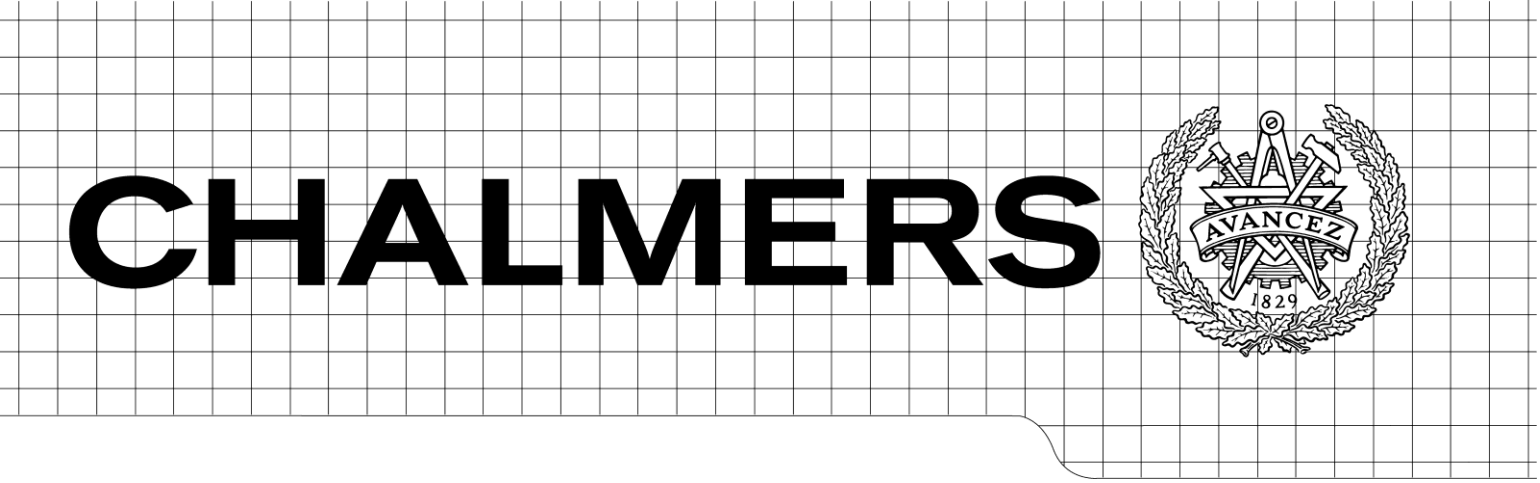



# CorporateFaces

Ett edutainment-spel för ökad sammanhållning

MARTIN BERGQVIST

JOAKIM JONSSON

*Examensarbete*

*Högskoleingenjörsprogrammet för datateknik*

CHALMERS TEKNISKA HÖGSKOLA
Institutionen för data- och informationsteknik
Göteborg 2012

*“Coming together is a beginning.*
*Keeping together is progress.*
*Working together is success.”*

*Henry Ford* 1863-1947
*Founder of Ford Motor Company*

## Abstract

Consulting companies have their employees working with companies on different geographical locations. In the nature of a shattered organization, employees tend to communicate less since they rarely know who is working where. In the long run, this is damaging the performance of companies. This paper describes a tool that can and should be used to enhance the relations between the people in those kinds of companies. The solution is a quiz-like game named CorporateFaces with the purpose of challenging employees to gain knowledge about each other. CorporateFaces is an online, web based game but also available in smartphones, both through web and native application. Even though the evaluation is not fully complete, according to scientific mathematical methods, our evaluation of this project is showing that the knowledge about other employees has gained after participated in the game.



## Sammanfattning

Företag i allmänhet och konsultbolag i synnerhet har idag en splittrad organisation, där anställda ofta är ute på uppdrag eller helt enkelt inte är stationerade på sitt företag. Denna företagsstruktur bidrar till att de anställda har mindre och mindre kommunikation sinsemellan, vilket i slutändan resulterar i sämre prestationer eftersom sammarbetet sviker. I denna rapport tas ett hjälpmedel fram för att främja kunskapen och minska trösklarna mellan de anställda, på ett trevligt och kul vis, med hjälp utav ett spel. CorporateFaces är ett quizliknande spel i strukturen och går ut på att samla poäng, som görs via besvarande utav frågor formulerade utifrån kollegor och dess attribut. Spelet är webbaserat men även utveckling för smarta telefoner finns. Resultatet av spelet visar tendenser på att kunskapen om anställda ökar signifikant, även om rapporten ej har fullständiga mätserier för en korrekt statistiskt säkerställd utvärdering.



Keywords: Web, Social environment, Quiz-game, .Net, MVC

# 1. Inledning

## 1.1 Bakgrund

Sigma AB är ett IT-konsult bolag med ca 1400 anställda i nio länder. Företagets inriktning varierar från att bygga hela kompletta “back-end” lösningar (där man bygger upp server, webb och andra större lösningar för kund) till små underhåll eller utveckling av enstaka komponenter åt andra företag.

IT-konsultföretag är vanligen uppbyggda av mindre interna divisioner som arbetar separat med olika projekt och kunder. Som anställd på ett sådant bolag jobbar man ofta intensivt inom sin grupp och det är inte alls ovanligt att man utför uppdrag på andra företag. Eftersom personalen sällan träffar folk från andra delar av företaget är det svårt att bygga upp en relation med de anställda som inte ingår i den egna gruppen.

Ett problem som dyker upp i och med denna gruppering är att de anställda inte ser på sina medarbetare som lagkamrater som kan bistå med hjälp, utan istället skapas det en "vi och dem känsla".

Kärnidén i ett företag är att tjäna mer pengar, detta kan man göra genom att effektivisera och vara kreativ för att skapa nya lösningar. Om företaget då inte utbyter idéer och lösningar internt kan effektiviteten gå förlorad. En grupp inom företaget kan sitta och bearbeta ett problem fast det redan finns en lösning. Kompetens som finns inom vissa delar av organisationen kanske inte finns i andra grupperingar. Om ett utbyte av idéer och tankar grupperingarna sinsemellan kan genomföras inom företaget finns möjligheten att kreativiteten kan öka och tillsammans kan man utveckla nya smarta lösningar.

För närvarande finns många verktyg för kommunikation trots att den geografiska eller organisatoriska verksamheten inte sammanfaller med allas placering.

Ett av verktygen är framförallt de mobila enheterna, så som mobiltelefoner och bärbara datorer som idag är avancerade och uppkopplade mot internet. Genom att bygga en applikation till dessa enheter som kan hjälpa företagets anställda att lära känna varandra och framför allt minnas varandra på ett underhållande sätt, kan man på sikt minska trösklarna som råder inom företaget. De anställda kommer då känna att man kan kommunicera även med andra utanför sin egen division inom organisationen.

## 1.2 Problemet

Det finns ett behov av att anställda inom organisationer har kännedom om varandra. Detta dels för att underlätta kommunikationen mellan individerna och dels för att öka sammanhållningen inom organisationen.

Det saknas enkla och lättillgängliga verktyg för att lösa det här problemet.

## 1.3 Syfte

Den här rapporten vill påvisa att underhållande, enkla och lättillgängliga applikationer kan användas för att minska de interna trösklarna som idag finns på grund av separationen av företagets olika delar. Detta ska leda till förbättrad sammanhållning och ökad effektivitet som i slutändan resulterar i att organisationen presterar bättre.

## 1.4 Avgränsning

På grund av en begränsad tidsrymd för projektet fanns inte möjlighet att genomföra en fullständig statistiskt korrekt utvärdering utav resultatet.

# 2. Metod

Genom att utveckla ett spel för webbläsare och mobiltelefoner för en organisation där medlemmarna själva är en del av spelet, där spelaren får svara på frågor om andra medlemmar, kan man öka kunskapen om de som ingår i organisationen på både ett underhållande och lärorikt vis.

När utvecklingen av spelet var tillräckligt utförd fick ett antal test-personer spela spelet som innehöll frågor om kollegorna. Primärt, för rapportens räkning, krävs en metod för att validera problemhypotesen, om det med hjälp av en spelapplikation går att öka individernas kännedom om varandra. För att åstadkomma detta användes en trestegsmetod:

*1.* Visade fotografier av samtliga i personalen som ingår i spelet, där namn ej angavs. Lät vederbörande besvara dessa utifrån egen förmåga och noterade resultatet utan att avslöja detta för deltagaren.

*2.* Lät den berörda personen spela en omgång á 90 frågor i spelet.

*3.* Gjorde om testet i (1), noterade återigen resultatet och drog slutsatser.

# 3. Teknisk bakgrund

I detta kapitel ges en beskrivning av den teknik som använts under utvecklingen av spelet.

## 3.1 HTML

HTML (Data, 1999-2012) är en förkortning av Hyper Text Markup Language. HTML beskriver en webbsida med hjälp av markup-taggar. HTML är alltså inget programmeringsspråk utan ett markupspråk. Ett markupspråk kan definieras som en samling markup-taggar. Markup-taggar kallas vanligtvis för HTML-taggar.

## 3.2 JSON

JavaScript Object Notation, vanligen JSON, är ett textbaserat sätt att representera serialiserad data. JSON kan representera strängar, nummer, booleaner, objekt och listor. (Society, 2006)

Ett typexempel på hur en JSON kan se ut:
Låt säga att vi har en lista objekt, där varje objekt definieras enligt följande:

```
{
“parameter1” : “värde”
“parameter2” : “värde”
}
```

JSON-listan, definieras då inom ett par klamrar, “[“ respektive “]” där varje objekt separeras av ett komma.
Exempel:

```
[{“parameter1”:”värde”,”parameter2”:”värde”},{objekt2}]
```

## 3.3 AJAX

AJAX (Wikipedia, 2012) är en förkortning för Asynchronous JavaScript And XML. AJAX används för att göra anrop från JavaScript och därefter uppdatera delar av en sida.

# 3.4 JQuery

Ett kompletterande bibliotek till det vanliga JavaScript som har en högre abstraktionsnivå och inte är beroende av vilken webbläsare som används (JQuery.org).

JQuery har inbyggt stöd för fler avancerade metoder än JavaScript, t.ex AJAX och hantering utav JSON.

# 3.5 CSS

CSS (W3C ( MIT 2012), Cascading Style Sheet, är ett separat dokument med avsikt för att ge HTML-koden den visuella utformning som önskas.

CSS definierar vanligen klasser och typer.

Klasser har för avsikt att ge alla HTML-element som implementerar klassen samma visuella presentation.

Typer används för att ge specifika HTML-element dess grafiska utformning.

# 3.6 MVC

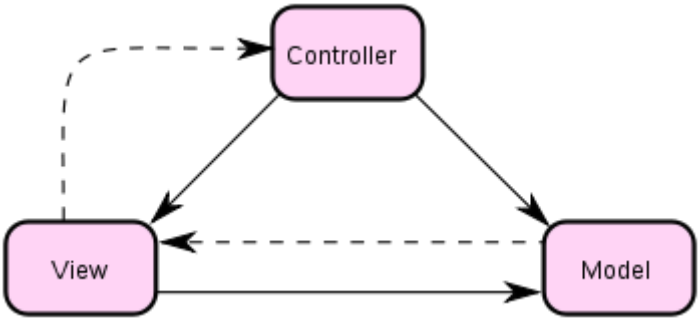
MVC är ett klassiskt designmönster som lämpar sig väl vid utveckling av en Webbserver.

MVC-modellen består av tre delar:

**Model** - Ett objekt med data som skall skickas till/från vyerna (Views). Används för att man endast skall få med exakt så mycket information som är nödvändigt, varken mer eller mindre.

**View** - Representerar en vy, i vårt fall webbsidan, och känner till just precis vilken model som skall användas. Vyn innehåller också HTML-kod, för att representera sidan.

**Controller** - Serverkontrollen, varje Controller-klass innehar metoder som returnerar vyer och tar modeller eller annan information som inparametrar. Här görs olika val för vilken vy som skall returneras, grundläggande serverlogik och kontroll utav inparametrar.



Modell över MVC, illustrerad av (Davjoh, 2010)

## 3.7 Microsoft Visual Studio

Mircosoft Visual Studio är microsofts egna utvecklingsverktyg som med hjälp av många olika utvecklingstekniker, hjälpmedel och språk gör det kompetent och mångsidigt.

I utvecklingsmiljön finns möjlighet att skriva kod i allt från maskinära språk (Visual C++), funktionella språk (F#) och objektorienterade språk (C#). Dessutom finns möjligheten till att utveckla javascript och CSS filer.

## 3.8 Microsoft SQL

Databasen som användes i detta projekt är en Microsoft SQL-server, denna server lämpar sig väl eftersom .NET har stöd för bland annat Microsofts egna databas, med möjligheter till att skapa tabeller och relationer automatiskt utifrån de domänmodeller man skapat.

## 3.9 Responsive Web Design (RWD)

RWD är ett koncept som innebär att man designar webbsidan dynamiskt utifrån vilken skärmstorlek och upplösning webbsidan visas på. Detta är en ren CSS(W3C ( MIT 2012) implementation som görs genom att man definierar utseende utifrån utstakade upplösningar, elementens storlek, placering och synbarhet.

RWD går att implementera på flera olika vis och med stöd av olika ramverk. I denna rapport har vi valt just ett specifikt ramverk; Less Framework (Korpi).

Less Framework en enkel implementation som bygger på att man tar sin befintliga CSS och bygger ut den med så kallade “Media Queries” som känner av hur bred skärm webbsidan visas på.

## 3.10 ASP.NET / .NET 4

.NET i Visual Studio tillåter att man kombinerar HTML och i detta fall C# för snabb och enkel applikationsutveckling för webb(Paharia, 2011). C# står för den logiska delen och HTML mer för den visuella. Sedan år 2007 har Microsoft utvecklat en alternativ form till .NET, .NET MVC som implementerar MVC-modellen i .NET(Nayyeri, 2009) (se kap 3.14).

# 3.11 Razor View Engine

Razor View Engine (ScottGu, 2010), eller vanligen förkortat Razor, är ett sätt att förenkla kodningen av ett HTML-dokument i Asp.NET.

Med hjälp av Razor kan man i HTML använda syntaxet @, vilket gör det möjligt att i stället för HTML skriva .NET-kod och få en kodstruktur som liknar vanlig C#. Detta kan med fördel användas för att göra loopar, if:satser och liknande, något som inte är möjligt utan att använda JavaScript eller JQuery annars.

Exempel på användning av Razor View Engine; Låt säga att Model är en godtycklig lista med Objekt av typen SomeObject som har en definierad sträng, det skulle då kunna se ut så här:

*Model= [{value="först"}, {value="sedan"}, {value="senare"}, {value="sist"}]:*

```
<HTML>
        @{
        Foreach( SomeObject x in Model)
        <H2> x.value</H2>
        }
</HTML>
```

I detta fall skulle det resultera i en lång rubriktext från x.value, Razor tolkar om det och det blir representerat som HTML när sidan renderas:

```
<HTML>
        <H2>först</H2>
        <H2>sedan</H2>
        <H2>senare</H2>
        <H2>sist</H2>
</HTML>
```

## 3.12 LINQ

LINQ är ett bibliotek för att underlätta kommunikationen med Microsoft SQL-databaser, skrivet för C# eller VB.NET.
LINQ fungerar på sätt och vis som ett vanligt SQL-syntax, fast man skriver det i C#-kod.
Ett exempel på en LINQ-förfrågan mot databas kan se ut på följande vis:

```
...
resultat = from X in  SQLDATABAS.Tabell where x.value == "något" select X;
...
```

Där X representerar varje rad i tabellen, vi plockar endast med det om x.value är lika med "något".

## 3.13 Entity Framework

Entity Framework (Paharia, 2011) bygger på att man modellerar sina databastabeller som klasser i C# och namnger dem som en Databasmodell / Entitet.
Genom att göra detta kan man kombinera LINQ med Entity Framework och därmed erhålla en samling objekt utifrån en databasförfrågan, vanligen som en mängd.
En förfrågan mot denna typ av koppling till databasen kan se ut på detta sätt;

```
set<Model> list = Databas.Model.where( x => x.value == "någonting");
```

efter anropet kommer list innehålla alla rader från databasen som har värdet "någonting" i kolumnen value.

Värt att notera är:

```
...
x=>x.value == "någonting"
...
```

Detta är något som kallas för ett lambdauttryck (MSDN, 2012). Det är ett speciellt uttryck som opererar som en anonym funktion, det vill säga vänster sidan är indata-parametern och högersidan är resultatet. Detta kan representeras som en metod:

```
.... method(Object x)
 return x.value == "någonting";
```

# 3.14 MVC4

MVC4 (Microsoft, 2012) är i betastadiet just för tillfället, men är relativt buggfritt.

MVC4 är en påbyggnad av en tidigare variant, MVC3, med nya funktionaliteter, de nya funktionerna som nyttjas i denna rapport är Web-API:et.

Web-API:et fungerar som ett gränssnitt mot databasen och returnerar svar i form av XML eller JSON beroende på hur förfrågan ser ut. Genom att använda API:et kan man på ett enkelt och hanterbart sätt få ut nödvändig och önskvärd information från databasen, utan att få HTML-kod. Detta medför att man kan skriva JQuery / JavaScript som hämtar nödvändig information från databasen och presenterar det, utan att man behöver läsa in nya sidor. En annan fördel med denna teknik är att både webbsidan och en Mobilapplikation kan nyttja samma API och får därför samma information att arbeta med.

I applikationen gjordes ett tillägg utöver ovan nämnda MVC-mönster. För att separera databashanteringen har en klass med applikationslogik skapats som står för all hantering till och från databasen.

# 3.15 Mock-up

Vid planering av hur spelet ska se ut, grafiskt, har vi använt oss utav mock-ups. Mock-up är en benämning för grafiska skisser gjorda i program som t ex Balsamiq.

Tanken med en mock-up är att man skall få en känsla för applikationen innan man utvecklat den, vilken information som skall presenteras och en övergripande känsla för designen, även om designen i detta stadiet är primitiv.

## 3.16 Scrum

Scrum är en agil utvecklingsmetod där fokus ligger på utvecklingen utav mjukvaran istället för att fokusera på hur projektet skall genomföras.

Utvecklarna får själva organisera arbetet och bestämma hur mycket de kan producera inom en förutbestämd period.

Istället för att bestämma hur projektets gång skall gå från början till slut så delar man upp projektet i iterationer. I varje iteration finns möjlighet att förändra projektplanen utifrån de problem och önskemål som finns vid just den tidpunkten. I projektet arbetar man utifrån en ”att göra lista”, mer känd som ”backlog”. I backlogen finns uppgifterna prioriterade och högre prioriterade uppgifter genomförs först.

# 4. Genomförande

Under projektets gång har ett spel i form av en webbapplikation implementerats till företaget SIGMA IT & MANAGEMENT som vi har samarbetat med.

I det här kapitlet ges en detaljerad beskrivning för utvecklingen av applikationen, vilka verktyg som använts, applikationens uppbyggnad samt andra tankar och funderingar vi har haft under projektets gång.

Spelet gjordes i Visual Studio, .NET, enligt deras MVC4-modell, detta för att information ska skickas mellan användaren och spelets databas på ett enkelt och säkert sätt. I .NET används i huvudsak två programmeringsspråk, C#/Razor och HTML. JavaScript/JQuery kan också användas i kombination med HTML och så har vi i huvudsak också gjort när det kommer till själva spellogiken:

- Hämta frågor från databasen
- blanda frågorna
- presentera fråga
- rätta svar
- räkna poäng
- presentera några av de felsvar man haft under spelets gång

I VS har Microsoft’s “Team Foundation Server” använts som versionshanterare men också för samordning av arbete, prioriterade

funktioner osv. Detta var ett system som användes på Sigma och om vi hade frågor så fanns hjälp att tillgå, vilket gjorde valet ganska självklart.

I inledningen av projektet var vi tvungna att skissera och definiera utseende och funktioner i spelet. Detta för att effektivt kunna samordna våra tankar och idéer för spelet.

Ett verktyg för denna fas är ett program som heter ”Balsamiq”. Balsamiq är ett grafiskt program för att skissera upp gränssnittet för applikationer, så kallade Mock-ups (se appendix 4.0 ).

Utifrån dessa Mock-ups kunde man snabbt se vilka typer av funktioner och som behövdes i vårt spel. Funktionerna specificerades i ett dokument med parametrar och returvärden samt tidsuppskattades för att se om vi skulle hinna med alla. Funktionerna lades dessutom in i en prioritetslista även kallad ”backlog” (se 4.2.3), på så sätt kunde de minst viktiga funktionerna förkastas vid en eventuell tidsbrist.

Utifrån kravspecifikationen bestämdes användarfall. Användarfallen specificerades i ett dokument (se appendix 3.0). Utifrån kravspecifikation, användarfall och backloggen kunde vi planera kodningsprocessen.

Prioriteringen i projektet har hela tiden legat på att få till själva spelet, snarare än att få till andra mindre viktiga funktioner som till exempel Administratör, design, etc.

Inledningsvis låg fokus på att få till möjligheten att kunna presentera en bild tillsammans med tre stycken olika alternativ tillhörande bilden. Bilderna presenterades med hjälp av en kombination av HTML, JQuery och AJAX, och var från början inte lagrade i databasen som bilder utan var endast lagrade som URL: er till bilder som hittats från sökningar.

Databasen vid det här stadiet var primitiv och innehöll endast de mest elementära bitarna för att kunna presentera en fråga. Databasen var dessutom lokal.

Nästa steg blev att migrera till en extern databas, det gjordes väldigt enkelt eftersom vi använde oss utav Entity Framework. Bara genom att peka på att man vill använda en annan server för databasen mappades det upp av miljön, utifrån de definitioner som hade specificerats (se 4.4.6 ).

Mätningarna har genomförts med hjälp av metoden presenterad i Metod-kapitlet.

# 4.1 Kravspecifikation

En kravspecifikation på hur spelet skall fungera i grova drag.

### 4.1.2 Allmänna krav

1. Spelet skall vara användarvänligt.
   Med användarvänligt avses att användaren skall ha få valmöjligheter på varje sida.
2. Spelet skall ha en frågesports struktur.
   Med frågesport avses att spelet presenterar en fråga med tillhörande svarsalternativ.
3. Frågorna i spelet skall gå att få i olika svårighetsgrader.
4. Varje fråga skall värderas med poäng.
5. Varje spel måste tillhöra en administratör.

### 4.1.3 Funktionella krav

1. Spelet skall automatiskt kunna generera frågor.
2. Spelet skall automatiskt generera till olika svårighetsgrader.
3. Spelet skall kunna spara ett fritt antal frågor.
4. Spelet skall kunna särskilja på män och kvinnor.
5. Spelet skall spara bilder i databasen.
6. Spelet skall presentera en felmeddelandetext om en användare gör en felaktig inmatning.
7. Spelet skall automatiskt hålla räkningen på spelarens poäng.
8. Spelet skall presentera felaktigt besvarade frågor.

### 4.1.4 Icke funktionella krav

1. Spelet bör kunna laddas inom en maximal tidsrymd inom 2 sekunder.
2. Spelets skall vara kompatibelt med olika plattformar.
3. Spelet skall bara tillåta administratören att se vilka frågor som finns.
4. Spelet skall kunna spelas i olika svårighetsgrader.

# 4.2 Metodik

### 4.2.1 Veckovis uppföljning

Minst en gång i veckan har en uppföljning gjorts, hur det har gått senaste veckan, vilka problem som är på agendan och om någon ytterligare hjälp behövs. Utöver det har dessa uppföljningar använts till att ventilera eventuella idéer, förslag och kompromisser som uppkommit.

### 4.2.2 Usability-tester

Kontinuerligt under slutfasen av projektet har så kallade "Usability-tester" genomförts bland anställda hos Sigma. Ett sådant test innebär att man låter en individ utan insyn i projektet testköra applikationen. Under testkörningen ombeds individen att hela tiden berätta hur denne tänker, varför denne gör de val som görs, och hur upplevelsen är.
Utifrån den kunskapen har lärdom dragits och finslipning gjorts, allt för att användargränssnittet skall bli så smidigt att använda som möjligt.

### 4.2.3 Backlog

Från projektets början skapades enligt scrum-metoden en så kallad "Backlog" (se Appendix 6). "Backloggen" består i en prioritetslista, där varje funktionalitet som är önskvärd i applikationen är listad med ett individuellt prioritetsnummer.

Högre prioritetsnummer innebär att funktionen är viktigare än någon med lägre prioritet och därmed bör implementeras före.
Genom att arbeta på detta vis kan man behålla en hög kvalitet och fortfarande röra sig inom den tidsram som är utstakad, det enda som kan variera är antalet funktioner.

## 4.3 Arkitektur

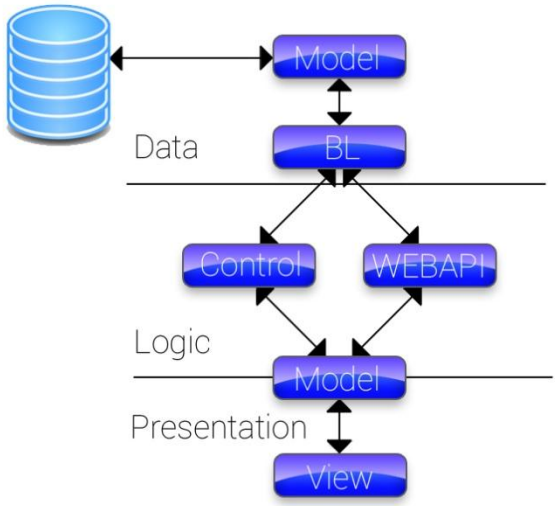
Arkitekturen är en Klient – Server modell, det vill säga att klienten frågar efter information som finns hos servern. Servern i sin tur returnerar relevant data till klienten som därefter kan behandla informationen.

Serverdelen i spelet har delats in i tre olika lager; data, logik och presentation. All kommunikation med databasen sker i datalagret, vilket innebär bibehållen kontroll över de kopplingar som finns till databasen.

Tanken har varit att göra en applikation i .NET med en sql-databas kopplat till den. Om vi dessutom har ett Web-API, med vitala funktioner som tillhandahålls i .NET MVC4 så kan om tid erbjuds även en IOS-applikation utvecklas där man gör så kallade http-post till Web-API för önskad spelfunktion.

I det logiska lagret finns de kontroller och Web-API som förser presentationslagret med data.

I presentationslagret finner man de webbsidor som presenteras för användaren, dessa sidor presenterar den data som kommer från det logiska lagret.




*Tre olika lager i arkitekturen*

Datalagret: EntityFramework, LINQ och Microsoft SQL

Logic/logik lager: JSON, JQuery, LINQ och Web-API

Presentationslagret: HTML, JSON, JQuery,CSS och RWD

# 4.4 Design

Mellan den logiska nivån och presentationslagret finner vi MVC-modellen. Det innebär att vyn och kontrollern kommunicerar med hjälp av en fördefinierad modell innehållandes data.

När en kontroller tar emot en modell görs en kontroll på att denna modell finns definierad. Modellens attribut kontrolleras innan den tillåts passera upp i datalagret. Om modellen inte är korrekt tolkas det som en felaktig operation och ett felmeddelande returneras tillbaka till vyn.

### 4.4.1 View

View eller en vy, är den aktuella webbsidan som presenteras för användaren. Webbsidan är uppbyggd helt på HTML med Razor View Engine och dekorerad med hjälp utav CSS.

### 4.4.2 VyModeller

I applikationen används ett antal vy-modeller. Detta för att på ett säkert sätt minimera insyn och otillåten modifiering i databasen.

Ett exempel:
När applikationen hämtar registrerade resultat från databasen för att kunna presentera en topplista för spelaren, skickas en förfrågan till Web-API:et som i sin tur gör en förfrågan till BusinessLayer. BusinessLayer hämtar resultat tillhörande det spelet, rad för rad ur databasen med samtliga kolumner. Därefter bygger BusinessLayer upp en ny samling resultat bestående av de kolumner som är nödvändiga för att presentera listan (namn och score), en lista innehållandes topplistmodeller.

### 4.4.3 Web-API

Web-API:et är ett verktyg för att utveckla applikationer för flera olika plattformar. Poängen med ett Web-API är att det är möjligt att konsumera oavsett vilken plattform utvecklingen sker på.

Man ställer frågor till API:et som svarar på frågan och returnerar ett resultat i form av JSON.

I projektet har Web-API:et utnyttjats så mycket som möjligt, därför läts både en webb liksom en eventuell IOS-applikation eller annan oberoende plattform konsumera APIet, för att få en så generell lösning som möjligt på spelet.

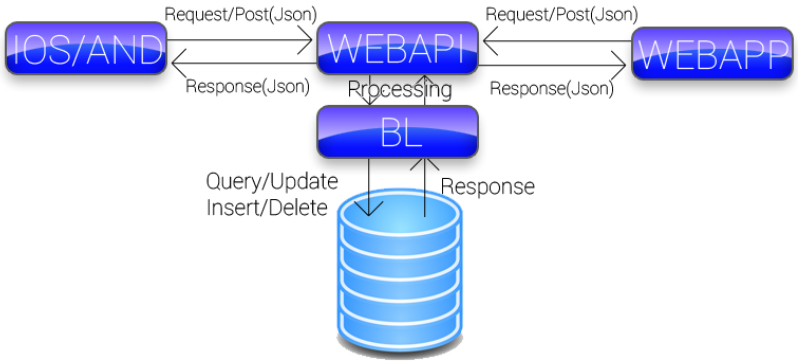



Illustration av APIet

### 4.4.4 Control

När en vy skall presenteras eller skicka data till servern sker det genom förfrågningar som först skickas till Control. Den ansvarar för att presentera den korrekta vyn för användaren och förse vyn med relevant data(Vymodell).

### 4.4.5 BusinessLayer (BL)

Applikationen använder sig av ett BusinessLayer, BL, genom detta lager sker all kommunikation med databasen. Inledningsvis i projektet användes inget BL, varje kontroller förfogade fritt över varje anslutning mot databasen. Detta behöver inte vara ett problem i sig, men för att få en ökad kontroll över de anslutningar som görs, och vilken data som plockas ut, valde vi att skapa detta lager.

Beroende på vilken plattform man spelar spelet sker kommunikationen till detta lager antingen via Control eller via Web-API:et.

### 4.4.6 Databas

I början av projektet var databasen som tidigare nämnts, primitiv. Databasen har successivt växt i takt med att nya funktioner har implementerats. Exempelvis implementerades inte Administratör förrän i slutfasen av projektet, fram till dess fanns inga tabeller för att lagra dessa, men i samband med att administratör implementerades skapades även databastabeller för detta.

Kommunikation med databasen sker via det tidigare nämnda BusinessLayer. Med hjälp av de två ramverken LINQ och Entity Framework har vi på ett smidigt sätt kunnat definiera databasen samt få tillgång till den.

Med Entity Framework kunde vi bygga databasen helt utan SQL-kod eftersom man enkelt kan definiera de tabeller man vill ha representerade med hjälp utav C#-klasser.

*Databasens struktur:*


*Databas-schema CorporateFaces (exkl Administratör), se appendix 2.0 för hela, och appendix 5.0 för SQL-kod*

**Games**

Games är tabellen där själva spelet representeras. Ett spel består utav ett Unikt ID (Guid), ett namn på spelet, en ägare och en bild som skall representera ett spel (logotyp).

Game ID genereras automatiskt när man väljer att skapa ett spel, och är nyckel för ett specifikt spel i databasen.

GamePicture (logotyp/bild) är ett valfritt alternativ.

**AdminToGames**

För att kunna skapa ett spel måste man ha registrerat sig som en administratör, när man gjort det har man ett konto i Memberships-tabellen (se appendix 5.0).

När man sedan har skapat ett spel så kopplas en admin ihop med ett spel i denna tabell.

Detta är för att man lätt skall kunna identifiera vem som är ägaren till ett specifikt spel.

**Questions**

Varje fråga i spelet är representerad i denna tabell. Här finns ett antal kolumner med data. Nycklarna Guid och qID används för att identifiera en specifik fråga och är således en supernyckel. qID är sekvensnumret på den specifika frågan i detta spel.

Utöver dessa finns de parametrar som på förhand bestämts definiera en fråga/person. Det är krav på namn, bild och kön. De övriga tre är valfria.

**Nicks**

Nicks representerar en spelare som spelat ett spel och valt att registrera sig. Varje spelare förblir unik, vilket möjliggör att spelare med samma namn inte kommer skapa konflikter i databasen.

**Scores**

Scores är en tabell med highscore för respektive spel, här lagras en spelare tillsammans med dess poäng och datum när de valt att registrera sig.

**Logs**

Logs är endast i syfte för att till denna rapport kunna skaffa sig tillräckligt med data för att kunna göra en utvärdering av spelet och ingår inte i implementationen.

Normalt kommer en spelare få välja om denne vill registrera sin poäng. I och med logs kommer dock alla spelade spel, utan användarnamn till hänsyn av integriteten, att sparas.

**Memberships**

För hanteringen av administratörer har .NETs egen kontohanterare använts, den är automatiskt genererad.

## 4.5 Flödesschema för en spelare

När en spelare blir inbjuden till ett spel presenteras denne direkt med ett inmatningsfält för namn och en möjlighet att direkt titta på den aktuella "Highscore-listan". Efter inmatat namn ska spelaren välja svårighetsgrad och därefter börja spela.

Se appendix 1.0 för flödesschema.

## 4.6 Optimering

Till en början upplevdes ett moment i spelet som långsamt, detta berodde på att det tog för lång tid att hämta frågorna från servern. Resonemanget som fördes löd: att om man vill ladda en spelomgång helt direkt och någon besitter en uppkoppling omkring 2Mbit, så vill man ej vänta längre än maximalt 2 sekunder för att ladda ett spel och för att det inte skall upplevas långsamt.

En spelsession består i 15 bilder per runda, och en maximal väntetid om 2 sekunder ger en formel för hur stor en bild får vara:

*2Mbit innebär en maximal överföringshastighet om 250kb/s.*

*Detta ger således en tämligen lättlöst ekvation*:

$$\frac{15 * Bildstorlek(kB)}{250} \leq 2$$

*Bildstorleken måste således vara:*

$$Bildstorlek(kB) \leq \frac{2 * 250}{15}$$

Detta ger en maximal bildstorlek om 33 kB, vilket är en relativt liten storlek i sammanhanget bilder. Formatet PNG gav vid komprimeringen storlekar omkring ~80 kB, vilket var långt över budget, medans JPG gav bilder som endast var ca 5 kB stora. Därför har JPG valts som komprimering.

# 5. Resultat

I detta kapitel kommer resultatet presenteras utifrån de mätningar som gjorts.
Resultatet presenteras i siffror och tabeller, allt för att få en så tydlig bild av utfallet som möjligt.

I vårt urval finns personal representerad från olika avdelningar i verksamheten samt nyanställda. De berörda personerna har fått ett häfte med bilder på anställda, och besvarat dem enskilt.

Mätserie 1 av CorporateFaces före spel:

| Individ | #1 | #2 | #3 | #4 | Medelvärde |
|---|---|---|---|---|---|
| Antal personer identifierade | 16/108 | 8/108 | 40/108 | 19/108 | 20,75/108 |

Individerna har besvarat 45 frågor på svårighetsgraderna *Easy* och *Medium* (se kap 3.3, Svårighetsgrader ), (svårighet *Hard* ej närvarande ty ej fullständigt implementerad).
Normalt sett kan man välja att spela så många frågor man själv vill, i vår undersökning så krävde vi 90 spelade frågor.

När ovan genomförts gjordes en uppföljning för att kunna validera eller falsifiera om hypotesen stämmer, att detta verkligen kan vara ett verktyg för att öka kännedomen om varandra inom organisationen.
Testpersonen gavs återigen exakt samma häfte som metoden i kap 2 förespråkar.

Mätserie 2 av Corporatefaces efter spel:

| Individ | #1 | #2 | #3 | #4 | Medelvärde | Diff |
|---|---|---|---|---|---|---|
| Antal personer Identifierade | 24/108 | 16/108 | 57/108 | 28/108 | 31,25/108 | +10,5 |

OBS! Samma nummer i denna mätserie som i mätserie 1 innebär att det är samma individ.

# 6. Slutsats

## 6.1 Resumé

Första delen i projektet gick ut på att forma det spel som skulle ligga till grund för vår undersökning. Spelets grafiska utformning skulle få spelaren att så enkelt som möjligt förstå vad spelet går ut på, utan att behöva tänka.

Andra delen i projektet ägnades åt implementationen av själva spelet, dvs. design av databasen, webbdesign, lösningar för framtida mobilapplikationer och så självklart att implementera webbversionen av spelet. Med hjälp av WebAPI:et och JQuery byggdes spelet undan för undan upp samtidigt som MVC-designmönstret agerade för resterande delar utav webbplatsen.

Slutfasen i projektet innebar verifiering av spelet, om det fanns några effekter utav det och testning utav den grafiska utformningen.

## 6.2 Kritisk diskussion

Precis som metoden förespråkar har mätningar gjorts på hur många individer den berörda personen i fråga känner igen.
Baskunskaperna är förstås varierande, en del känner igen väldigt många av sina kollegor, andra ganska få.
Det beror förmodligen på var i organisationen man arbetar, vad ens arbetssysslor består i och hur länge man har verkat inom den.

- Slutsatsen man kan dra från den första undersökningen är att de fyra berörda i genomsnitt känner igen ca 19,2% av de anställda.
- Slutsatsen man kan dra från den andra undersökningen är att de fyra berörda i genomsnitt känner igen ca 28,9% av de anställda.

Spelet CorporateFaces tycks lyckas väl och når det uppsatta målet, vilket är att öka personalens kännedom om varandra. I och med att det finns möjlighet att bygga ut spelet med de dynamiska frågorna (svårighetsgraden *Hard*), finns potential att kunna öka kunskaper på fler plan än bara igenkänningen.

Det är en tidskrävande process att validera om spelet verkligen fungerar, rent statistiskt, vilket har resulterat i en låg frekvens av mätningar. Pga. detta kan denna rapport ej verifiera huruvida de nuvarande mätningarna stämmer överens med en mätning som skulle kunna göras över en större testgrupp, men hittills har inga mätningar där igenkänningen ökat med mindre än

42,5% påvisats. Inte heller har mätningarna tagit hänsyn till tidsaspekten, det vill säga hur lång tid som förflutit från det att spelet spelats till det att försökspersonen har fått genomgå mätningen igen.

## 6.3 Generaliseringar

Eftersom spelet ej är knutet till ett specifikt företag, mer än att det är Sigma som står bakom det, finns möjligheten för fler företag och organisationer att nyttja spelet.

## 6.4 Fortsatt forskning

Vidare arbete och insamling av registrerade resultat, intervjuer och utveckling av applikationen skulle kunna bevisa hypotesen i syftet, då mätningarna i denna rapport ej har tillräckligt stor mängd mätdata.

# Referenser


DATA, R. 1999-2012. *HTML Introduction* [Online]. Available: http://www.w3schools.com/html/html_intro.asp.

DAVJOH 2010. ModelViewControllerDiagram. *In:* MODELVIEWCONTROLLERDIAGRAM2.SVG (ed.). http://commons.wikimedia.org/wiki/File:ModelViewControllerDiagram.svg.

JQUERY.ORG. *Browser Compatibility* [Online]. Available: http://docs.jquery.com/Browser_Compatibility [Accessed 23-May 2012 2012].

KORPI, J. *Lessframework* [Online]. Available: Lessframework.com [Accessed 23-May 2012.

MICROSOFT. 2012. *ASP.NET MVC 4 Beta* [Online]. Available: http://www.asp.net/mvc/mvc4.

MSDN, M. 2012. *Lambda Expressions (C# Programming Guide)* [Online]. Available: http://msdn.microsoft.com/en-us/library/bb397687.aspx [Accessed 23-May 2012.

NAYYERI, S. C. A. K. 2009. -Introduction, Beginning ASP.NET MVC 1.0.

PAHARIA, S. H. A. S. 2011. .NET 4 for Enterprise Architects and Developers. *.NET 4 for Enterprise Architects and Developers.*

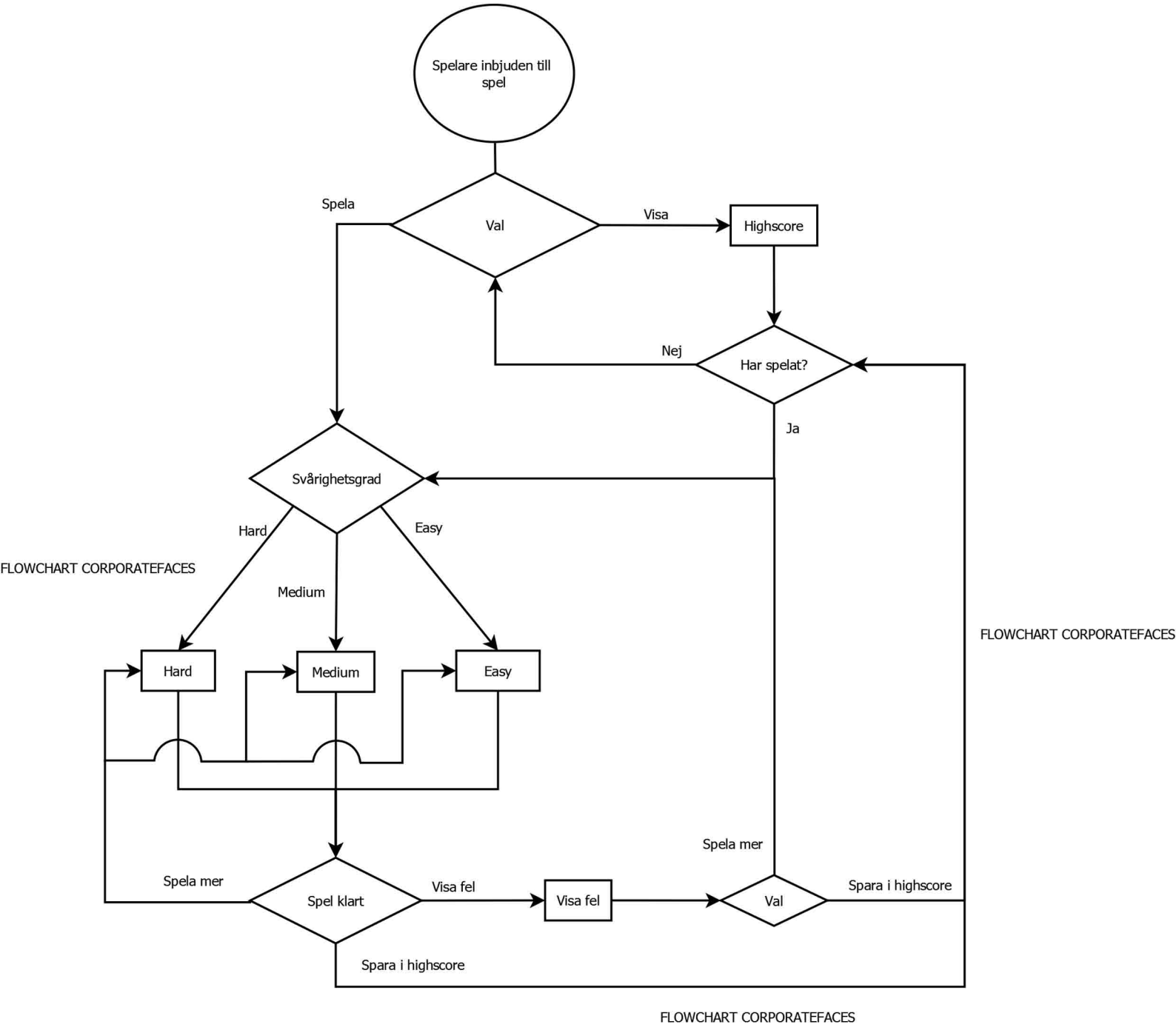
SCOTTGU. July 02, 2010 11:01 2010. Introducing "Razor" – a new view engine for ASP.NET. Available from: http://weblogs.asp.net/scottgu/archive/2010/07/02/introducing-razor.aspx 2012].

SOCIETY, T. I. 2006. The application/json Media Type for JavaScript Object Notation (JSON). *The application/json Media Type for JavaScript Object Notation (JSON).* http://tools.ietf.org: The Internet Society

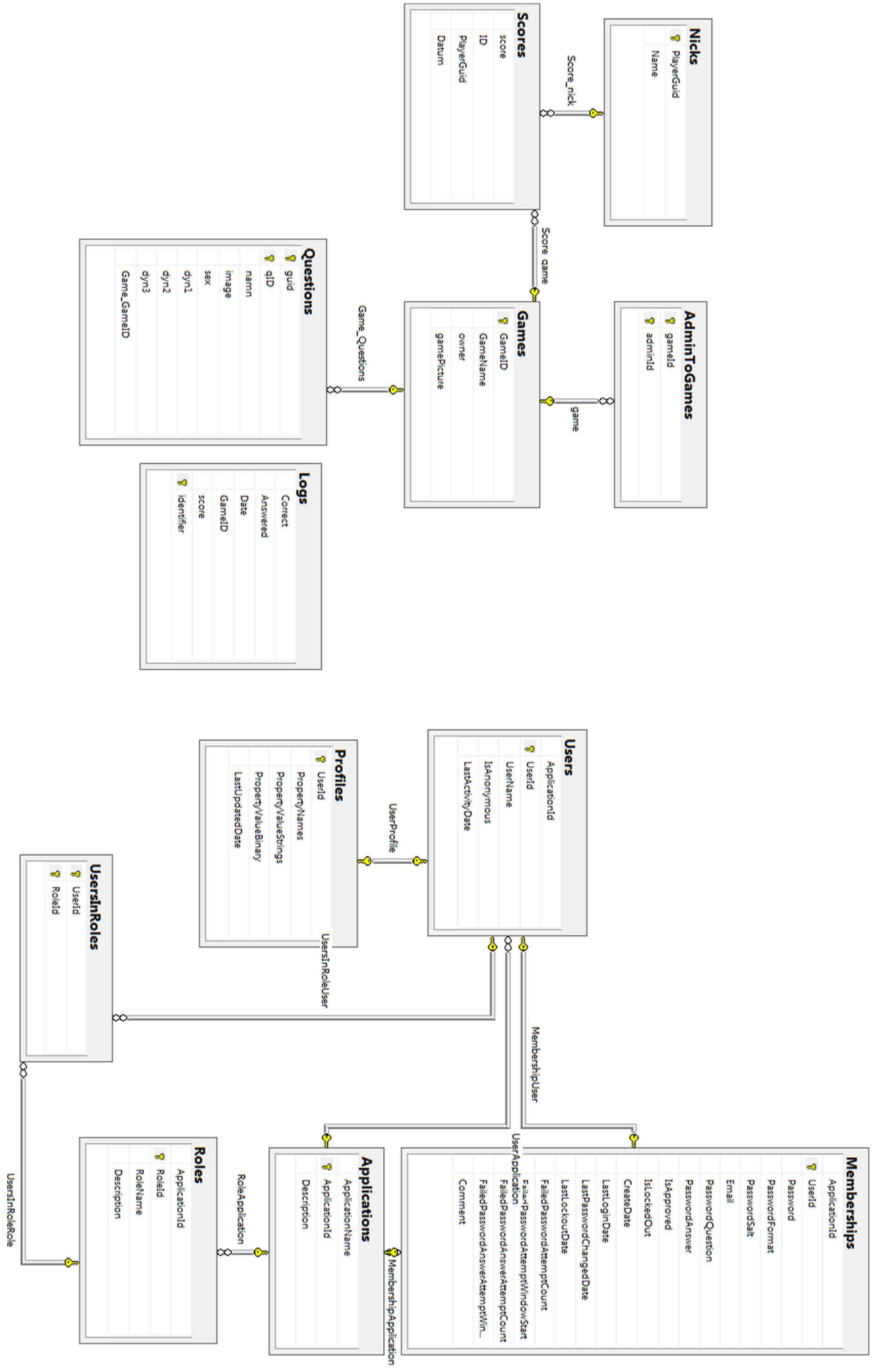
W3C ( MIT , E., KEIO) 2012. *What is CSS?* [Online]. w3c. Available: http://www.w3.org/standards/webdesign/htmlcss#whatcss [Accessed May-23 2012.

WIKIPEDIA. 2012. *Ajax* [Online]. Available: http://sv.wikipedia.org/wiki/AJAX.

# Appendix 1.0 Flödesschema



# Appendix 2.0 Fullständig Databas



# Appendix 3.0 Use Cases

## 1. Use Cases för Administratör:

### 1.1 Registrering

Registreringen för att bli administratör och kunna skapa ett spel görs via en länk på startsidan.
Vid registreringen måste man ange ett antal vitala parametrar:

- användarnamn
- mailadress
- lösenord

### 1.2 Skapa spel

Aministratörens roll / spelarens roll
Administratören registrerar sig på sajten och presenteras med ett UI där det är möjligt att lägga till personer, med bild, i databasen kopplat till administratörens spel. Administratören delar sedan med sig spel-id till sina kollegor så att de kan gå in och spela.

### 1.3 Skapa fråga

Administratören har när som helst möjlighet att gå in och lägga till en fråga i spelet, fördelaktligen skapas frågorna innan man bjuder in till spel, detta eftersom att ett större urval av frågor för spelet gör att det minskar risken att samma fråga kan upprepa sig.

Vid skapandet utav en fråga får man mata in följande:

- Namn
- Bild
- Kön
- dynamiskt
- dynamisk
- dynamisk

### 1.4 Radera och uppdatera fråga och spel

På administratörens begäran kan spelet och dess frågor när som helst raderas via hemsidan. Det finns även möjlighet uppdatera en befintlig fråga. Det görs helt sonika genom att denne matar in namn och bild precis som vanligt, spelet känner då av om denna fråga redan finns och uppdatera då bara den.

## 2. Use Case för användare:

### 2.1 Spelare inbjuden till spel

En spelare inbjuds till spel där spelaren klickar på den givna länken, länken medför att spelaren gör en “get” från servern via en metod i indexcontrollern. Spel-id agerar parameter i form av en vymodell till controllern (se 5.3). Controllern anropar buisnesslayer som kontrollerar att spelet existerar i databasen genom att göra en förfrågan mot databasen enligt exemplet i beskrivningen av “Entity Framework” (se 4.1.13). Om spelet existerar presenteras spelaren med en vy där denne ombeds fylla i sitt namn, vid klick på knappen “Play” postas namn och spel-id som vymodell till controllern. Controllern presenterar spelaren med en vy där spelaren ombeds att välja svårighetsgrad.

### 2.2 Val av svårighetsgrad

Spelaren har presenterats med en vy där spelaren ombeds att välja svårighetsgrad. Vid klick på vald svårighetsgrad, “Easy” eller “Medium”, görs en “get” mot controllern med spel-id och namn som parameter i form av en vymodell. Controllern kontrollerar att namn och spel-id har en giltig form och returnerar spelvyn till spelaren.

### 2.3 Spelvyn

Vid laddning utav sidan skickas en förfrågan till servern om frågor för det det aktuella spelet med detta ID, förfrågningen görs med hjälp av en Jquery(se 4.1.3) Ajax(se kap:4.1.4) mot APIet (se 5.3.3).
APIet i sin tur returnerar en Json array (se 4.1.2) bestående av den nödvändiga informationen för att bygga upp en spel session.

Bilder till spelet är lagrade i databasen (se 5.3.6) som byte-arrayer kommer APIet se till att göra om dessa byte-fält till strängar, som enkelt kan skickas som Json. Det är önskvärt att hålla nere mängden information som skickas från servern till klienten, så har det legat ett stort fokus på att hålla nere storleken(i byte, inte upplösning) på bilderna. Detta har gjorts genom att komprimera bilderna hårt, och skala ner dem till en låg upplösning som är

gemensam för samtliga bilder i spelet.

## 2.4 Spela mer

Om spelaren valt “Spela mer” så byggs vyn om med hjälp utav JQuery och en ny förfrågan om frågor skickas till servern och spelvyn presenteras för spelaren, dock med skillnaden att jsonArrayen nu fylls på med de hämtade frågorna, dvs om spelaren innan hade besvarat 15 frågor så kommer denne nu att besvara fråga 16 av 30.

## 2.5 Registrera resultat

Om spelaren valt “Registrera resultat” så postas ett svarsfält med de valda svaren under spelomgången till servern(APIet) för rättning samt spel-id, namn och svårighetsgrad. Spelarens namn och resultat förs in i databasen och spelaren presenteras med en ny vy som visar de tio bästa resultaten för spelet med samma spel-id.

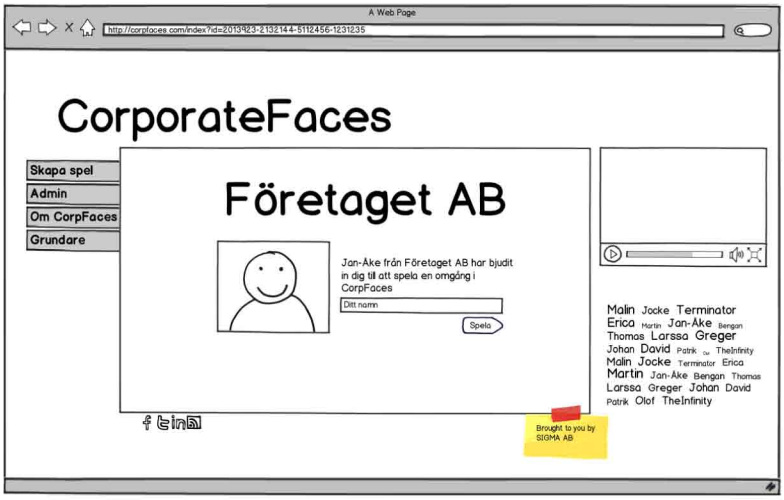
## 2.6 Visa fel

Om spelaren valt “Visa fel” så renderas sidan om (notera ej laddas om), med hjälp av JQuery, som sätter in lämpliga HTML- element med lämpliga fördefinerade CSSer. Kontentan av detta blir en rutnäts representation med alla felaktiga svarsalternativ spelaren haft under spelets gång, med en minimerad bild, rätt svar och det felaktiga svaret.

I denna vyn finns möjligheten att registrera resultat (se ovan ,app. 2.5) och börja om , det vill säga gå till “välj nivå” (se ovan, app.2.2).

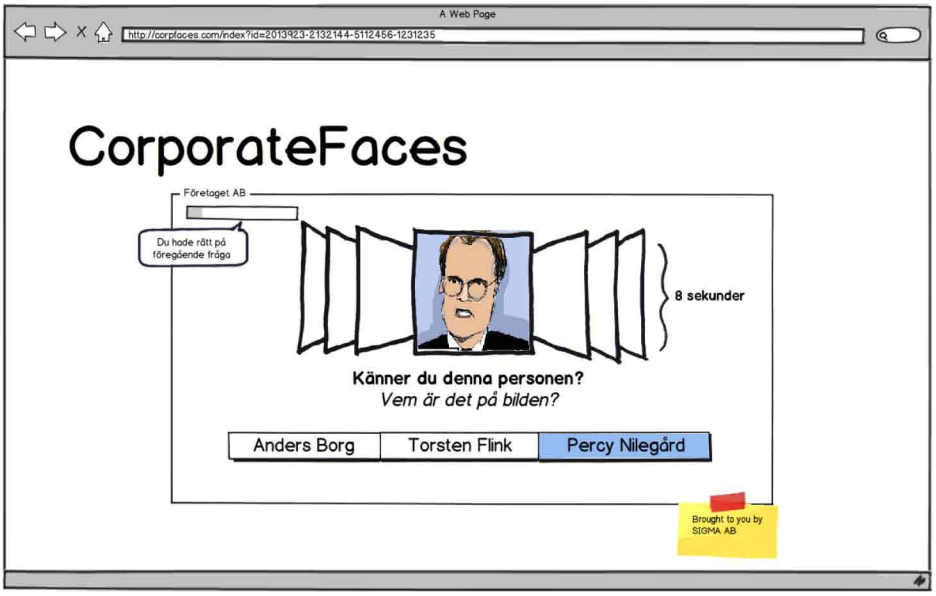
# Appendix 4.0 Mock-up

## Startsida

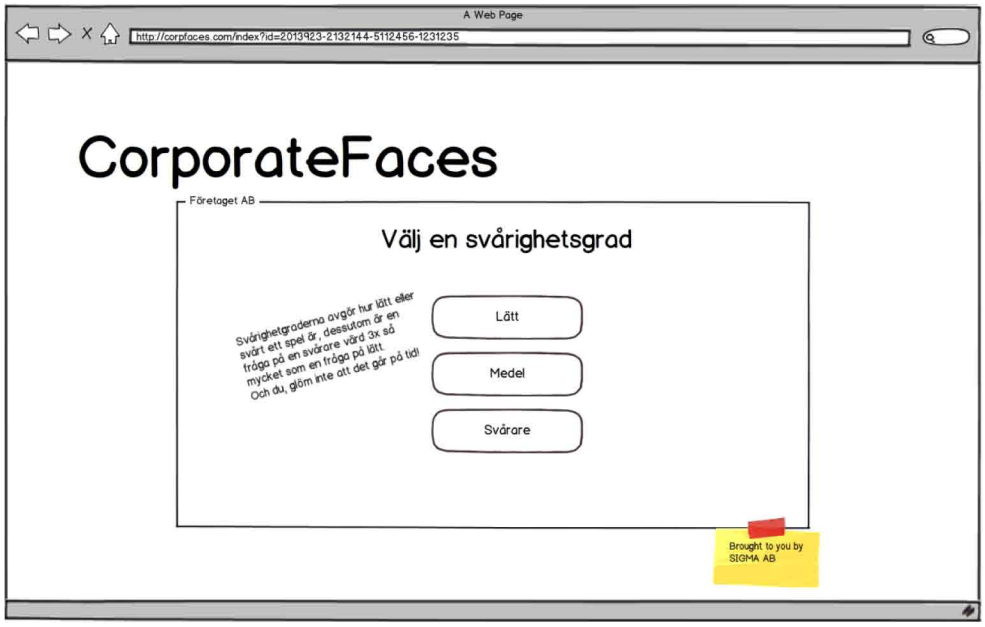




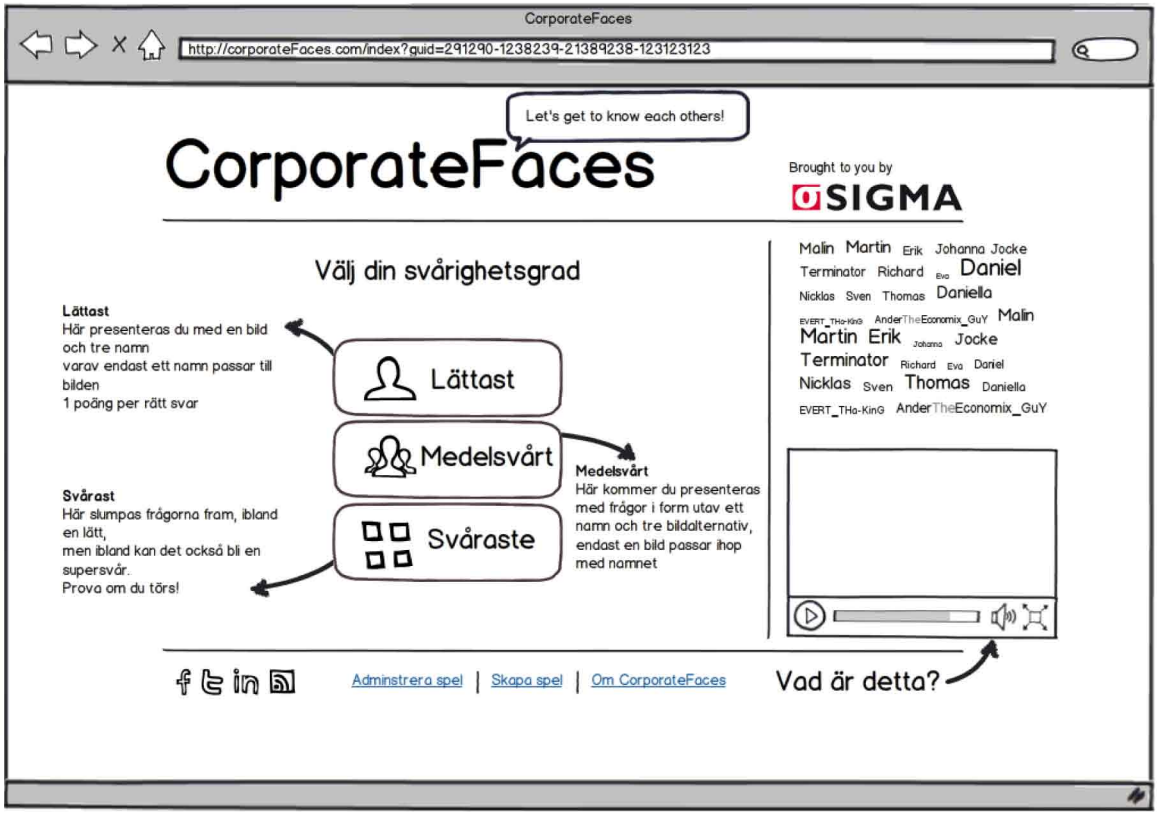
## Medium svårighetsgrad



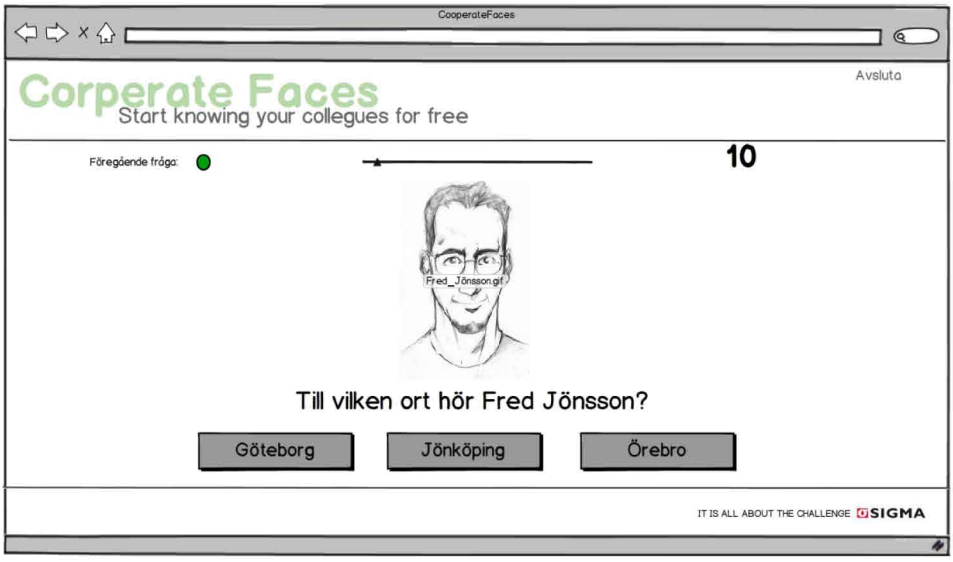

# Val av svårighetsgrad

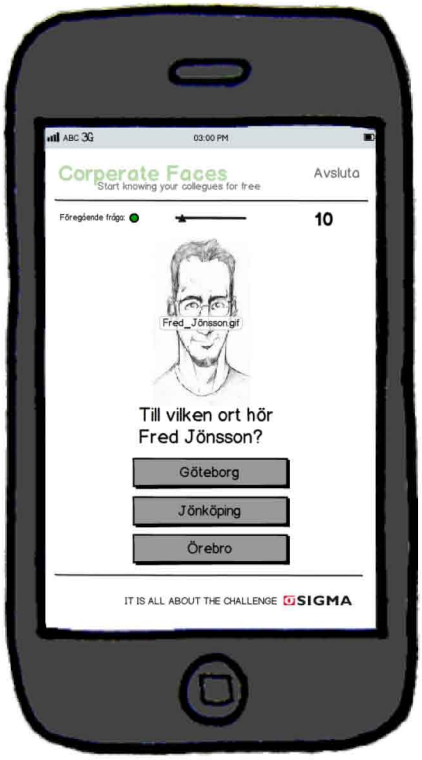




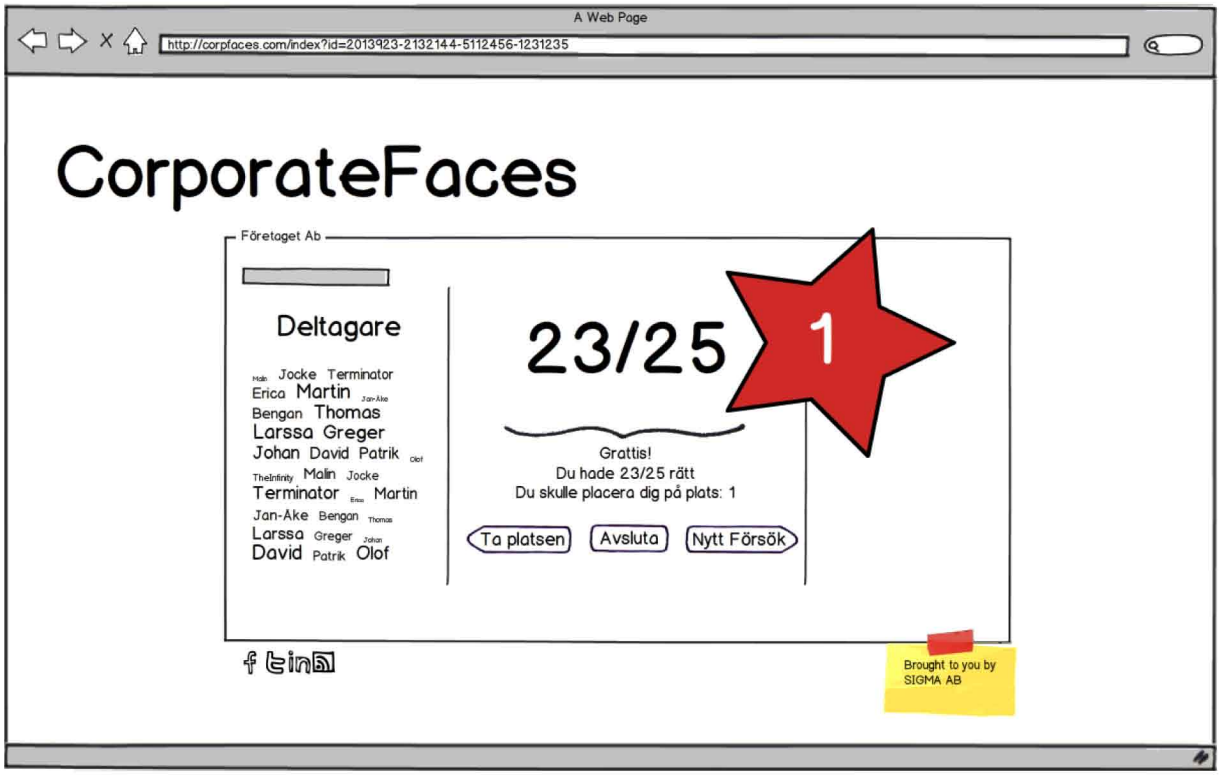


# Svårighetsgrad Easy




# Omgång slut



# Apendix 5.0 Databasen

## AdminToGames

```
USE [CoporateFaces]
GO

/****** Object:  Table [dbo].[AdminToGames]    Script Date: 06/01/2012
10:52:34 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[AdminToGames](
           [gameId] [uniqueidentifier] NOT NULL,
           [adminId] [uniqueidentifier] NOT NULL,
 CONSTRAINT [PK_AdminToGames] PRIMARY KEY CLUSTERED
(
           [gameId] ASC,
           [adminId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[AdminToGames]  WITH CHECK ADD  CONSTRAINT [game] FOREIGN
KEY([gameId])
REFERENCES [dbo].[Games] ([GameID])
GO

ALTER TABLE [dbo].[AdminToGames] CHECK CONSTRAINT [game]
GO
```

## Games

```
USE [CoporateFaces]
GO

/****** Object:  Table [dbo].[Games]    Script Date: 06/01/2012 10:53:17
******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
```

```
GO

CREATE TABLE [dbo].[Games](
            [GameID] [uniqueidentifier] NOT NULL,
            [GameName] [nvarchar](4000) NOT NULL,
            [owner] [nvarchar](4000) NOT NULL,
            [gamePicture] [varbinary](max) NULL,
 CONSTRAINT [PK_Games] PRIMARY KEY CLUSTERED
(
            [GameID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO
```

## Logs

```
USE [CoporateFaces]
GO

/****** Object:  Table [dbo].[Logs]    Script Date: 06/01/2012 10:53:34
******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Logs](
            [Correct] [int] NOT NULL,
            [Answered] [int] NOT NULL,
            [Date] [datetime] NOT NULL,
            [GameID] [uniqueidentifier] NOT NULL,
            [score] [int] NOT NULL,
            [identifier] [uniqueidentifier] NOT NULL,
 CONSTRAINT [PK_Logs] PRIMARY KEY CLUSTERED
(
            [identifier] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
```

## Nicks

```
USE [CoporateFaces]
```

```
GO

/****** Object:  Table [dbo].[Nicks]    Script Date: 06/01/2012 10:53:56
******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Nicks](
           [PlayerGuid] [uniqueidentifier] NOT NULL,
           [Name] [nvarchar](4000) NULL,
 CONSTRAINT [PK_Nicks] PRIMARY KEY CLUSTERED
(
           [PlayerGuid] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
```

## Questions

```
USE [CoporateFaces]
GO

/****** Object:  Table [dbo].[Questions]    Script Date: 06/01/2012 10:54:11
******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[Questions](
           [guid] [uniqueidentifier] NOT NULL,
           [qID] [int] NOT NULL,
           [namn] [nvarchar](50) NOT NULL,
           [image] [varbinary](max) NOT NULL,
           [sex] [bit] NOT NULL,
           [dyn1] [nvarchar](50) NULL,
           [dyn2] [nvarchar](50) NULL,
           [dyn3] [nvarchar](50) NULL,
           [Game_GameID] [uniqueidentifier] NULL,
 CONSTRAINT [PK_Questions] PRIMARY KEY CLUSTERED
(
           [guid] ASC,
           [qID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO

SET ANSI_PADDING OFF
GO

ALTER TABLE [dbo].[Questions]  WITH CHECK ADD  CONSTRAINT [Game_Questions]
FOREIGN KEY([Game_GameID])
REFERENCES [dbo].[Games] ([GameID])
GO

ALTER TABLE [dbo].[Questions] CHECK CONSTRAINT [Game_Questions]
GO
```

## Nicks

```
USE [CoporateFaces]
GO

/****** Object:  Table [dbo].[Nicks]    Script Date: 06/01/2012 11:00:12
******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Nicks](
           [PlayerGuid] [uniqueidentifier] NOT NULL,
           [Name] [nvarchar](4000) NULL,
 CONSTRAINT [PK_Nicks] PRIMARY KEY CLUSTERED
(
           [PlayerGuid] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
```

## Scores

```
USE [CoporateFaces]
GO

/****** Object:  Table [dbo].[Scores]    Script Date: 06/01/2012 10:54:26
******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
```

```
CREATE TABLE [dbo].[Scores](
           [score] [int] NOT NULL,
           [ID] [uniqueidentifier] NOT NULL,
           [PlayerGuid] [uniqueidentifier] NOT NULL,
           [Datum] [datetime] NOT NULL
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[Scores]  WITH CHECK ADD  CONSTRAINT [Score_game] FOREIGN
KEY([ID])
REFERENCES [dbo].[Games] ([GameID])
GO

ALTER TABLE [dbo].[Scores] CHECK CONSTRAINT [Score_game]
GO

ALTER TABLE [dbo].[Scores]  WITH CHECK ADD  CONSTRAINT [Score_nick] FOREIGN
KEY([PlayerGuid])
REFERENCES [dbo].[Nicks] ([PlayerGuid])
GO

ALTER TABLE [dbo].[Scores] CHECK CONSTRAINT [Score_nick]
GO
```

## Membership (Microsofts)

```
USE [CoporateFaces]
GO

/****** Object:  Table [dbo].[Memberships]    Script Date: 06/01/2012 10:54:49
******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Memberships](
           [ApplicationId] [uniqueidentifier] NOT NULL,
           [UserId] [uniqueidentifier] NOT NULL,
           [Password] [nvarchar](128) NOT NULL,
           [PasswordFormat] [int] NOT NULL,
           [PasswordSalt] [nvarchar](128) NOT NULL,
           [Email] [nvarchar](256) NULL,
           [PasswordQuestion] [nvarchar](256) NULL,
           [PasswordAnswer] [nvarchar](128) NULL,
           [IsApproved] [bit] NOT NULL,
           [IsLockedOut] [bit] NOT NULL,
           [CreateDate] [datetime] NOT NULL,
           [LastLoginDate] [datetime] NOT NULL,
           [LastPasswordChangedDate] [datetime] NOT NULL,
           [LastLockoutDate] [datetime] NOT NULL,
           [FailedPasswordAttemptCount] [int] NOT NULL,
```

```
            [FailedPasswordAttemptWindowStart] [datetime] NOT NULL,
            [FailedPasswordAnswerAttemptCount] [int] NOT NULL,
            [FailedPasswordAnswerAttemptWindowsStart] [datetime] NOT NULL,
            [Comment] [nvarchar](256) NULL,
PRIMARY KEY CLUSTERED
(
            [UserId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[Memberships]  WITH CHECK ADD  CONSTRAINT
[MembershipApplication] FOREIGN KEY([ApplicationId])
REFERENCES [dbo].[Applications] ([ApplicationId])
GO

ALTER TABLE [dbo].[Memberships] CHECK CONSTRAINT [MembershipApplication]
GO

ALTER TABLE [dbo].[Memberships]  WITH CHECK ADD  CONSTRAINT [MembershipUser]
FOREIGN KEY([UserId])
REFERENCES [dbo].[Users] ([UserId])
GO

ALTER TABLE [dbo].[Memberships] CHECK CONSTRAINT [MembershipUser]
GO
```

## USERS (Microsofts)

```
USE [CoporateFaces]
GO

/****** Object:  Table [dbo].[Users]    Script Date: 06/01/2012 10:55:06
******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Users](
            [ApplicationId] [uniqueidentifier] NOT NULL,
            [UserId] [uniqueidentifier] NOT NULL,
            [UserName] [nvarchar](50) NOT NULL,
            [IsAnonymous] [bit] NOT NULL,
            [LastActivityDate] [datetime] NOT NULL,
PRIMARY KEY CLUSTERED
(
            [UserId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO

ALTER TABLE [dbo].[Users]  WITH CHECK ADD  CONSTRAINT [UserApplication]
FOREIGN KEY([ApplicationId])
REFERENCES [dbo].[Applications] ([ApplicationId])
GO

ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [UserApplication]
GO
```

## UsersInRoles (Microsofts)

```
USE [CoporateFaces]
GO

/****** Object:  Table [dbo].[UsersInRoles]    Script Date: 06/01/2012
10:55:21 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[UsersInRoles](
           [UserId] [uniqueidentifier] NOT NULL,
           [RoleId] [uniqueidentifier] NOT NULL,
PRIMARY KEY CLUSTERED
(
           [UserId] ASC,
           [RoleId] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[UsersInRoles]  WITH CHECK ADD  CONSTRAINT [UsersInRoleRole]
FOREIGN KEY([RoleId])
REFERENCES [dbo].[Roles] ([RoleId])
GO

ALTER TABLE [dbo].[UsersInRoles] CHECK CONSTRAINT [UsersInRoleRole]
GO

ALTER TABLE [dbo].[UsersInRoles]  WITH CHECK ADD  CONSTRAINT [UsersInRoleUser]
FOREIGN KEY([UserId])
REFERENCES [dbo].[Users] ([UserId])
GO

ALTER TABLE [dbo].[UsersInRoles] CHECK CONSTRAINT [UsersInRoleUser]
GO
```

# Appendix 6.0 Backlog

**Project:** Sigma-CorporateFaces **Server:** origo.sigma.se\Vast-Inhouse **Query:** Iteration Backlog **List type:** Tree

| ID | Work Item Type | Title 1 | Title 2 | Assigned To | Stack R | Activity |
|---|---|---|---|---|---|---|
| 822 | User Story | User enters login page | | Ext Joakim Jonsson | 80 | |
| 823 | Task | | user wants to login | Ext Joakim Jonsson | 100 | Development |
| 816 | User Story | User want to see highscore | | Ext Joakim Jonsson | 90 | |
| 819 | Task | | show highscore | Ext Joakim Jonsson | 95 | Development |
| 818 | Task | | create Highscore page | Ext Joakim Jonsson | 100 | Design |
| 806 | User Story | User want to or has finished game | | Ext Joakim Jonsson | 97 | |
| 813 | Task | | If user is qualified for highscore then t | Ext Martin Bergqvist | 90 | Development |
| 814 | Task | | Play Again | Ext Joakim Jonsson | 92 | Development |
| 812 | Task | | End Game | Ext Joakim Jonsson | 95 | Development |
| 808 | Task | | Game result | Ext Joakim Jonsson | 99 | Development |
| 807 | Task | | Create the finishgame page | Ext Joakim Jonsson | 100 | Design |
| 796 | User Story | Create an own game | | Ext Joakim Jonsson | 98 | |
| 803 | Task | | Link to admin page | Ext Joakim Jonsson | 97 | Development |
| 815 | Task | | Disclamer | Ext Joakim Jonsson | 98 | |
| 800 | Task | | Database logic to add admindata to da | Ext Joakim Jonsson | 99 | Development |
| 797 | Task | | Create Admin Page | Ext Joakim Jonsson | 100 | |
| 791 | User Story | User want information | | Ext Martin Bergqvist | 99 | |
| 795 | Task | | Create a video that describes the gam | Ext Martin Bergqvist | 97 | |
| 794 | Task | | Create who is behind page | Ext Martin Bergqvist | 98 | |
| 793 | Task | | Link to information page from MainPa | Ext Martin Bergqvist | 99 | |
| 792 | Task | | Create the information Page | Ext Martin Bergqvist | 100 | Design |
| 786 | User Story | User enters homepage | | Ext Joakim Jonsson | 100 | |
| 821 | Task | | Log in | Ext Joakim Jonsson | 85 | Development |
| 817 | Task | | GameId Entering | Ext Joakim Jonsson | 97 | Development |
| 801 | Task | | create Link to admin page | Ext Joakim Jonsson | 98 | Development |
| 787 | Task | | Design the interface | Ext Joakim Jonsson | 100 | Design |
| 790 | Task | | Create Game | Ext Joakim Jonsson | 101 | |
| 798 | User Story | User wants to play the game | | Ext Martin Bergqvist | 101 | |
| 820 | Task | | Dynamic Questions in HardMode | Ext Martin Bergqvist | 69 | |
| 810 | Task | | User is able to select difficulty level | Ext Martin Bergqvist | 70 | Development |
| 837 | Task | | Create select gametype page | Ext Martin Bergqvist | 71 | Design |
| 811 | Task | | select and Correct the answer before | Ext Martin Bergqvist | 94 | Development |
| 809 | Task | | Create player name/start Game | Ext Martin Bergqvist | 95 | Development |
| 805 | Task | | Create game logic | Ext Martin Bergqvist | 98 | Development |
| 804 | Task | | Create in game page | Ext Martin Bergqvist | 99 | Design |
| 799 | Task | | Create database logic inorder to get th | Ext Martin Bergqvist | 100 | |
| 802 | Task | | Create the startpage of a game | Ext Martin Bergqvist | 101 | Design |