave, også skal bygges efter det fundne mønster.

Et af de udbredte mønstre, er MVC. Model – View – Control.

Model holder de data, der skal arbejdes med. View er til brugerinteraktion, og Control er forretningslogikken. MVC er ikke kun anvendeligt i forbindelse med personer, men kan også fint bruges ved kommunikation mellem maskiner. I stedet for at hænge en webbrowser på view-delen, kunne man udstille dataudvekslingen som webservice (stikord: SOAP/WSDL, REST).

Herunder er det skitseret, hvordan sammenhængen mellem de tre dele af MVC kan opfattes.

 [18]

I ASP.Net MVC opnås sammenhængen ved at navnet på view’et og navnet på metoden knyttes sammen.

### Et dynamisk website

I sagens natur kan man i praksis ikke klare opgaven med statiske web-sider alene. I så fald skulle enhver ændring af data medføre en manuel ændring på en eller flere sider.

Det påtænkte website vil have en fastlagt struktur, men indholdet vil variere ud fra da lagrede data; herunder vil visse data og funktioner kun være tilgængelige for personer, der er registreret som administrator.

Brugeren, der har oprettet data, og administrator vil have mulighed for at ændre data.

### Betragtninger omkring sikkerhed

### Databasemodel

**Attributter Anvendelse**

brugerId brugerens eget valg af identifikation. Skal være unik.

createTs tidspunkt for, hvornår rækken er oprettet eller ændret

password brugerens password i hashed udgave

salt det salt, der skal gøre hash'en forskellig fra andre hashes af samme password

email email-adresse

prim markering af den foretrukne email-adresser

erAdmin om brugeren har administrative rettigheder

lastLogon hvornår har brugeren sidste logget sig på systemet

logonCnt hvor mange gange har brugeren været på

arrangementBeskrivelse en kort forklaring på, hvad arrangement går ud på

arrangementWeb link til web-siden, hvis en sådan findes

arrangementDt hvilken dato afholdes arrangementet

arrangementTid hvad tid på dagen indledes arrangementet

erTilmeldt om bruger er tilmeldt arrangement

spiseSted hvis der er spisning inkluderet, så navn og adresse på spisested

spiseTid tidspunkt for spisning under antagelse af, at det er samme dato som arrangementet

spiser om bruger deltager i spisning

billedAdresse path/filename.extention for uploadede billeder fra arrangementet

kommentar en optionel tekst om billedet

Password og salt gemmes i et antal versioner for at brugeren ikke skal kunne genanvende et password før der er gået et antal generationer.

Da email-adresser er universelt unikke, vil det være en oplagt mulighed at lade adressen indgå som en del af nøglen, men det er valgt, at email-adressen skal være optionel, og det skal være muligt at ændre den.

I nogle RDBMS'er eksisterer en auto-increment værdi, så hver gang der indsættes en række tælles op. Microsoft SQLserver [8] har en værdi, der tager udgangspunkt i timestamp, men er garanteret unik. Disse er naturlige valg til identifikation af en række i en tabel.

Hvis ikke der er en datatype med automatisk optælling, kan man konstruere sig frem til en sådan, under forudsætning af, at operationen kan køre atomisk. Altså at der ikke er flere, der kan indsætte eller opdatere samtidigt på samme tabel. Konstruktionen vil se ud som følgende:

Der er en kolonne med navn serialNumber, der indeholder et stigende rækkenummer for tabellen.

Insert into tabelNavn (serialNumber, ....) values((select max(serialNumber)+1, ...);

I en miniature-applikation som nærværende vil det ikke være noget problem at have en lås på hele tabellen, så kun en af gangen kan opdatere i den. I lidt mere anvendte applikationer, vil det normalt ikke være acceptabelt, at låse hele tabellen. Her vil man skulle nøjes med en rækkelås, eller lås den page, der er underopdatering, og så vil ovenstående SQL ikke kunnet garantere en unik værdi.

MySQL har timestampværdier med et-sekunds opløsning som standard. Der er så alligevel mulighed for at oprette op til seks cifre efter som en brøkdel af sekunder; altså en opløsning i mikrosekunder [9].

Andre RDBMS'er anvender sekunder, millisekunder eller mikrosekunder. Nogle gange er også 100 nanosekunder set [10]. I denne webapplikation, er alle indsætninger og opdateringer på tabellerne initieret manuelt. Det vil derfor ikke være forventeligt, at en bruger foretager to ændringer inden for et sekund. BrugerId er unikt og oprettelsestidspunktet (createTs) vil være unikt pr. bruger, når det nu antages, at brugeren ikke kan foretage sig flere handlinger inden timestamp er ændret. Det vil derfor være en mulighed at anvende brugerId og createTs som sammensat nøgle for alle data.

Hvis man forsøger at indsætte to rækker i en tabel, med samme nøgle, opstår der en constraint violation; man kan ikke anvende en given nøgle mere end en gang. Det skal så i givet fald en fejltekst i webbrowseren, så brugeren kan se, at der skal forsøges en gang mere. En mere brugervenlig udgave vil selvfølgeligt være at applikationen i første omgang selv håndterer situationen og f.eks. læser current timestamp nogle gange, og, når timestamp har ændret værdi, indsætter/opdaterer en gang mere.

Attributterne har hver netop en type. Der er ikke nogle af felterne, der skal opdeles, så de er atomiske. Det samme som, at der ikke er nogle repeterende felter i ovenstående, og ved anvendelse af brugerId og createTs som sammensat nøgle, vil alle kolonnerne i en tabel være på 1NF. Det ses dog hurtigt, at der ikke er fuld funktionel afhængighed for alle øvrige kolonner. F.eks. er arrangementTid en attribut, der fortæller om selve arrangementet, men ikke fortæller noget om personen, der har oprettet arrangementet eller har tilmeld sig. Dermed er der ikke en funktionelt afhængighed til nøglen, hvilket er et krav på 2NF.

Generelt kan det siges, at 1NF kræver atomiske værdier og ikke-repeterende grupper. Desuden skal der være en eller flere attributter, der tilsammen entydigt kan identificere en tuple.

2NF kræver, at alle attributter, der ikke indgår i nøglen, skal være fuld funktionelt afhængige af nøglen.

3NF siger, at der ikke må være nogen transitive afhængigheder.

4NF kræver, at der er max en flerværdiet afhængighed i hver relation.

Hvis der er en relation med bilmodeller, motorstørrelser, farver og priser, og vi antager, at der udelukkende er metalfarver. R(model, motor, farve, pris). Det viser sig, at pris ikke varierer med farven, men kun model og motor. For at komme på 2NF skal R splittes i R1(model, motor, farve) og R2(model, motor, pris).

3NF kan illustreres med R(navn, adresse, postnummer, by), hvor by er transitivt afhængig af postnummer. Derfor splittes R til R1(navn, adresse, postnummer) og R2(postnummer, by)

Til 4NF kan vi se på bil-relationerne igen. Der er flere modeller, flere motorer og flere farver. Motor og farve har ingen indbyrdes afhængighed, så derfor bliver R1(model, motor, farve) opdelt i R1a(model, motor) og R1b(model, farve).

Til praktisk anvendelse, er 3NF normalt helt fin; dog finder 4NF også nogle gange god anvendelse, specielt hvis der er tale om større datamængder. Yderligere normalisering er vist mest en teoretisk øvelse.

Ovenstående diskussion udmønter sig i at der skal laves fem tabeller, og at der skal tilføjes et entydigt arrangementId, som vil blive automatisk optalt nummer eller, hvis det ikke findes i pågældende RDBMS’s, current timestamp.

|  |  |  |
| --- | --- | --- |
| **Logon** |  |  |
| brugerId | lastLogon | logonCnt |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **User** |  |  |  |  |  |  |
| brugerId | createTs | password | salt | email | prim | erAdmin |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Arrangement** |  |  |  |  |  |
| arrangementId | arrangementBeskrivelse | createTs | arrangementWeb | arrangementDt | arrangementTid |

|  |  |
| --- | --- |
|  |  |
| spiseSted | spiseTid |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tilmeldinger** |  |  |  |  |
| arrangementId | brugerId | createTs | erTilmeldt | spiser |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Billeder** |  |  |  |  |
| billedAdresse | arrangementId | brugerId | createTs | kommentar |

Ovenstående konstruktion kan også give nogle ganske ”krøllede” SQL’er. Hvis man f.eks. skal finde ud af, hvem der deltager i spisning til et givet arrangement, skal man finde den nyeste række for et givet arrangement pr. bruger. Der er mange dialekter af SQL. En af dem er DB2 for zOS og her ville forespørgslen kunnet formuleres som følgende:

select brugerId

from tilmeldinger

where (arrangementId, brugerId, createTs) in (

select arrangementId, brugerId, max(createTs)

from tilmeldinger

where arrangementId = :hvadErDetForEtArrangementDerSpørgesPå

group by brugerId

)

and spiser = ’True’

Det er dog ikke alle SQL dialekter, der tillader sådan en konstruktion med flere attributter i where-in-clause. Som en omgåelse kunne man konkatenere arrangementId, brugerId og createTs til en tekststreng både i where-clausen og i subselecten. Hvis det viser sig, at der ikke er en enkelt konstruktion, der kan løse opgaven, kan man oprette en temporær tabel og anvende den i et select-statement.

create table sel (arrangementId char(…), brugerId char(…), createTs Timestamp);

insert into sel

select arrangementId, brugerId, max(createTs)

from tilmeldinger

where arrangementId = :hvadErDetForEtArrangementDerSpørgesPå

group by brugerId

;

select brugerId

from tilmelding t, sel s

where t.arrangementId = s.arrangementId

and t. brugerId = s.brugerId

and t.createTs = s.createTs

and t.spiser = ’True’

;

drop table sel;

Det bliver til en længere historie; dog har flere SQL varianter andre muligheder, der gør forespørgslen mindre klodset. Den dyreste mulighed er at tage resultatet af subselecten ind i et array i programmet, og så derefter slå op i tabellen tilmelding med hver enkelt række i arrayet. Der bliver meget transport og initiering af forespørgsel i databaseserveren.

Man kan argumentere for en yderligere normalisering, da man kan risikere redundans. F.eks. er User-tabellen ikke i 4NF. Det medfører, at en ændring i f.eks. email-adressen giver en ekstra række i tabellen, men med de samme data som i den nyeste række, bortset altså fra email-adressen.

Der eksisterer utallige beskrivelser af normalisering, men nogle af dem er [11], [12] pp387 og [14].

Opsplitningen vil give tre tabeller, hvor password og salt hører sammen i den ene tabel, email-adressen hhv. erAdmin kommer i to andre tabeller. Hvis ikke der var interesse for historikken på Admin-tabellen, så ville createTs være overflødigt. Dermed ville der være brugerId som nøgle. Da erAdmin er en boolsk værdi, ville det ikke være nødvendigt at have den som selvstændig attribut, og derfor ville Admin-tabellen kun bestå af brugerId.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **UserPassword** | |  |  |  | **Email** |  |  |  | **Admin** |  |  |
| brugerId | createTs | password | salt |  | brugerId | createTs | email |  | brugerId | createTs | erAdmin |

Af praktiske grunde accepteres redundans, og normalisering ud over 3NF (og evt. Boyce Codd) vil ikke blive gennemført.

Auto-increment konstruktionen blev fundet på mysql.com [13]. Den varierer meget mellem de forskellige database systemer. Samme sted blev også fundet en boolean datatype, men den gav syntaxfejl ved import i MySQL-Workbench; derfor blev datatypen ændret til char(1), der så f.eks. kan antage værdierne ’T’ og ’F’. Det vil blive styret i programmet, der skal læse og skrive på tabellerne, ved hjælp af to konstanter. I C# bliver det f.eks. til:

public const char booleanTrue=’T’;

public const char booleanFalse=’F’;

Herunder er så en MySQL flavoured DDL for oprettelse af en database med de fem tabeller, og et EER-diagram importeret i MySQL-Workbench.

create database VoresJazzklub;

use VoresJazzklub;

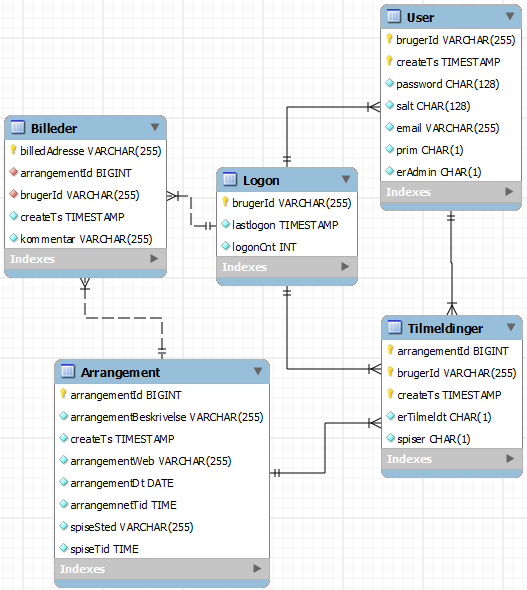
create table Logon(brugerId varchar(255) not null, lastlogon timestamp not null, logonCnt integer not null, primary key (brugerId));

create table User(brugerId varchar(255) not null, createTs timestamp(6) not null, password char(128) not null, salt char(128) not null, email varchar(255) not null, prim char(1) not null, erAdmin char(1) not null, constraint pk\_user primary key (brugerId, createTs));

create table Arrangement(arrangementId BIGINT UNSIGNED NOT NULL AUTO\_INCREMENT UNIQUE, arrangementBeskrivelse varchar(255) not null, createTs timestamp(6) not null, arrangementWeb varchar(255) not null, arrangementDt date not null, arrangemnetTid time not null, spiseSted varchar(255) not null, spiseTid time not null, primary key(arrangementId));

create table Tilmeldinger(arrangementId BIGINT not null, brugerId varchar(255) not null, createTs timestamp(6) not null, erTilmeldt char(1) not null, spiser char(1) not null, constraint pk\_tilmeldinger primary key(arrangementId, brugerId, createTs));

create table Billeder(billedAdresse varchar(255) not null, arrangementId BIGINT not null, brugerId varchar(255) not null, createTs timestamp(6) not null, kommentar varchar(255) not null, primary key(billedAdresse));



## Praktisk opgave

*website baseret på dynamiske websider*

*databaser som kan tilgås eller administreres over netværket*

*netværksapplikationer (server og/eller klient eller peers)*

*kommunikationsprotokoller*

*netværkskomponenter (routere, firewalls, etc.)*

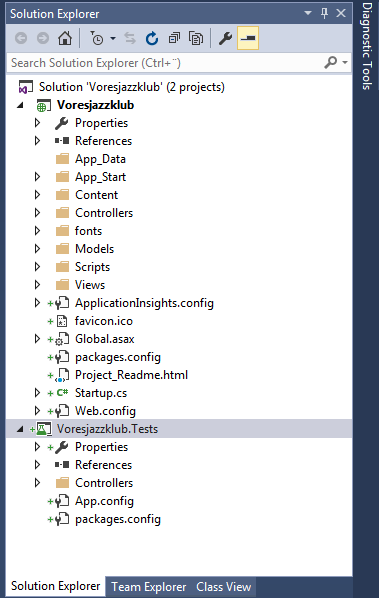
*værktøjer til test eller validering af ovennævnte*

Projektet er offentligt tilgængeligt på github [7]

### I gang med dette konkrete setup

En del af opgaven er at undersøge, hvordan ASP.NET MVC fungerer, og holde det op mod andre måder at løse samme slags web-kommunikation på. Derfor bliver der oprettet nogle små projekter, der på ingen måde vil kunne anvendes til noget praktisk, men kun er til for at afsløre, hvordan centrale problemstillinger bliver adresseret.

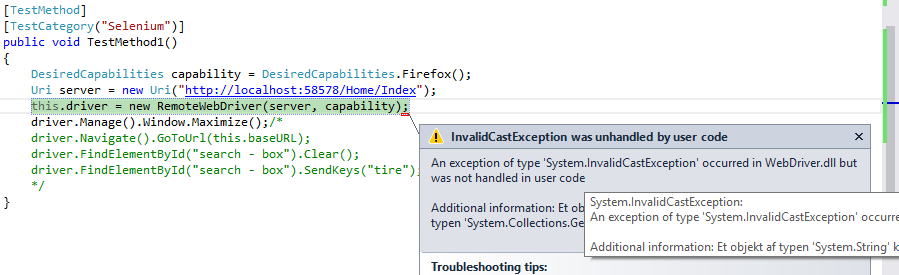
Når man opretter et projekt i Visual Studio og vælger ASP.NET MVC med tilhørende testprojekt, får man en kørende applikation og tre fungerende test i en struktur som vist her; dog er mappen Models i testprojektet tilføjet efterfølgende.



Der er små 600 filer og over 800 mapper. Om det er en fordel eller ulempe, at få så meget, komme nok an på, hvor godt man kender IDE’et og den generelle tankegang i MVC-modellen.

I første omgang kan man nøjes med at koncentrere sig om mapperne Controllers, Models og Views i projektet, og de tilsvarende mapper for controllers og views i testprojektet. Der er ikke umiddelbart nogen test af webgrænsefladen, men i betalingsudgaven af Visual Studio er der mulighed for at få startet IE og der igennem optage nogle handlinger mod applikationen. Handlingerne kan så afspilles igen, uden anvendelse af en browser. Det er så kommunikationen mellem browseren og webserveren, der undersøges. Det er også muligt at tilføje web-tests, ved at skrive dem selv efter en optagelse [20]. Alternativt kan Selenium anvendes, som er et produkt til test af web-sider [21]. Det kræver så mere installation og applikationen, der skal testes, skal køre på en server uden for IDE’et

Det skulle selvfølgeligt prøves, men der kom flere udfordringer ud af det. Google kunne ikke umiddelbart komme med nogen brugbar forklaring, så tilbage er kun at grave sig ned i problemet, hvilket formentlig er ganske tidskrævende.



Fejlteksten er ikke særligt sigende, og det er en fejl, der opstår i en komponent, der ikke er beskrevet detaljeret nok, til at fejlen umiddelbart kan afsløres.

For komme i gang, oprettes først en controller-klasse med en public metode. Som udgangspunkt rammer man metoden ved at anvende en URL på følgende form: <domænenavn>/<controller>/<metode>. Det kan man konfigurere sig ud af, hvis der ønskes en anden logisk struktur set fra brugerens side.

Controlleren kan for så vidt godt anvendes alene, hvis metoden f.eks. returnerer en tekststreng eventuelt med et stykke html, men ellers kan browseren godt vise ren tekst uden markups. Det bliver dog nogle noget fattige websider, som kan laves på den måde, med mindre man koder sig frem til en webside med alt indhold; men det er absolut ikke at anbefale, da det er svært at overskue og hurtigt kan blive et meget omfattende stykke arbejde at vedligeholde koden.

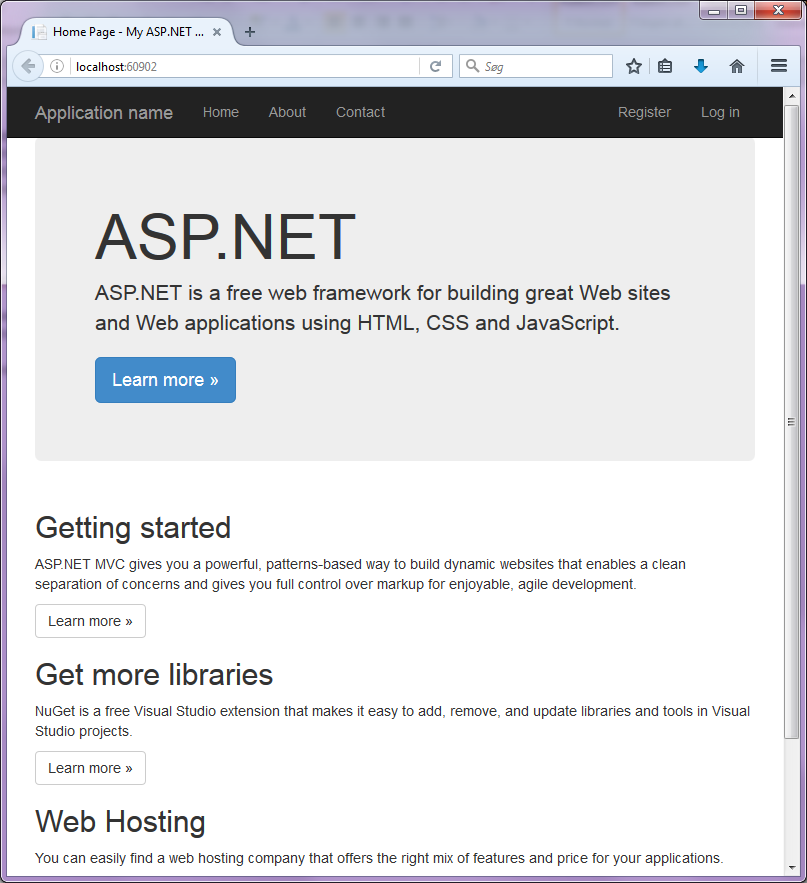
Dernæst laves et view. Da controlleren blev oprettet, blev der dannet to mapper. En i controller-mappen og en i view-mappen. På den måde bliver controller og view knyttet sammen. De enkelte metoder, der skal returnere et view, skal have et view med samme navn som metoden. Én metode returnerer ét view.

Der er et master-view, som man kan lade de øvrige views arve fra. På den måde kan man opnå ensartethed på tværs af siderne i applikationen.

Den tredje komponent er modellen. Her holdes de data, der skal vises på glaspladen subsidiært persisteres i databasen. Ved oprettelse af en controller blev der samtidigt oprettet en mappe til de tilhørende views. Det sker så ikke i relation til modellerne. Der er ikke den samme en-til-en relation, da man sagtens kan forestille sig, at en given model anvendes i flere sammenhænge. F.eks. kunne der være en side, som kun viser data, og en anden side, hvor data kan oprettes, vises, ændres og slettes. Den første side kan alle tilgå, og den anden side kan kun privilegerede få fat på. Det er de samme data, der skal anvendes på de to sider, og dermed samme model.

Modellen vil have adgang til forskellige metoder til læsning og skrivning på en tabel og i filsystemet. Her skal man så være opmærksom på konfigurationsfilerne. Til projektet er det Web.config og til test-projektet er det App.config. Her lægges connection strings til databaserne. Alt andet, der skiller test fra produktion, bør på lignende måde placeres i konfigurationsfiler.

En sidste ting, der kan være relevant på den funktionelle side af web-serveren, er RouteConfig.cs. Her kan man bestemme, hvor i applikationen man kommer hen ved at angive en sti efter domænenavnet i browserens URL input box.



web.config +=

<add name="MySQLConnection" connectionString="Server=localhost; user id=root; password=; persist security info=True; database=voresjazzklub" providerName="System.Data.SqlClient" />

<httpErrors errorMode="Custom" existingResponse="Replace" >

<remove statusCode="403" />

<error statusCode="403" responseMode="ExecuteURL" path="/Error/Forbidden" />

<remove statusCode="404" />

<error statusCode="404" responseMode="ExecuteURL" path="/Error/NotFound" /> <!-- viser kun den genererede side - overrides customErrors i system.web -->

</httpErrors>

DEBUG er en variable, som VS sætter ud fra om den arbejder i Debug eller Release modus. Den kan så anvendes af C#’s precompiler til at inkludere eller ekskludere kode.

#if DEBUG

ViewBag.isDebugging = "Hey! We're debugging!!";

#endif

I view koden undersøges, om der er en værdi med navn isDebugging og hvis det er tilfældet, bliver den vist på web-siden. Teksten bliver stylet ud fra to classes. Der anvendes to styles sheets, det ene indeholder styling af alle almindelige elementer og det andet bruges til ekstra styling og/eller til at override eksisterende styles. Her anvendes text-danger, som er med i projektet fra starten. Den giver den føde farve, og large-text er ny, da der ikke var andre styles, der virkede naturlige at anvende. Det er nemt at lave sådan en klasse, men man skal jo også have øje for, at der ikke er flere klasser, end at det er til at vedligeholde.

.large-text {

font-size: 72px;

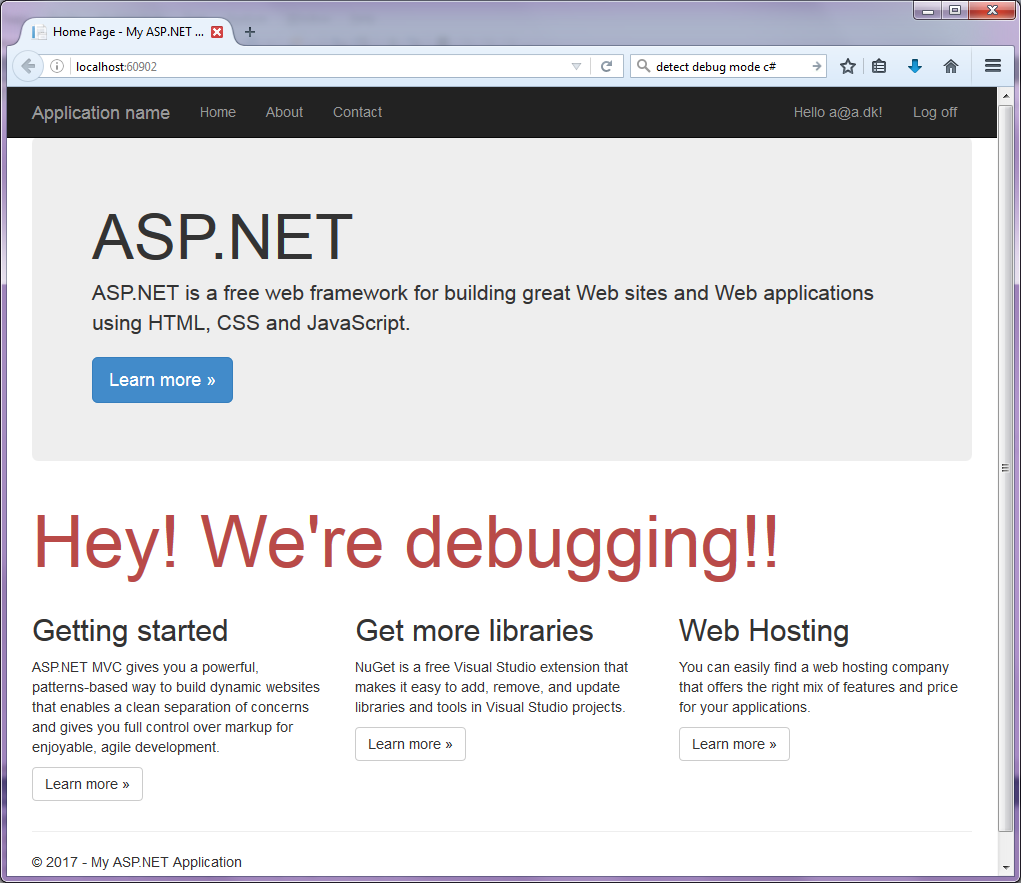
}

@{ if(ViewBag.isDebugging != null) {

<w class="text-danger large-text">@ViewBag.isDebugging</w>

}

}



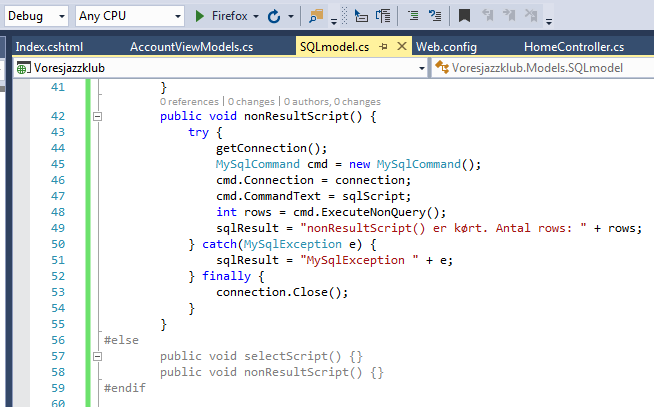
SQLmodel

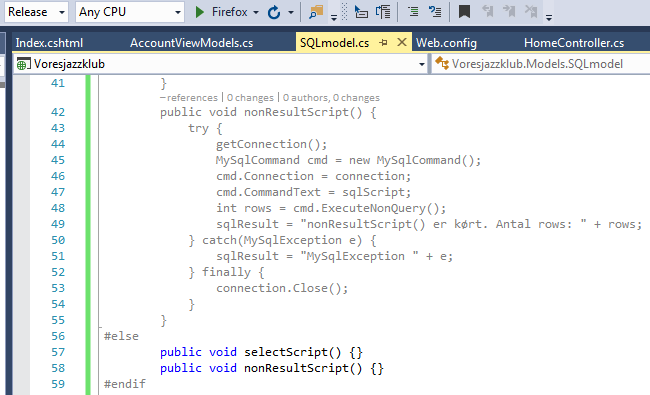
<https://dev.mysql.com/downloads/connector/net/5.2.html>

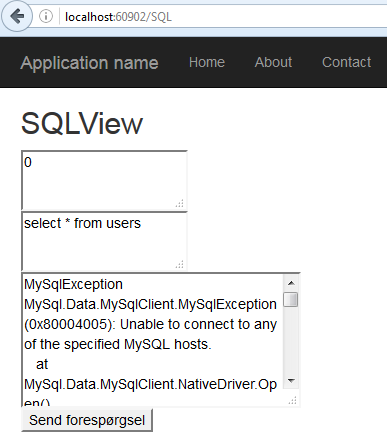
Add Reference i Solution Explorer

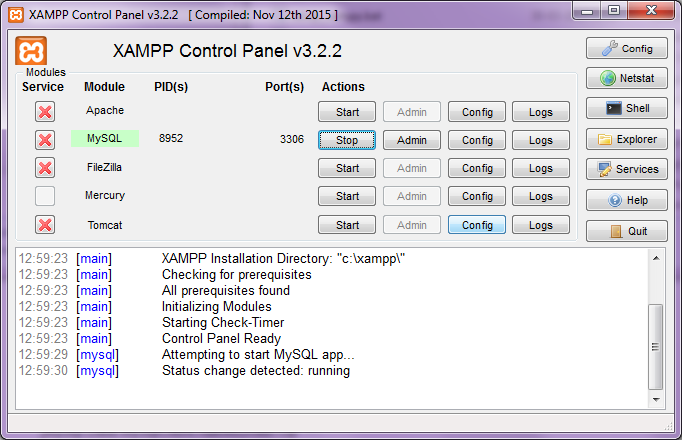
Find C:\Program Files (x86)\MySQL\Connector.NET 6.9\Assemblies\v4.5\MySql.Data.dll

Brug using MySql.Data.MySqlClient; i CS-koden

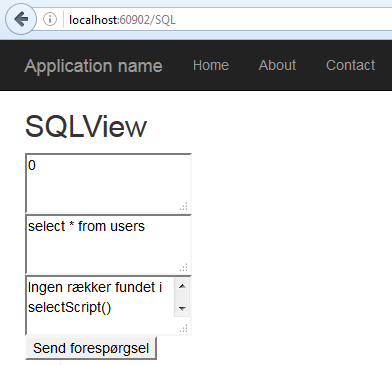




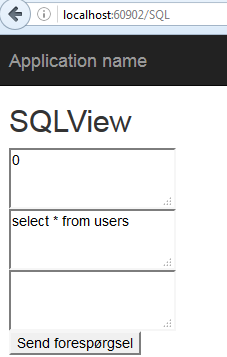




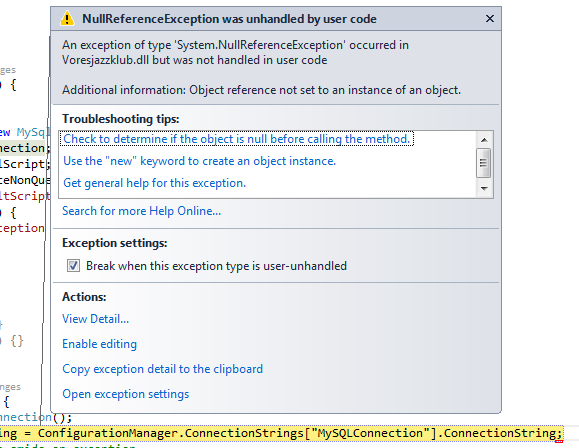
Kørsel af forespørgsel i Debug Configuration



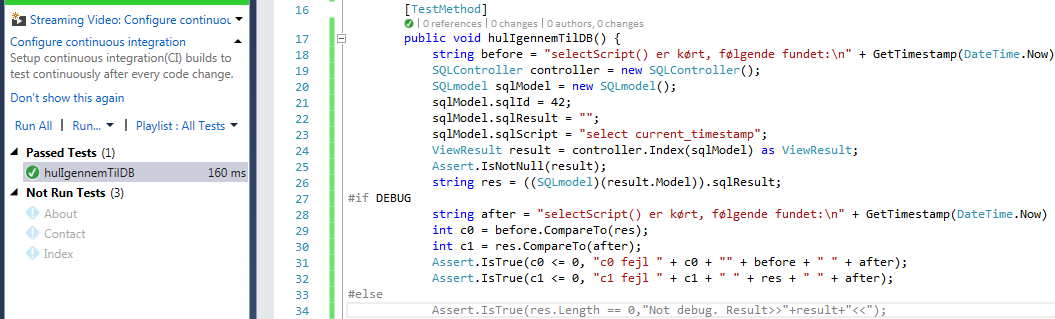
Kørsel af forespørgsel i Release Configuration



Testprojekt mod database



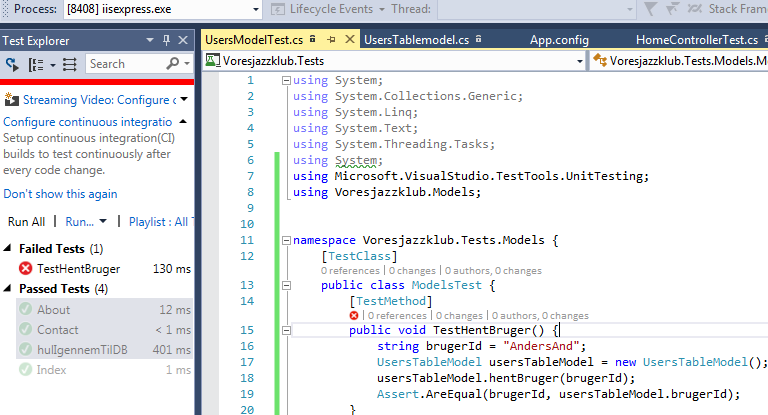
I Testprojektets App.config indsættes connection strings fra hovedprojektets Web.config

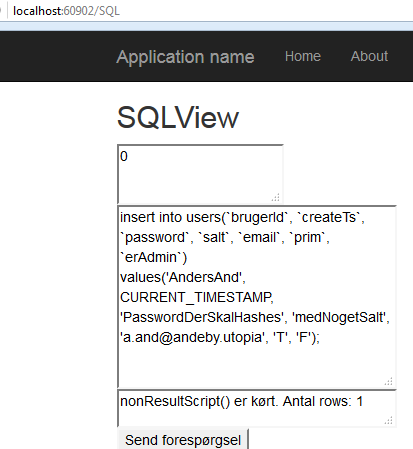


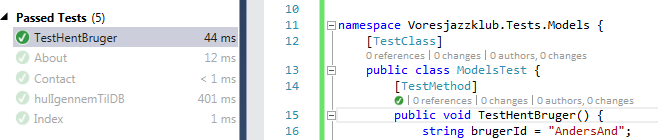
Er afprøvet i både Release og Debug modus. Fra tid til anden skal projektet Cleanes og Rebuildes, når der bliver skiftet mellem Release og Debug, da der er cachet information, der får testprojektet til at køre mod en forkert version.

En lidt bagvendt rækkefølge taget i betragtning af, at det er et TDD-projekt. Forsvaret for denne måde at gribe det an på, er, at det dels er en noget utraditionel feature, der skulle indbygges og dels, at det er i projekt-setup-fasen, hvor forskellige komponenter skal spille sammen. Når først der er hul hele vejen igennem, bliver det mere relevant at køre TDD på den rigtige måde.

Nu er tiden så inden til at fremstille den første del, der skal anvende i det færdige produkt. Det bliver model og controller til håndtering af brugere







### Migrering af adgangskontrol fra SQLserver til MySQL

Med det projekt, der bliver genereret af Visual Studio, kommer også et sikkerhedsframework. Da det foregår i en Microsoftverden, er det naturligt, at der anvendes Microsoftprodukter. Således bruges SQLserver, men da det erMySQL, som skal holde alle data i Voresjazzklub, skal der foretages nogle ændringer i koden.

Der er nogle klasser i Microsoft.AspNet.Identity.EntityFramework, der tilsammen håndterer brugeradgangen. Forventningen var, at der skulle skiftes ud i nogle metoder og andre skulle overrides, og at connection string skulle ændres fra SQLserver til MySQL.

Efter at have boret i problematikken et par timer, måtte det konstateres, at det nok kunne lade sig gøre, men at det samtidigt ville være en større operation.

Hvis projektopgaven var fokuseret på sikkerhed, ville det ganske givet være en interessant øvelse at ændre i frameworket. Her anvendes så i stedet dele fra en opskrift fundet på nettet. Det, der ikke anvendes, er hosting i Azure. [16] Der skal hentes et projekt [17] bestående af ti klasser. Det skal bygges og i Voresjazzklub skal der så være en reference til det. Derudover et par mindre justeringer i Voresjazzklub.

Download AspNet.Identity.MySQL, og build AspNet.Identity.MySQL.sln

Højre klik Voresjazzklub | Add | Reference

Browse til AspNet.Identity.MySQL..dll

Der står ganske vist noget helt andet i opskriften, men den er datostemplet 22/5/2015 – og det er lang tid i den verden.

Der skal ændres i tre filer. Microsoft.AspNet.Identity.EntityFramework skal skiftes ud med AspNet.Identity.MySQL + et par småjusteringer i to metoder. I web.config skal DefaultConnection flyttes til at pege på MySQL

**IdentityModels.cs:**

//using Microsoft.AspNet.Identity.EntityFramework;

using AspNet.Identity.MySQL;

public class ApplicationDbContext : MySQLDatabase {

public ApplicationDbContext(string connectionName)

: base(connectionName) {

}

public static ApplicationDbContext Create() {

return new ApplicationDbContext("DefaultConnection");

}

}

**IdentityConfig.cs:**

//using Microsoft.AspNet.Identity.EntityFramework;

using AspNet.Identity.MySQL;

var manager = new ApplicationUserManager(

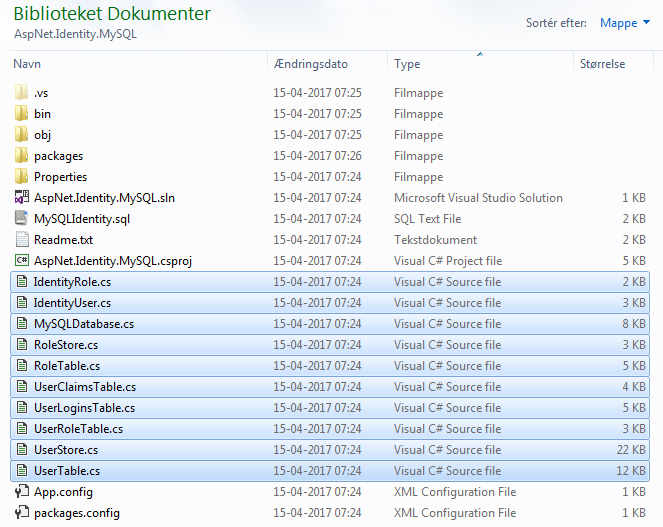
new UserStore<ApplicationUser>(

context.Get<ApplicationDbContext>() as MySQLDatabase));

**web.config:**

Connection string change to MySQL

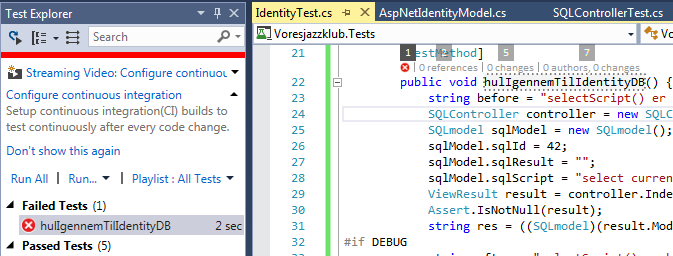
Disse klasser kommer der ud af at hente AspNet.Identity.MySQL:

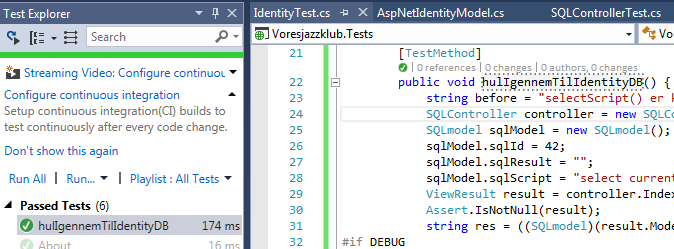


En sidste lille ting, der skal gøres før Voresjazzklub igen kan komme op at køre, er, at der skal oprettes fem tabeller i MySQL (se Appendix: ”Script til migrering af adgangskontrol”). Da tabellerne allerede ligger separeret fra de øvrige tabeller, er det valgt, at fortsætte sådan.

De tabeller, der er oprettet specifikt til Voresjazzklub ligger i databasen Voresjazzklub, og tabeller til adgangskontrol ligger i databasen MySQLIdentity. Når tiden er moden til en ændring af login mv. så det ligner de oprindelige tanker, er det nemmere at skille tingene ad, fordi der skal fjerne en database og en reference til et andet projekt.

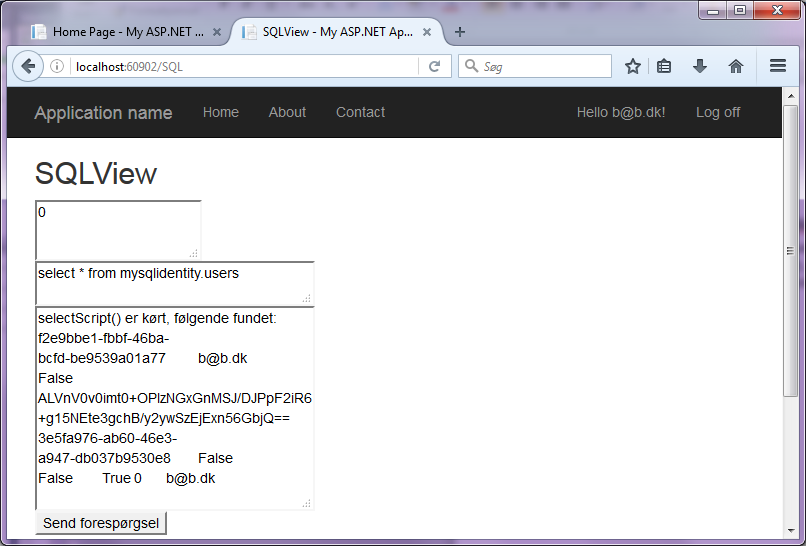
Når der bliver lavet noget nyt, skal der også være kørt en test. Første gang den køres, forventes det, at den fejler, og når der er kodet tilpas meget, og testen køres igen, skal den blive grøn. Det er naturligvis også tilfælde her, hvor det tjekkes om der er forbindelse til den nye database.





Herefter startes applikationen og en bruger registreres. I dette tilfælde er det brugeren [b@b.dk](mailto:b@b.dk), som det kan anes oppe i højre hjørne i billedet.

Ved at gå ind på siden SQL, hvor et vilkårligt SQL-script kan køres fra, hvilket muligvis kan opfattes som en sikkerhedstrussel, vælges alle registrerede brugere. Her genfindes så den nyligt registrerede bruger.



Downside ved at anvende Microsfts identitests framework er, at det kompromitterer den valgte datamodel. Det betyder så også, at applikationens funktionalitet bliver lidt anderledes en oprindeligt tænkt.

### Bygning af Voresjazzklub.dk – de første skridt

Den første rudimentære udgave af modellen for tabellen for brugere: Users.

Der er to forskellige udgaver af prim og erAdmin, da datatypen i databasen er tekst, men rent logisk er det en boolsk værdi. C# kan ikke blande datatyper ved get/set, så derfor er der to properties for hver værdi. Disse properties bliver synkroniseret ved set.

Der er oprettet en metode, hentBruger, men som det fremgår, er den aldeles triviel og ganske værdiløs; bortset fra, at den er nødvendig for at kunne konstruere en test.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace VoresJazzklubSomMVC.Models {

public class UsersTableModel {

private bool \_prim, \_erAdmin;

private char \_primChar, \_erAdminChar;

public string brugerId { get; set; }

public DateTime createTs { get; set; }

public string password { get; set; }

public string salt { get; set; }

public string email { get; set; }

public bool prim {

get { return \_prim; }

set { \_prim = value; \_primChar=(\_prim)?'T':'F'; }

}

public char primChar {

get { return \_primChar; }

set { \_primChar = value; \_prim = \_primChar == 'T'; }

}

public bool erAdmin {

get { return \_erAdmin; }

set { \_erAdmin = value; \_erAdminChar = (\_erAdmin)? 'T':'F'; }

}

public char erAdminChar {

get { return \_erAdminChar; }

set { \_erAdminChar = value; \_erAdmin = \_erAdminChar=='T'; }

}

public void hentBruger(string brugerId) {

}

}

}

Tanken med klassen er, at når metoden hentBruger bliver kaldt, skal den finde brugeren på MySQL-databasen i tabellen voresjazzklub.Users og sætte klassens properties i overensstemmelse med de fundne data.

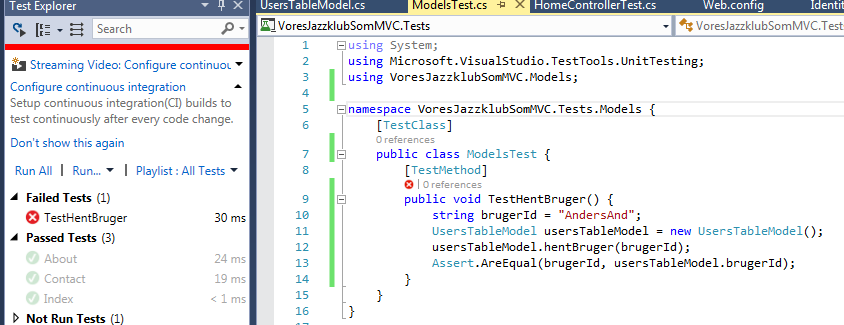
Følgende script køres i phpmyadmin web-siden i xampp mod MySQL for at have noget testdata. Senere vil det komme til at indgå i et test-script, således, at tabellen tømmes før testserierne og undervejs vil de enkelte rækker blive lagt ind, ændre og slettet for at gennemføre test af controllers og models.

insert into users(`brugerId`, `createTs`, `password`, `salt`, `email`, `prim`, `erAdmin`)

values('AndersAnd', CURRENT\_TIMESTAMP, 'PasswordDerSkalHashes', 'medNogetSalt', 'a.and@andeby.utopia', 'T', 'F');

En testklasse oprettes til at undersøge, om en given person rent faktisk kan findes. I metoden TestHentBruger skrives navnet på en bruger, der skal kunne findes. Der oprettes en instans af klassen, der skal testes, og herefter kaldes, den metode, der skal testes. Til sidst sammenlignes brugerId i testklassen med brugerId i den testede klasse.

I nedenstående skærmdump ses tydeligt, at testen fejlede. Ikke at det kommer som en overraskelse, da der jo netop ikke er nogen kode til at udføre det ønskede arbejde. Hvis det havde været en stor, kompleks applikation med dependency injection i configuration filer og andre sløringer af, hvor data kommer fra, ville situationen kunne se anderledes ud.



Man starter med testmetoden. IDE’et vil ikke kunne kompilere testmetoden, da der hverken er en klasse eller metode IDE’et kan finde referencer til. Så laves det minimum, der skal til at kunne kompilere, hvorefter testen køres.

De næste trin er at tilføre og ændre koden, i den testede metode, indtil testen bliver grøn. Alle test, til de berørte komponenter, skal også køres igen, for at se, om man har fået ødelagt noget.

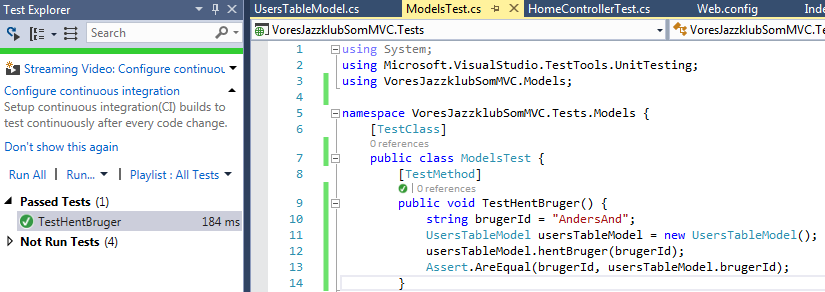
Sidste step er at checke sourcen incl test ind i ens versionsstyringssystem, hvortil der knyttet et Continuous Integration (CI) set-up [19], hvilket betyder, at der bliver kørt et komplet build og alle test bliver kørt. Testene skal så gerne være grønne alle sammen.

Til et miniature skole/hobby-projekt, virker CI som overkill, men tanken om at indlede med en test ,og herefter ændre koden og teste indtil tingen virker, kan man sagtens gennemføre på udviklingsmaskinen, og så en gang imellem køre samtlige test.

Efter lidt tilføjelse af kode og variable, blev testen grøn og glad. Resultatet er ganske vist ikke kønt at se på, der mangler noget kodning, og fejlhåndteringen skal ændres. Det skal også overvejes, hvor connection.Close(); skal indsættes; men hovedformålet, at få en grøn test, blev nået.

I koden refereres klassens properties prefixed med this. for ikke at ramme metodens variable i stedet. Ser man godt efter, konstateres to forskellige farver på this. Det er fordi det kun er ved this.brugerId, hvor det er nødvendigt, men hellere skrive det nogle flere gange fremfor at buge tid på fejlfinding.





Det har vist sig at være en særdeles dårlig ide at anvende tekst som nøgle i database. Ikke fordi der var problemer i forhold til MySQL, men navnene på billederne kan indeholde både punktum, bindestreg og skråstreg. Det fungerer blot ikke med Asp.Net MVC, da link til detaljer og redigering indeholder teksten præcis som den ser ud i databasen, og det giver problemer i forhold til routing.

Nogle forsøg på at ændre RouteConfig, til at acceptere ”besynderlige” tekster som nøgle, hjalp ikke i tilstrækkeligt omfang. Tilbage var så kun at ændre i tabellen og de tilhørende web-sider, Controller og Model. Helt konkret blev der oprettet en ny tabel med en automatisk nummerering, der så anvendes som nøgle:

create table Billeder2(billedId BIGINT UNSIGNED NOT NULL AUTO\_INCREMENT UNIQUE,

billedAdresse varchar(255) not null, arrangementId BIGINT not null, brugerId varchar(255) not null, createTs timestamp(6) not null, kommentar varchar(255) not null, primary key(billedId));

Tilmeldinger 🡪 TilmeldingersController

id i routeConfig 🡪 id i controller

View oprettes sammen med Controller

### Test/Validering

Et af målene med opgaven er, at gennemføre udviklingen som TDD – Test Driven Development. I bogen Clean Code [5] pp 122 beskrives tre regler for TDD:

1. Der må ikke skrives produktionskode, før der er skrevet en fejlende test
2. Der må ikke skrives mere i testen, end nødvendigt for, at den kan fejle
3. Der må ikke skrives mere produktionskode, end nødvendigt for succes ved testen

Herefter argumenteres for, at man kan få dynger af tests, der skal vedligeholdes og hvor mange af dem med tiden bliver overflødige.

Noget af det første, der blev skrevet til Voresjazzklub efter TDD-principperne, var en test af, om der var hul igennem til den database, der holder information om brugerne (MySQLIdentity). Hvis det nu var sådan, at tabellerne skulle flyttes til en anden database, skal testen justeres til den nye database før den eksisterer. Herefter vil testen fejl e og gå godt, når databasen er oprettet.

Når projektet er kommet så langt, at det kunne være relevant at ændre på MySQLIdentity-databasen, vil der være adskille andre test, der tilgår databasen i forbindelse med oprettelse af brugere. De test vil så overlappe hulIgennemTilIdentityDB, hvorfor den så er overflødig.

Der vil ikke være nogen grund til at have overflødige test i projektet, derfor skal de fjernes, når man er sikker på at testen bliver dække af en eller flere andre test.

Med en passende code-coverage vil TDD gøre, at man kan være rimeligt sikker på at være på rette vej med de ting, der bliver udviklet og ændret.

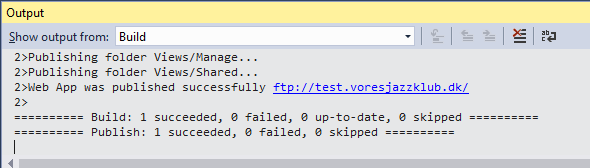
Hver gang der er skrevet en mængde kode, der synes at være i orden, køres samtlige test igen. I et lidt større setup, hvor flere personer skal arbejde på samme projekt, vil det være på sin plads at anvende en maskine kun til byg, som bygger i Release/Produktion-modus og kører alle test.

## Det videre forløb

Web-adressen til arrangement kunne delvist valideres ved automatisk at kontakte adressen, der indtastes

Ændring af sikkerhed

Deployment gik tilsyneladende godt, men applikationen kunne ikke køre på voresjazzklub.dk, kun lokalt på udviklingsmaskinen



Selenium

## Konklusion

Der er fremstillet en web-applikation, der kan håndtere for emnet relevante data. Data persisteres i en database og der er frembragt kode til oprettelse, ændring og sletning af data på individniveau. Der er kode til læsning af enkeltelementer og, hvor det har været gavnligt, som lister.

Applikationen er afprøvet delvist maskinelt og manuelt. Funktionaliteten, der skal anvendes, er til stede og virker. Hovedformålene (fra afsnittene Problemformulering, Afgrænsning og Teknologivalg) med opgave, at lære MVC-frameworket at kende, at anvende nyere teknologier som HTML5 og CSS3 og at fremstille en dynamiske web-applikation på walking skelleton-niveau, er nået.

Der er givetvis flere forhold i applikationen, der med fordel kan ændres. F.eks. er betjeningen ikke nødvendigvis særlig UX-venlig. Det skønnes, at der er plads til forbedringer, selvom der også er lækre ting imellem, som f.eks. listen af små billeder, hvor de enkelte billeder kan forstørres ved blot at holde musen hen over dm.

Datamodellen ville være anderledes, hvis Identity framework var kendt på forhånd. Alternativt skulle der have været brugt en del tid på at justere i frameworket for at akkommodere den valgte datamodel.

Portalløsning med fælles login ..?

asp.net MVC kan en milliard ting, så det er ikke et framework man sætter sig ind i på en weekend.

Bindinger på metodernes navne i Controller til RouteConfigs navngivning

## Litterarurliste

[1] <http://hvemkommer.dk/> ejet af Billetto, sidst besøgt 1. marts 2017

[2] [https://da.wikipedia.org/wiki/HTML5 HTML5](https://da.wikipedia.org/wiki/HTML5%20HTML5) blev offentliggjort oktober 2014, siden besøgt 1. marts 2017, hvor sidste opdatering var: ”Denne side blev senest ændret den 21. oktober 2016 kl. 16:07”

[3] <http://www.web10.dk/v6_information/pricelist>, det koster 8 kr. pr. md. og det kan en fattig studerende ikke klare.WEB10.dk er hostingselskabet som lægger severe til sitet. Prisen er sidst set 1. marts 2017

[4] <https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha512managed%28v=vs.110%29.aspx>, set 16. marts 2017

[5] Clean Code ISBN 978-0-13-235088-4, Robert C. Martin, Prentice Hall 2013

[6] <http://www.web10.dk/v5_order/packs_basic>, sidst besøgt 1. marts 2017

[7] <https://github.com/ClausTetens/WEB01Projekt> indeholder sourcen til web-sitet

[8] https://msdn.microsoft.com/en-us/library/ms182776.aspx, set 16. marts 2017

[9] https://dev.mysql.com/doc/refman/5.7/en/fractional-seconds.html, set 16. marts 2017

[10] https://www.w3schools.com/sql/sql\_datatypes.asp, set 16. marts 2017

[11] <https://www.ischool.utexas.edu/~wyllys/DMPAMaterials/normstep.html>, set 16. marts 2017

[12] Fundamentals of Database Systems, ISBN 0-201-53090-2, Elmasri/Navathe, The Benjamin/Cummings Publishing Company Inc, 1989

[13] https://dev.mysql.com/doc/refman/5.7/en/numeric-type-overview.html, set 16. marts 2017

[14] <http://stackoverflow.com/questions/8437957/difference-between-3nf-and-bcnf-in-simple-terms-must-be-able-to-explain-to-an-8>, set 16. marts 2017

[15] <https://www.apachefriends.org/index.html>, besøgt 15. april 2017

[16] <https://docs.microsoft.com/en-us/aspnet/identity/overview/extensibility/implementing-a-custom-mysql-aspnet-identity-storage-provider>, set 15. april 2017

[17] https://aspnet.codeplex.com/SourceControl/latest#Samples/Identity/AspNet.Identity.MySQL/, set 15. april 2017

[18] <https://www.ics.com/files/docs/qicstable/2.4/mvc.jpg>, set 1. april 2017

[19] <https://martinfowler.com/articles/continuousIntegration.html>, set 16. april 2017

[20] <https://msdn.microsoft.com/en-us/library/ms182538%28v=vs.90%29.aspx>

[21] <https://blogs.msdn.microsoft.com/visualstudioalm/2016/01/27/getting-started-with-selenium-testing-in-a-continuous-integration-pipeline-with-visual-studio/>

Følgende henvisninger er til sider, der har bidraget med afklaringer og produkter, men som ikke direkte indgår i rapporten.

[100] <https://dev.mysql.com/doc/connector-net/en/connector-net-installation-binary-windows-installer.html>, set 16. april 2017

[101] http://machinesaredigging.com/2013/10/29/how-does-a-web-session-work/ , set 16. april 2017

[102] <https://en.wikipedia.org/wiki/List_of_HTTP_status_codes>, set 16. april 2017

[103] <http://www.prideparrot.com/blog/archive/2012/7/understanding_routing>, set 16. april 2017

[104] <https://www.asp.net/identity>, set 16. april 2017

[105] <https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/>, set 16. april 2017

[106] <https://www.apachefriends.org/index.html>, set 16. april 2017

[107] <https://www.asp.net/downloads>, set 16. april 2017

[108] <https://dev.mysql.com/downloads/connector/net/5.2.html>, set 1. april 2017

[109] <https://www.codeproject.com/Articles/822392/Connecting-to-MySQL-from-ASP-NET-MVC-using-Visual>, set 1. april 2017

[110] <https://www.mysql.com/why-mysql/windows/visualstudio/>, set 1. april 2017

## Appendix

### Lidt om, hvordan programkoden bliver til

Til projektet er der ikke kopieret kode direkte, men der er ofte kun en måde, når der skal vælge fra best practice. Ofte kan man finde code sippets og anbefalinger på nettet. I nærværende projekt er det typisk hjælpesider fra Microsoft, der er leverandør af bl.a. C# og .Net, og uafhængige sites som Stackoverflow.

F.eks. er koden til SHA512-hashing delvis hentet hos Microsoft [4] og ser sådan ud:

byte[] data = new byte[DATA\_SIZE];

byte[] result;

SHA512 shaM = new SHA512Managed();

result = shaM.ComputeHash(data);

Der var et ønske om af opbevare forvanskede data i datatypen string. Best practice for konvertering af et array af bytes, er, at løbe arrayet gennem med foreach og opbygge en streng med StringBuilder. Med tanker om læsbarhed og vedligeholdelsesvenlighed hentet i f.eks. Clean Code bogen [5], er rammerne relativt faste. Derfor koden snublende nær:

string getHashSha512(string stringToBeHashed) {

byte[] bytesToBeHashed = Encoding.Unicode.GetBytes(stringToBeHashed);

SHA512 sha512 = new SHA512Managed();

byte[] hashedBytes = sha512.ComputeHash(bytesToBeHashed);

StringBuilder hashedString = new StringBuilder();

foreach(byte hashedByte in hashedBytes) {

hashedString.Append(String.Format("{0:x2}", hashedByte));

}

return hashedString.ToString();

}

En lidt mere kompakt udgave kunne se sådan ud:

string getHashSha512(string stringToBeHashed) {

const int bitsToCharShiftCount = 2; // 8 bits/byte and 2 characters/byte

SHA512 sha512 = new SHA512Managed();

StringBuilder hashedString = new StringBuilder(sha512.HashSize >> bitsToCharShiftCount);

foreach(byte hashedByte in sha512.ComputeHash(Encoding.Unicode.GetBytes(stringToBeHashed))) {

hashedString.Append(String.Format("{0:x2}", hashedByte));

}

return hashedString.ToString();

}

eller sådan:

string getHashSha512(string stringToBeHashed) {

StringBuilder hashedString = new StringBuilder();

foreach(byte hashedByte in new SHA512Managed().ComputeHash(Encoding.Unicode.GetBytes(stringToBeHashed))) {

hashedString.Append(String.Format("{0:x2}", hashedByte));

}

return hashedString.ToString();

}

Der er ikke gjort nogle forsøg på at finde en tilsvarende kodestumper på nettet, men der er en forventning om, at det kan lade sig gøre.

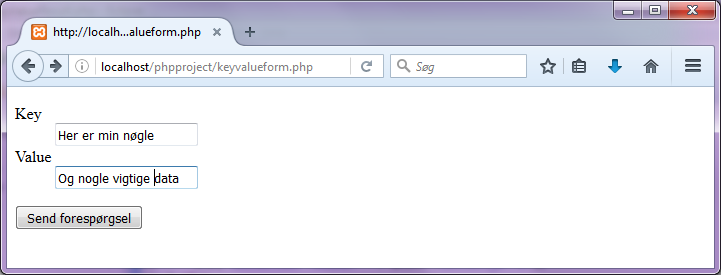
Visual Studio 2015, som er Microsofts IDE til bl.a. C# .Net, indeholder flere templates, herunder et komplet, fungerende web-site. Det kunne med fordel vælges, hvis man skal hurtigt i gang, idet der er anvendt gennemtænkte løsninger til de forskellige funktioner. Ulempen er så, at det det er noget omfattende og måske kan være for omfattende at finde rundt i, hvilket kan lede til fejltagelser og være tidskrævende.

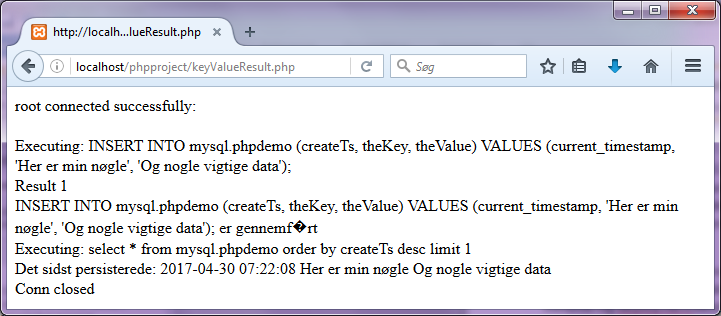
### Noget hurtig PHP

Nedenstående PHP-scripts er lidt noget rod at se på og det eksisterer da også kun for at kunne se, hvordan PHP og processen med at få en LAMP/WAMP-applikation til at køre i praksis fungerer. Det hele (incl. oprettelse af tabellen) har taget en lille halv time at fremstille og deployment består også i en meget simpel operation: Kopier PHP-filen til den folder, Apache skal læse den fra.

Nedenstående to skærmdumps viser resultatet af de to PHP-scripts, der er gengivet derunder. En HTML form tager to tekst-input og submitter til et modtager-script. Det persisterer de indtastede data sammen med current\_timestamp. Hvorefter nyeste række læses og returneres.

Man kommer hurtigt fra nul til noget anvendeligt. Hvis det er meningen, at det også skal kunnet vedligeholdes, bør man nok tænke struktur og genbrug. Hvis der ligefrem også er krav om, at det skal se pænt ud og sikret mod fejl, så medgår der en del mere tid på styling og validering af data.





Keyvalueform.php

<html><body>

<form method=*"post"* action=*"/phpproject/keyValueResult.php"*>

<dl>

<dt>Key</dt>

<dd><input name=*"akey"* type=*"text"*></dd>

<dt>Value</dt>

<dd><input name=*"avalue"* type=*"text"*></dd>

</dl>

<input type=*"submit"*>

</form>

</body></html>

KeyValueResult.php

<html><body>

<?php

$servername = "localhost";

$username = "root";

$password = "";

$conn = **new** mysqli($servername, $username, $password);

*mysqli\_select\_db*($conn, 'mysql');

**if** (!$conn) {

*die*("Connection failed: " . $conn->connect\_error);

} **else**

**echo** $username . " connected successfully: " . $conn->connect\_error . "<br/><br/>";

$akey = **$\_POST**["akey"];

$avalue = **$\_POST**["avalue"];

$sql="INSERT INTO mysql.phpdemo (createTs, theKey, theValue) VALUES (current\_timestamp, '$akey', '$avalue');";

**echo** "Executing: " . $sql;

*printf*("<br/>");

$result=$conn->query($sql);

*printf*("Result %s<br/>", $result);

**if**($result) { *printf*("%s er gennemført<br/>", $sql); }

**else** { *printf*("%s fejlede<br/>", $sql); }

$sql="select \* from mysql.phpdemo where createTs=(select max(createTs)) from mysql.phpdemo)";

$sql="select \* from mysql.phpdemo order by createTs desc limit 1";

**echo** "Executing: " . $sql;

*printf*("<br/>");

$result=$conn->query($sql);

**if**($result) {

**echo** "Det sidst persisterede: ";

**while**($row = *mysqli\_fetch\_array*($result)) {

**for**($i=0; $i<3; $i++) {

*printf*("%s ", $row[$i]);

}

*printf*("<br/>");

}

}

**echo** $conn->close()?"Conn closed":"Error closing connection";

?>

</body></html>

Nedenstående PHP-script er noget rod at se på og det eksisterer da også kun for at kunne se, hvordan PHP og processen med at få en LAMP/WAMP-applikation til at køre i praksis fungerer. Scriptet er skrevet på få minutter og deployment består også i en meget simpel operation: Kopier PHP-filen til den folder, Apache skal læse den fra.

Scriptet opretter en række i en tabel, læser den tilbage og display’er den på en web-side.

<html><body>

<?php

**echo** "php is running<br/><br/><br/><br/>";

$servername = "localhost";

$username = "claus";

$password = "";

**echo** "username $username<br/>";

**echo** 'username $username<br/>';

$conn = **new** mysqli($servername, "root", "");

*mysqli\_select\_db*($conn, 'mydb');

**if** (!$conn) {

*die*("Connection failed: " . $conn->connect\_error);

} **else**

**echo** $username . " connected successfully: " . $conn->connect\_error . "<br/><br/>";

$sql="INSERT INTO mysql.user (host, user, password, select\_priv, insert\_priv, update\_priv) VALUES ('localhost', 'Gudrun',PASSWORD('guest123'), 'Y', 'Y', 'Y');";

**echo** "Executing: " . $sql;

*printf*("<br/>");

$result=$conn->query($sql);

*printf*("Result %s<br/>", $result);

**if**($result) { *printf*("%s er gennemført<br/>", $sql); }

**else** { *printf*("%s fejlede<br/>", $sql); }

$sql="select \* from mysql.user where user='Gudrun'";

**echo** "Executing: " . $sql;

*printf*("<br/>");

$result=$conn->query($sql);

**if**($result) {

**while**($row = *mysqli\_fetch\_array*($result)) {

*printf*(":::");

**for**($i=0; $i<21; $i++) {

*printf*("\t>>%s<<,", $row[$i]);

}

*printf*("<br/>");

}

} **else** {

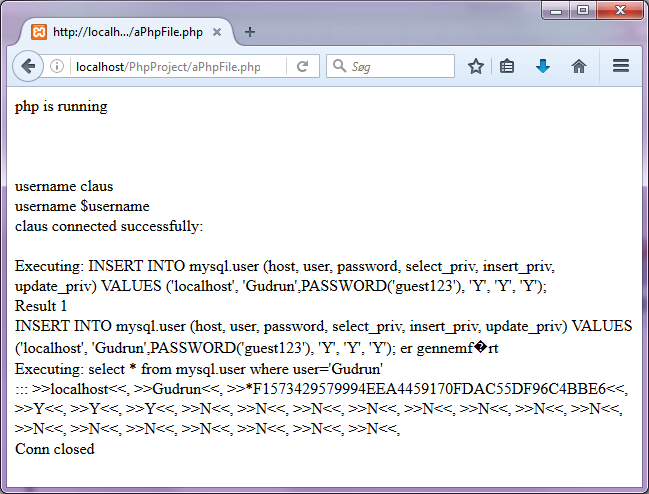
}

**echo** $conn->close()?"Conn closed":"Error closing connection";

?>

</body></html>

Kønt er barnet ikke, men der kommer data i en tabel, data bliver læst tilbage og vist på glaspladen. Browseren skal vist også have at vide, at det nok ikke er UTF-8, men muligvis ISO8859-1 der anvendes.



### Noget hurtig ASP.NET

ASP.NET i traditionel udgave består primært af to filer på side. Den ene holder html-tags, og den anden, som bliver refereret til via CodeBehind, holder den C#-kode der skal afvikle forretningslogikken.

Kommunikationen mellem web-siden og koden bagved foregår via tags i namespace asp som f.eks. nedenfor, hvor <asp:Contents… fortæller, at her er der noget C#-koden skal forholde sig til. Alle html-elementer placeres i asp-namespace, så compileren kan knytte en begivenhed på web-siden sammen med handleren.

<%: Title %> bliver erstattet med ”Contact” fra øverste linje. En konstruktion som f.eks. <%= this.navn %> og en public, læsbar property med navnet navn, vil indsætte C#-klassens property i html-koden.

**Contact.aspx**

<%@ Page Title="Contact" Language="C#" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeBehind="Contact.aspx.cs" Inherits="VoresJC.Contact" %>

<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">

<h2><%: Title %>.</h2>

<h3>Your contact page.</h3>

<address>

One Microsoft Way<br />

Redmond, WA 98052-6399<br />

<abbr title="Phone">P:</abbr>

425.555.0100

</address>

<address>

<strong>Support:</strong> <a href="mailto:Support@example.com">Support@example.com</a><br />

<strong>Marketing:</strong> <a href="mailto:Marketing@example.com">Marketing@example.com</a>

</address>

</asp:Content>

**Contact.aspx.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

namespace VoresJC {

public partial class Contact : Page {

protected void Page\_Load(object sender, EventArgs e) {

}

}

}

### Script til migrering af adgangskontrol

create database MySQLIdentity;

use MySQLIdentity;

CREATE TABLE `roles` (

`Id` varchar(128) NOT NULL,

`Name` varchar(256) NOT NULL,

PRIMARY KEY (`Id`)

);

CREATE TABLE `users` (

`Id` varchar(128) NOT NULL,

`Email` varchar(256) DEFAULT NULL,

`EmailConfirmed` tinyint(1) NOT NULL,

`PasswordHash` longtext,

`SecurityStamp` longtext,

`PhoneNumber` longtext,

`PhoneNumberConfirmed` tinyint(1) NOT NULL,

`TwoFactorEnabled` tinyint(1) NOT NULL,

`LockoutEndDateUtc` datetime DEFAULT NULL,

`LockoutEnabled` tinyint(1) NOT NULL,

`AccessFailedCount` int(11) NOT NULL,

`UserName` varchar(256) NOT NULL,

PRIMARY KEY (`Id`)

);

CREATE TABLE `userclaims` (

`Id` int(11) NOT NULL AUTO\_INCREMENT,

`UserId` varchar(128) NOT NULL,

`ClaimType` longtext,

`ClaimValue` longtext,

PRIMARY KEY (`Id`),

UNIQUE KEY `Id` (`Id`),

KEY `UserId` (`UserId`),

CONSTRAINT `ApplicationUser\_Claims` FOREIGN KEY (`UserId`) REFERENCES `users` (`Id`) ON DELETE CASCADE ON UPDATE NO ACTION

);

CREATE TABLE `userlogins` (

`LoginProvider` varchar(128) NOT NULL,

`ProviderKey` varchar(128) NOT NULL,

`UserId` varchar(128) NOT NULL,

PRIMARY KEY (`LoginProvider`,`ProviderKey`,`UserId`),

KEY `ApplicationUser\_Logins` (`UserId`),

CONSTRAINT `ApplicationUser\_Logins` FOREIGN KEY (`UserId`) REFERENCES `users` (`Id`) ON DELETE CASCADE ON UPDATE NO ACTION

);

CREATE TABLE `userroles` (

`UserId` varchar(128) NOT NULL,

`RoleId` varchar(128) NOT NULL,

PRIMARY KEY (`UserId`,`RoleId`),

KEY `IdentityRole\_Users` (`RoleId`),

CONSTRAINT `ApplicationUser\_Roles` FOREIGN KEY (`UserId`) REFERENCES `users` (`Id`) ON DELETE CASCADE ON UPDATE NO ACTION,

CONSTRAINT `IdentityRole\_Users` FOREIGN KEY (`RoleId`) REFERENCES `roles` (`Id`) ON DELETE CASCADE ON UPDATE NO ACTION

) ;