CHALMERS | GÖTEBORGS UNIVERSITET

Pluginramverk för webbaserade spel

Kandidatarbete inom Data- och informationsteknik

Andreas Svanström
John Johansson
Magnus Larsson
Mattias Philip

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2013

Pluginramverk för webbaserade spel

Andreas Svanström
John Johansson
Magnus Larsson
Mattias Philip

Examiner: Arne Linde

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

Abstract

In order to attract talented consultants, Sigma, a consultant company, wanted an online game for recruitment. In the game, potential employees' skills could be tested to determine whether they are qualified for the job.

This paper presents a general technique for creating such a recruitment game. The report generalizes the recruitment game to a level-based game for the web, where a player should be able to progress from one level to the next by solving a given problem. A generalized framework and a prototype of the game have been created. The prototype, which is built upon the framework, acts as a proof-of-concept game and proves that the concept works.

Sammanfattning

För att locka duktiga konsulter ville företaget Sigma ha ett rekryteringsspel, där potentiella anställdas kunskaper kunde testas. Detta för att utvärdera om personerna i fråga var kvalificerade för tjänsten.

Den här rapporten presenterar en generell teknik för att skapa ett sådant rekryteringsspel. Rapporten generaliserar rekryteringsspel till nivåbaserade spel för webben, där en spelare ska kunna ta sig från en nivå till nästa genom att lösa ett problem. Ett generaliserat ramverk samt en prototyp av rekryteringsspelet har skapats. Spelprototypen kodades i ramverket och bevisar att konceptet fungerar.

Begreppslista

Ajax - En webbteknik för att hämta data i bakgrunden med Javascript.

API - Application Programming Interface, möjliggör kommunikation mellan olika programvaror.

Asp - Active Server Pages, är ett serverskriptspråk för dynamisk generering av webbinnehåll. Det vill säga att servern genererar html utifrån ett eget skriptspråk på förfrågan.

Asp.net - Ett koncept för dynamiskt webbinnehåll, en webbanpassning av .Net-plattformen, vilket bland annat innebär att serverprogrammet kan skrivas i valfritt språk som stöds av .Net-plattformen.

Azure - Windowsbaserad molntjänst för webbapplikationer.

C# - Objektorienterat programspråk för .Net.

CMS - Content Management System, innehållshanteringssystem, ett mjukvarusystem som underlättar skapande och redigerande av webbinnehåll, till exempel genom att tillhandahålla ett användargränssnitt som liknar en ordbehandlare.

Entitet - Term inom databaser för ett objekt med varaktig identitet, till exempel en person eller händelse.

Javascript - Ett skriptspråk som används främst på klientsidan i webbapplikationer, det vill säga exekveras i en webbläsares Javascript-motor.

Json - Ett kompakt, textbaserat format för att överföra objekt på webben.

Molntjänst - En virtuell servertjänst. Har som fördel mot en traditionell webbserver att systemresurser tillfälligt kan ökas eller minskas efter behov.

Plugin - Program som inte kan köras fristående, utan som laddas in som en del i ett annat program och tillför nya egenskaper eller ny funktionalitet. Kallas även insticksprogram.

PMS - Plugin Management System, Pluginhanteringssystem, ett mjukvarusystem som underlättar att skapa, redigera och sammankoppla plugin som i sin tur bygger upp en applikation eller tjänst.

Ramverk - En uppsättning regler för hur något ska gå till. Inom programmering anger reglerna hur kod ska skrivas och ett ramverk kan då även inkludera en samling komponenter.

SVN - Förkortning för Apache Subversion, ett versionshanteringssystem.

Virtuell maskin - en mjukvarubaserad abstraktion av en dator. Att skriva program för en virtuell maskin innebär att programmet kan köras med exakt samma kod på alla datorer som har den virtuella maskinen installerad.

Webbapplikation - Programvara där användargränssnittet körs i en webbläsare.

Innehållsförteckning

1 Inledning

1.1 Bakgrund

Webbutveckling är idag en mycket använd teknik. Sedan Internet kommersialiserades i mitten av 1990-talet har många företag, offentliga verksamheter och även privatpersoner flyttat ut delar av sin verksamhet på Internet (Internet world stats, 2013). Ju fler aktörer som ansluter sig till Internet, desto större blir vikten av att använda robusta, snabba och effektiva webbtekniker. Robusta så att webbservern inte går ner, snabba så att användaren inte känner av fördröjningar och effektiva så att de inte tar upp onödigt mycket systemresurser eller bandbredd.

Webben har växt från att till en början bara innehålla statiska informationssidor till att idag tillhandahålla interaktiva webbtjänster av många slag. Idag finns det tjänster på webben där man kan se på film och lyssna på musik. Sociala nätverk där människor samlas och umgås är också mycket vanligt. Många företag har även börjat flytta över delar av sin rekryteringsprocess till webben, där de både testar och lockar programmeringstalanger. Ett sådant företag är Sigma.

1.1.1 Sigmas grundkoncept

Sigma är ett It-företag i konsultbranschen som har haft tankar om att skapa ett kunskapstestande rekryteringsverktyg för webben. Avsikten med verktyget är att användaren måste visa tillräckliga färdigheter och kunskaper för att bli kallad till en anställningsintervju. Sigmas idé är att utforma verktyget som ett nivåbaserat spel där varje nivå innebär nya utmaningar för användaren.

1.1.2 Generaliserat koncept

Många dataspel är uppbyggda av olika nivåer, till exempel klassiska plattformsspel som Super Mario eller Sonic. I dessa spel tar spelaren sig igenom en rad utmaningar för att ta sig till nästa nivå tills dess att nivåerna tar slut och spelet är avklarat. Sigmas idé kan ses som något liknande, att en användare, eller spelare, skulle ta sig igenom en rad utmaningar och till sist bli kallad till en anställningsintervju.

Likheten mellan Sigmas idé och klassiska spel gav upphov till tanken att generalisera ett nivåbaserat ramverk. Detta

ramverk skulle då sköta hantering och hämtning av logik och grafik till de olika nivåerna. Sigma skulle därefter kunna använda ramverket för att realisera anställningsspelet.

1.2 Mål

Målet med projektet är att ta fram ett ramverk för nivåbaserade webbspel. Ramverket ska fungera genom en presentationssida som presenterar nivåerna. Nivåerna ska i sin tur vara uppbyggda som fristående enheter. Dessa ska kunna laddas in i presentationssidan, och innehålla både logik och grafik för att presentera en nivå. Det ska endast vara möjligt att visa en nivå i taget. Det ska även vara möjligt för Sigma att använda sig av ramverket för att utforma ett rekryteringsspel.

1.3 Metod

Genom en agil arbetsmetod har ett ramverk utvecklats. I det ramverket har sedan en prototyp av ett spel byggts.

Ramverket byggdes i utvecklingsmiljön Microsoft Visual Studio då den har fullt stöd för utveckling mot .Net-plattformen. För att underlätta arbetet med flera utvecklare användes SVN som versionshanteringssystem.

Insamlandet av den information som krävdes för att bygga ramverket och spelprototypen har främst skett genom litteraturstudier av API:erna för de tekniker som använts.

En sökning efter artiklar och rapporter som behandlar liknande koncept, webbapplikationer byggda av Javascriptmoduler eller -plugins, genomfördes.

1.4 Tidigare forskning

Ingen tidigare dokumenterad forskning har hittats som beskriver något liknande koncept.

1.5 Avgränsningar

Ramverket ska vara enkelt, vilket innebär att endast de nödvändigaste delarna kommer att implementeras. Det betyder att det ska gå att skapa en prototyp av ett

nivåbaserat spel med ramverket, men det är också det enda kravet. Alltså kommer fokus inte att läggas på aspekter som användarvänlighet eller säkerhet.

Ramverket kommer att byggas på .Net-plattformen då det är den mjukvaruplattform Sigma främst använder sig av.

1.6 Rapportens upplägg

Kapitel ett fastställer rapportens sammanhang genom bakgrund, mål, metod och avgränsningar. Kapitel två ger en teknisk bakgrund till de tekniker som används för att bygga upp projektramverket. Därpå följer kapitel tre med analyser av målet och kraven, och slutsatser för hur ramverket ska utformas dras sedan därifrån. I kapitel fyra presenteras resultatet av projektet, för att sedan diskuteras i kapitel fem. I det avslutande sjätte kapitlet dras slutsatser utifrån resultatet och diskussionen.

2 Teknisk bakgrund

Här följer teknisk bakgrund för de olika tekniker som använts för att implementera ramverket. Först presenteras modulär programmering och plugins, då utbytbara komponenter är den centrala delen i projektet. Därefter beskrivs .Net-plattformen, som använts för att bygga projektramverket. Slutligen förklaras hur webbläsare och Javascript fungerar, eftersom båda är vitala delar för ramverkets funktionalitet.

2.1 Modulär programmering

Modulär programmering är en programmeringsteknik som går ut på att dela upp ett större system i mindre komponenter, så kallade moduler. Varje modul har ett eget avskilt ansvarsområde och ska fungera självständigt, dock kan en modul ta hjälp av en annan för att lösa ett delproblem (Brogi, 1994).

En av fördelarna med att programmera modulärt är att ett projekt går att dela upp i mindre delar, vilket gör att koden i projektet blir mer hanterbar och får en logisk uppdelning. Det går också att dela upp programmeringsuppgifterna i en utvecklargrupp, så att arbetet kan utföras snabbare. Ytterligare en fördel med modulär programmering är att kod kan återanvändas. Detta gör det lättare att underhålla kod, eftersom den är väl avgränsad gällande funktionalitet och ansvar. Det innebär att det är lätt att leta upp den specifika del som behöver ändras (Brogi, 1994).

2.2 Plugins

Pluginbaserat tankesätt liknar till viss del modulär programmering. Ett plugin är ett program som inte kan köras självständigt utan laddas in som en del i ett program. Grundtanken med plugins är att kunna tillföra funktionalitet till ett program i efterhand. Samtidigt ger plugins möjligheten att minska ner storleken på ett program genom att endast grundprogram och plugins för önskad funktionalitet installeras. (Wordpress Codex, 2013).

Ett välkänt exempel på ett plugin är Adobes Flash Player som finns till de flesta stora webbläsare. Det tillför funktionaliteten att kunna visa upp flashgrafik i webbläsaren (Adobe, 2013).

För att förstå vad ett plugin är kan följande två citat beaktas: *“a plugin is code that can become part of a program under that program's control”* (Vondevoorde, 2006) och *“a plugin is a re-usable and configurable software component that can be dynamically loaded by an application in order to extend its existing functionalities”* (Incardona et.al, 2009). Citaten ovan förklarar att ett antal plugins tillsammans med en webbläsare eller annan programvara kan bilda ett system med utökad funktionalitet.

Ett plugin är således en form av en modul som inte kan köras som ett eget program. Det behöver istället ett värdprogram att laddas in i, vars funktionalitet det sedan utökar.

2.3 .Net

.Net-plattformen är utvecklad av Microsoft och består av en stor mängd klassbibliotek med funktionalitet för ett flertal olika områden, så som databashantering, grafiska gränssnitt och webbutveckling (Microsoft, 2013).

Common Language Runtime (CLR) är en virtuell maskin samt en del av .Net-plattformen. Med hjälp av CLR kan all kod som skrivits i något av de programspråk som stöds av .Net köras. CLR kan hantera både objekt och referenser till objekt, och använder sig av skräpsamling för att ta bort objekt som inte längre används. CLR förenklar designen av komponenter och program då det låter komponenter interagera med varandra mellan program oavsett programspråk. Detta gör det enkelt att skriva olika delar av ett system i olika programspråk utefter vad som passar varje del bäst (CLR, 2013).

Asp.net, ett webbramverk baserat på .Net, använder en blandning av två olika språk för att skapa webbsidor, ett för att generera html-element och ett för logik. Html-elementen skapas med märkspråket Asp och logiken kodas i antingen C# eller Visual Basic. Vilket av de två som ska användas i en webbapplikation anges när projektet skapas i Visual Studio, därmed kan C# och Visual Basic inte användas tillsammans i samma webbapplikation. Båda språken är objektorienterade, vilket innebär att kod kan struktureras upp i separata klasser, vilka sedan kan användas av webbsidan (Asp.net (1), 2011).

Visual Basic och C# har också tillgång till html-elementen som skapats med Asp-kod, vilket innebär att dessa kan

modifieras direkt från backendkoden. På samma sätt kan Asp-koden använda sig av klasser och metoder lagrade i en Visual Basic- eller C#-fil (Asp.net (1), 2011).

2.4 Webbläsare

Dagens webbläsarmarknad är mycket mer fragmenterad än vad den var när webben kommersialiserades i mitten av 1990-talet. Idag är det inte längre en webbläsare som dominerar marknaden, utan istället finns det tre stora webbläsare i form av Googles Chrome, Microsofts Internet Explorer och Mozillas Firefox. Dessa tre innehar vardera mellan 20-40% av marknaden enligt olika mätningar (Statcounter Globalstats, 2013).

Idag stöder alla stora webbläsare html, Javascript och css, vilket är tre programspråk som används för att beskriva hur en hemsida ska se ut och fungera. Samtliga stora webbläsare följer idag givna standarder i stor utsträckning, dock har de olika renderingsmotorer. Därför kan viss specialkod behöva skrivas för att en webbsida ska fungera på samma sätt i alla webbläsare.

2.5 Javascript

Javascript är ett skriptspråk framtaget av Netscape Communications i mitten av 1990-talet. Det utvecklades för att användas som ett klientskriptspråk i webbläsare, men det finns även andra tillämpningar av det. Att det är ett klientskriptspråk innebär att Javascriptkoden tolkas och körs direkt i klientens webbläsare (Flanagan, 2006).

Språket har stöd för att skapa webbinnehåll, till exempel kan ett Javascript-program skapa alla typer av html-element. Det finns också stöd för att skapa andra, Javascript-specifika, element, till exempel popup-rutor (Flanagan, 2006). Javascript stödjer också rendering på så kallade canvas, vilket är en yta där animationer och figurer kan ritas upp utan begränsningar. Detta ger utökade möjligheter när det gäller spelteknik, då en hel spelvärld kan målas ut direkt i webbläsaren. Canvas i sig är inte specifikt för Javascript utan något som ingår i version 5 av html-standarden (Flanagan, 2010).

Det är möjligt att omvandla vanliga textsträngar till körbar Javascriptkod, då Javascript tolkas direkt i webbläsaren.

Språket har en inbyggd metod för detta, kallad eval, som stöds av alla moderna webbläsare (Flanagan, 2006).

Det finns inbyggt stöd för att hämta textsträngar i realtid med hjälp av asynkrona anrop. Detta betyder att ett Javascript-program kan hämta ny data i bakgrunden utan att ladda om sidan. Javascripts inbyggda teknik för asynkrona anrop kallas för Ajax (Flanagan, 2006). Det finns också stöd för att hämta objekt från en webbserver. Språket följer en inbyggd standard för att omvandla objekt till textsträngar, kallad Javascript object notation, vilken är mer känd under förkortningen Json. Standarden för Json finns beskriven i standarddokumentet ECMA-262 (ECMA International, 2011).

Genom att kombinera Json, Ajax och Javascripts inbyggda evalueringsmetod är det således möjligt för en klient kodad i Javascript att hämta kod från en server och sedan köra den. Servern gör först om koden från ett objekt till en textsträng enligt Json-standarden. Klienten hämtar textsträngen i bakgrunden med hjälp av Ajax, gör om den hämtade strängen till körbar kod med evalueringsmetoden och anropar slutligen den körbara koden precis som om den hade funnits tillgänglig från början.

3 Realisering

I detta kapitel redovisas hur ramverkets prototyp tagits fram, hur de olika teknikerna hör samman samt motiveringar till de ställningstaganden som gjorts. Beskrivningarna sker till en början på en hög abstraktionsnivå, där en grundläggande analys av krav och mål görs. Genom denna analys tas en övergripande struktur fram. Den bryts därefter ner i mindre delar som beskrivs på en detaljerad nivå.

3.1 Analys

Utifrån projektets mål och litteraturstudierna som gjordes framstod den bästa lösningen vara att bygga ramverket som ett pluginsystem. Ett sådant system är uppbyggt av utbytbara komponenter och tillåter återanvändande av kod. Fördelen med att bygga varje nivå som ett plugin är att programmeraren inte behöver ta hänsyn till resterande delar av systemet, då varje plugin är helt oberoende av övriga komponenter.

I och med att plugins möjliggör återanvändande av kod, bör det finnas möjlighet att spara funktionalitet i ett globalt bibliotek. Detta för att undvika onödig redundans i form av metoder och variabler som är lagrade flera gånger.

Då ett av projektets mål är att utbytbara komponenter ska kunna laddas in i en presentationssida, passar Javascript bra som pluginspråk. Som beskrivits i den tekniska bakgrunden har Javascript möjlighet att hämta, evaluera och sedan köra kod som laddats in efterhand. Det har dessutom stöd för att skapa interaktivt och dynamiskt innehåll, samtidigt som det är implementerat i alla stora webbläsare.

För att klientens presentationssida ska kunna exekvera ett plugin krävs ett sätt att initiera nivåerna. Detta kan realiseras genom att varje plugin innehåller en initeringsmetod som ramverket anropar vid inladdning av pluginet. Då det också ska vara möjligt att byta nivå behövs det ett sätt att ta bort alla delar av det gamla pluginet. Detta löses genom att varje plugin innehåller en avslutningsmetod som tar bort allt som skapats av pluginet.

De plugins som läser in någon form av svarsdata behöver kunna kontrollera densamma. I och med att ramverket ska stödja rekryteringsspel är det viktigt att ett plugins korrekta

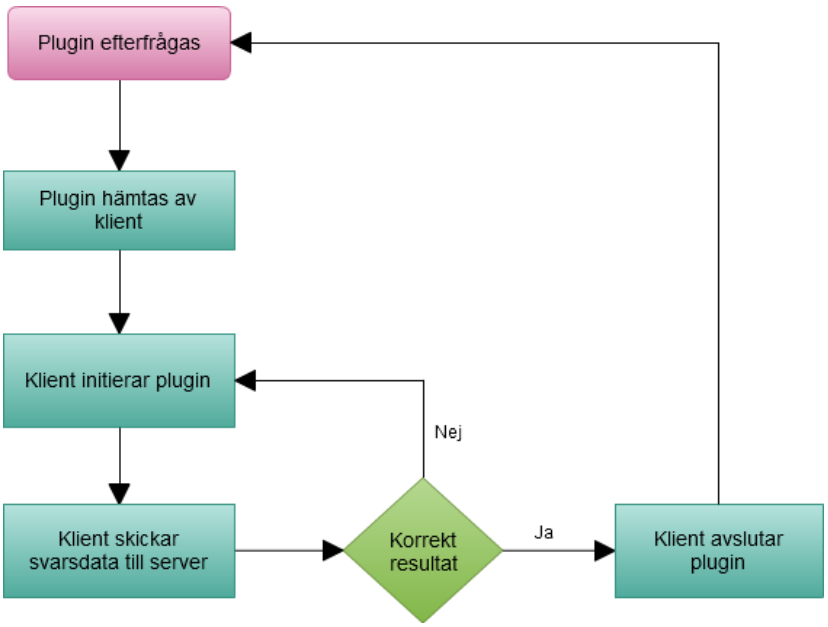
svar är dolt för användaren, vilket innebär att svarsdata måste kontrolleras på serversidan.

Webbläsaren ska alltid stanna på samma presentationssida, eftersom innehållet på sidan ändras dynamiskt vid nivåbyte. Detta beror på att det inte finns något behov av att ha flera olika sidor, utan det behövs endast en sida som hämtar och visar en nivå i taget. För att applikationen ska följa målet att endast ha en presentationssida som aldrig laddas om, behöver klienten hämta all data genom Ajax-anrop.

Klienten ska kunna hämta plugins från servern. Till detta behövs ett väldefinierat API med två sätt att efterfråga plugins. Det ena sättet innebär att klienten efterfrågar ett specifikt plugin från servern. Det andra innebär att klienten efterfrågar ett plugin för nästa nivå från servern.

3.2 Struktur

Som specificerats i målet ska ramverket kunna skicka data mellan klient och server, vilket visas i figur 1. Kommunikationen mellan klient och server sker i enlighet med analysen som gjorts i föregående delkapitel.



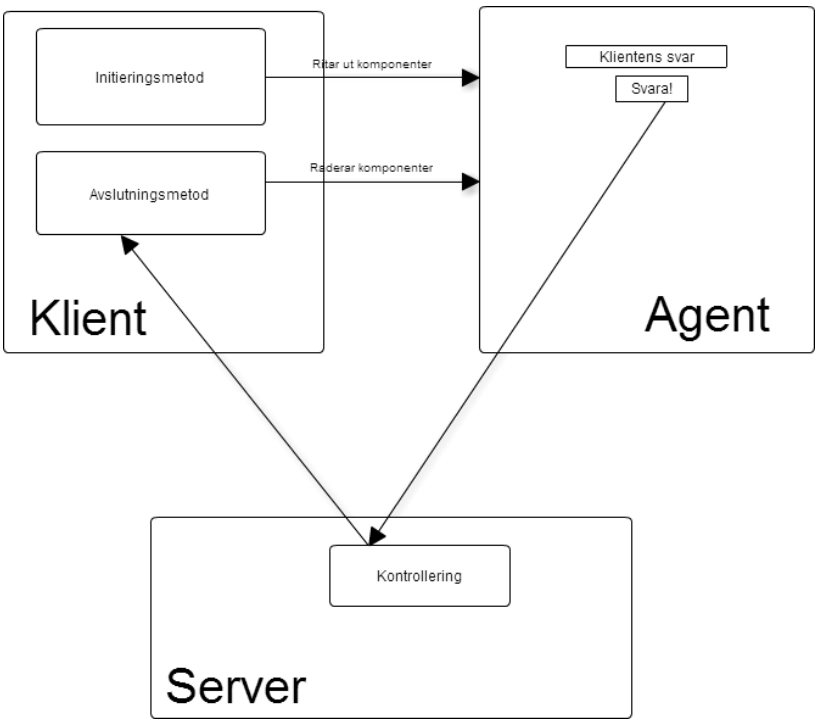
Figur 1. Flödesschema över ett plugins livscykel

Initieringen av kommunikationen sker genom att klienten efterfrågar och hämtar ett plugin från servern. Klienten initierar därefter pluginet och dess innehåll renderas på presentationssidan. När användaren är färdig med ett plugin

skickas pluginets svarsdata till serverdelen. Serverdelen kontrollerar den mottagna datan. Baserat på kontrollens utfall behålls antingen det nuvarande pluginet, eller så returneras ett nytt plugin innehållandes nästa nivå. Innan ett nytt plugin skickas från servern anropas avslutningsmetoden i det nuvarande och sålunda är flödesschemat tillbaka där det började. Om en användare stänger ner webbläsaren efter att ha avancerat några nivåer och återansluter vid ett senare tillfälle kommer servern returnera det senaste plugin som användaren hämtat.

3.2.1 Pluginstruktur

Varje plugin behöver innehålla en initieringsmetod och en avslutningsmetod, vilket specificerats i analysen. Initieringsmetoden ritar ut ett plugins komponenter och avslutningsmetoden raderar de utritade komponenterna. Ett plugin behöver också innehålla en serverdel som kontrollerar svarsdata. I figur 2 visas ett plugins livscykelsdiagram.



Figur 2. Förenklad översikt av ett plugins livscykel.

3.2.1.1 Klientdel

Initieringsmetoden ansvarar för att bygga upp designen för det plugin som den är en del av. Designen kan bestå av till exempel html-element, css eller bilder. Vidare ansvarar initieringsmetoden för att starta eventuella tjänster som krävs för att pluginet ska kunna köras och avslutas korrekt.

Exempel på tjänster som kan startas är inläsning av video i webbläsaren samt starta videon när tillräckligt med data hämtats. Initieringsmetoden ska kontrollera programflödet, och när exekveringen når sin slutpunkt anropa avslutningsmetoden, för att därefter ladda nästa plugin.

Avslutningsmetoden ansvarar för att, efter pluginet körts klart, plocka bort alla komponenter som pluginet skapat. Även variabler som pluginet skapat ska tas bort av avslutningsmetoden. Om objekt inte förstörs i avslutningsmetoden kommer de finnas kvar när nästa plugin laddas, och kan då skapa krockar både i koden och i det grafiska gränssnittet.

3.2.1.2 Serverdel

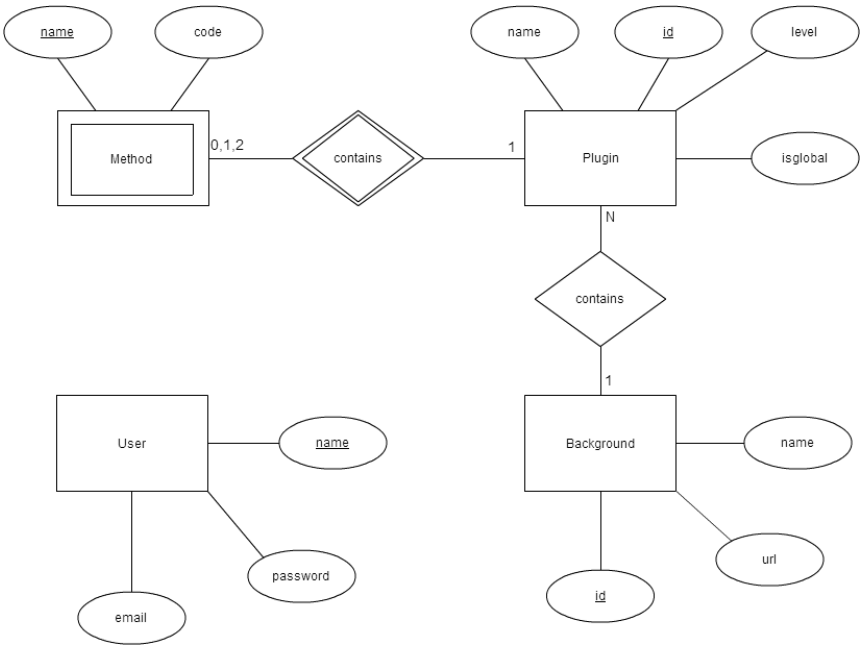
Serverdelen ska tillhandahålla en metod för att utvärdera svarsdata. Denna metod ska returnera ett sanningsvärde som indikerar om korrekt svarsdata har angivits, alternativt returnera sant om inga svarsdata behövs.

3.3 Pluginhantering

För att underlätta utvecklingen av prototypen utvecklades ett administrationsverktyg för att skapa, redigera och testa plugins. Administrationsverktyget, kallat PMS (Plugin Management System), behövde funktionalitet för att lägga till extra data så som bilder, videor eller exekverbara program, att användas av plugins. Se appendix, 8.1 för fullständig utformning av pluginhanteringssystem.

3.3.1 Databas

För att utvecklade plugins ska kunna sparas och användas behövs någon form av datalagring, vilket i det här fallet sköts av en databas. I figur 3 visas relationerna mellan entiteterna i databasen i ett entitetsdiagram.



Figur 3. Databasen och dess kopplingar

User är helt fristående, utan någon koppling till andra entiteter. Den håller reda på alla PMS-användarnas namn, lösenord och e-postadress. Namnet identifierar en spelutvecklare och måste således vara unikt.

Högst upp i höger hörn finns entiteten Plugin. Varje Plugin har ett namn, ett unikt id och en nivå. Plugin har dessutom en medföljande bakgrund som i sin tur har ett namn, en URL till bilden och ett unikt id.

Högst upp i vänster hörn i enitetsdiagrammet finns Method, en svag entitet med identifierande entitetstyp Plugin. Den har attributen namn och kod. Här lagras initierings- och avslutningsmetoderna för respektive Plugin. Koden lagras i databasen som ren text.

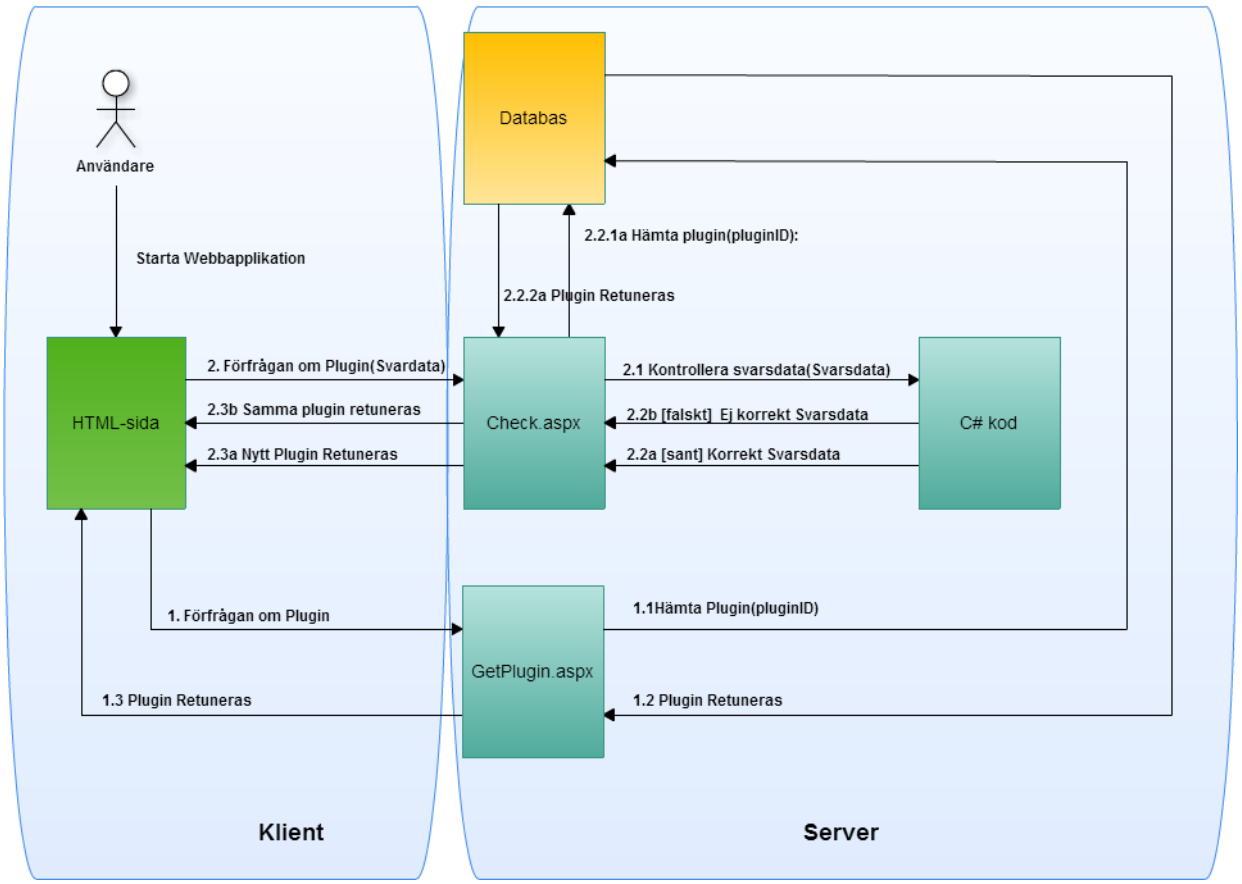
4 Resultat

Ett fungerande webbramverk som uppfyller projektets mål och krav har tagits fram. En prototyp i form av ett rekryteringsspel har skapats i ramverket. Ramverket stödjer nivåbaserade spel, och varje nivå i systemet är uppbyggd som en utbytbar komponent. Systemet har både en serversida och en klientsida vilka har olika ansvarsområden. Serversidan ansvarar för att kontrollera vilken nivå den anslutna klienten är på och räknar ut vilken som är nästa nivå när klienten har klarat av den nuvarande nivån. Klienten beskrivs närmre i kapitel 4.1.

Utseendemässigt kommer en slutanvändare aldrig att se någon skillnad mellan en hemsida uppbyggd med det här ramverket och en annan hemsida.

4.1 Implementation

Figur 4 visar implementationen i sin helhet och interaktionen mellan de olika komponenterna.



Figur 4. Implementationen i ett förenklat kommunikationsdiagram

4.1.1 Presentationssida

Presentationssidan är en ensidesapplikation, och visar det plugin som hämtats hem från servern. Pluginet som ritas upp representerar en spelnivå.

Klienten behöver följa ett antal steg för att ett plugin ska kunna visas på rätt sätt i presentationssidan. Det första steget är att ansluta sig till ett API där aktuellt plugin hämtas. När klienten hämtat ett plugin har den tillgång till det som ett Json-objekt. Eftersom varje plugin är skrivet i Javascript behöver klienten omvandla Json-objektet till körbar kod via text. När omvandlingen från text till körbar kod är gjord anropar klienten initieringsmetoden från den nyligen

omvandlade texten. Detta beskrivs i figur 5.

```
function runPlugin(plugin) {
    if (plugin == null) {
        console.log("Cannot run null!");
        return;
    }

    json = JSON.parse(plugin);

    methods = json["methods"];

    background = json["background"];
    setBackground(background.url);

    for (i = 0; i < methods.length; i++) {
        eval("function " + methods[i].name + "() { " + methods[i].code + " }");
    }

    onCreate();

    console.log(plugin);
}
```

Figur 5. Kod för att köra ett plugin i klienten

För att omvandla ett plugin från Json till körbar kod används Javascripts inbyggda metoder Json.parse samt eval. Json.parse gör om Json-data till en ren textsträng. Den textsträngen skickas sedan som parameter till eval som gör om den till exekverbar Javascript-kod.

Då initieringsmetoden är ansvarig för programflödet sköter den kommunikationen med serverdelen. När klienten ska skicka svarsdata till servern sker detta via initieringsmetoden. Om servern godkänner svarsdatan returneras ett nytt plugin, var på avslutningsmetoden anropas. När avslutningsmetoden körts är flödet tillbaka till ursprungspositionen, där det nya pluginet startas.

4.1.2 API

För att klienten ska kunna hämta plugins från servern har ett API skapats. Detta API består av två webbsidor som uppfyller de krav som analysen specificerar. Kraven innebär att klienten ska kunna hämta ett specifikt plugin eller ett plugin för nästa nivå från servern. Om det är första gången klienten efterfrågar ett plugin kommer API:t att returnera ett slumpmässigt plugin från lägsta möjliga nivå, oavsett vilken av webbsidorna som anropas.

Den första webbsidan, kallad GetPlugin, tar ett plugin-id som inparameter. Om det existerar ett plugin vars id är samma som den angivna inparametern, returneras pluginet till

klienten. Om det plugin-id som angavs som indata däremot inte matchar ett plugin, returneras det plugin som servern senast godkände. Om inget plugin har godkänts returneras något av de plugins som har lägst nivå.

Den andra webbsidan för pluginöverföring, Check, returnerar nästa nivås plugin. Innan nästa nivå returneras kontrolleras pluginets svarsdata. Kontrollen görs genom att Check anropar pluginets serverdel, som antingen godkänner eller underkänner svarsdatan. Godkänns svarsdatan skickas ett slumpmässigt valt plugin från nästa nivå tillbaka till klienten. Om svarsdatan däremot inte godkänns skickas null som returvärde. Fås null som returvärde gör klienten ingenting.

Båda webbsidorna i API:t returnerar ett plugin i Json-format som resultat, vilket innehåller all information klienten behöver för att kunna köra pluginet.

4.1.3 Plugins

Varje plugin är indelat i två delar, en initieringsmetod och en avslutningsmetod. I implementationen av ramverket har dessa kallats onCreate och onDestroy.

I figur 6 visas en del av den kod som ligger till grund för ett av de plugins som implementerats. I detta exempel skapas några html-element i onCreate. onCreate innehåller dessutom en metod som hanterar hur klienten kommer vidare till nästa nivå. Exemplet visar att när användaren trycker på bilden "DubbeldörrMedHandtag.png" anropas först onDestroy för att ta bort de komponenter som lagts till av pluginet, och sedan körs det plugin som hämtats från servern. Anledningen till att onDestroy anropas utan att veta om servern kommer att godta användarens svar beror på att servern alltid returnerar sant vid ett anrop till servern för just det här pluginet.

```
        without = document.createElement("div");
        without.id = "without";
        withoutImg = new Image();
        withoutImg.id = "doorWithoutHandles";
        withoutImg.width = w;
        withoutImg.height = h;
        withoutImg.src = "/Images/Uploaded/DubbeldörrUtanHandtag.png";
        without.appendChild(withoutImg);
        withDiv = document.createElement("div");
        withDiv.id = "withDiv";
        withImg = new Image();
        withImg.id = "handles";
        withImg.width = w;
        withImg.height = h;
        withImg.src = "/Images/Uploaded/DubbeldörrMedHandtag.png";
        withImg.onclick = function () { onDestroy(); getAndRunPlugin("/Plugins/Check.aspx"); };
        withDiv.appendChild(withImg);
        body.appendChild(without);
        body.appendChild(withDiv);
```

Figur 6. Javascript för ett plugin

Varje plugin har också en implementerad serverdel med en metod som utvärderar pluginets svarsdata. Figur 7 visar ett exempel på hur en serverdel till ett plugin ser ut.

```
public bool coolTest(){
   string s = Context.Request.Form["answer"];

   if(s == null) {
     return false;
   }
   s = Server.UrlDecode(s);
   return s.ToLower().Equals("gropen");
}

```

Figur 7. Serverdel av ett plugin vars svarsdata är korrekt om det är "gropen".

4.1.4 Globalt bibliotek

Det implementerade ramverket har stöd för återanvändande av funktionalitet. Detta har möjliggjorts genom ett globalt bibliotek där det är möjligt att spara funktioner och variabler som alla plugins har tillgång till. Det globala biblioteket kan redigeras i ramverkets pluginhanteringssystem. Alla plugins

som använder det globala biblioteket har en stark koppling till det. Det innebär att om en metod eller variabel i biblioteket ändras påverkas alla plugins som använder den. För mer information om det globala biblioteket, se Appendix.

5 Diskussion

I detta kapitel följer en diskussion av projektet. Diskussionen inleds med en beskrivning av de problem och utmaningar som uppstått under projektet. Därefter följer jämförelser med andra liknande projekt. Från dessa jämförelser sker sedan en djupare analys där för- och nackdelar mellan vårt och andra projekt jämförs. Slutligen kommer projektets framtid och eventuella expansionsmöjligheter att diskuteras.

5.1 Problem och utmaningar under projektet

Den första stora utmaningen som stöttes på under projektets gång var att rapporten skulle bidra med något nytt till vetenskapen, samtidigt som det givna projektet endast bestod i att utveckla en rekryteringstjänst åt Sigma. Det var först efter mycket gruppdiskussion och efterforskning som nuvarande mål kunde tas fram. När målen väl var utformade gick utvecklingen av ramverket snabbt framåt.

Alla i projektgruppen behövda läsa på om och sätta sig in i .Net-plattformen, Visual Studio, Javascript och C#, då det var nya tekniker för oss. Därefter löpte projektet på utan några större svårigheter.

Tyvärr upphörde samarbetet med Sigma på grund av meningsskiljaktigheter, vilket innebär att prototypen inte kommer att nyttjas av Sigma. Detta har dock inte haft någon inverkan på projektet eller rapporten i övrigt.

5.2 Jämförelser med liknande projekt

Som nämnts i kapitel 1.4 har inte något projekt som bygger på samma koncept som vårt hittats, trots utförliga efterforskningar. Den stora skillnaden mellan vårt koncept och vanliga webbapplikationer är att vårt hämtar ner Javascriptkod från en server för att sedan exekvera den i realtid. Vanligtvis finns istället all logik (Javascript) med från början när en sida laddas, och eventuellt kan delar av innehållet på sidan dynamiskt laddas in vid senare tillfälle.

5.3 Jämförelser av koncept

Då kandidatarbetet påbörjades fanns ingen teknisk specifikation för hur webbapplikationen skulle vara konstruerad. För- och nackdelar med olika metoder och

tillvägagångssätt jämfördes. De förutsättningar som var tvungna att uppfyllas var att svaren på uppgifterna inte fick visas för klienten samt att lösningen skulle stödja en mängd varierande problem i olika storlek och komplexitet

En lösning som diskuterades byggde på en fet klientmodell, alltså att lägga så mycket funktionalitet som möjligt hos klienten snarare än servern. Dock valdes den lösningen bort då det behövs en del serverlogik i form av svarsrättning till ett problemlösningsspel. Alltså skulle ett sådant spelprojekt inte lämpa sig bra för när en så fet klientmodell som möjligt ska tas fram.

En annan idé som diskuterades var att försöka strukturera upp designen av applikationen som en objektorienterad modell. Detta för att ta fram riktlinjer för vad som bör ligga i Javascript, html och css var för sig. Eventuellt hade det konceptet gått att kombinera med ett spelprojekt på ett tillfredsställande sätt, men för att ordentligt testa hur en uppdelning generellt bör se ut, bör nog snarare webbsidor av flera olika typer byggas. Detta för att kunna dra slutsatser om vad som är gemensamt eller skiljer olika webbsidetyper emellan.

Med det taget i beaktning valdes att bygga ramverket som ett pluginsystem då det medför fördelar som stämmer väl överens med de mål som satts upp. En komplett lista på de fördelar som kommer med pluginsystemet kan läsas i kapitel 5.4.

5.4 För- och nackdelar med konceptet

En fördel med att hämta hem plugins från servern vid exekvering är att användaren inte har tillgång till skriptet i förväg vilket kan vara fördelaktigt om webbapplikationen behöver dölja information för användaren. Däremot kan något längre väntetider uppstå för användaren under interaktionen med applikationen i och med att det nya pluginet måste hämtas hem från servern innan det kan exekveras.

En annan fördel som kan bli påtaglig vid implementering i större webbapplikationer är att bandbredd sparas på att inte mer logik än den som används förs över från server till klient. Det kan innebära kortare laddtider totalt över hela applikationens körning, då inte lika mycket data behöver överföras till en början. Den totalt överförda datamängden blir också mindre om inte hela webbapplikationen används, samt

kan flera små överföringar istället för en stor innebära att användaren aldrig märker av någon laddtid.

Vad som avgör om sidan upplevs som långsammare eller snabbare än om all logik laddas direkt beror på logikens storlek för nivåerna. Det bör visa sig mest påtagligt vid ett spel med många små nivåer, där laddtiden vid start skulle bli märkbar om hela spelet skulle laddas ned direkt, emedan laddtider antagligen inte skulle märkas alls om endast en nivå laddas ned i taget.

Det pluginbaserade ramverket är lätt att utöka och bygga ut. Dessutom fås en nästintill objektorienterad uppdelning av kod (med varje nivå som ett objekt) vilket förenklar utvecklingen i den mening att koden har en logisk uppdelning. Samtidigt finns ingen koppling mellan olika plugins, vilket ytterligare förenklar för utvecklarna. En utvecklare kan alltså till exempel ta bort eller redigera ett plugin utan att det påverkar andra plugins.

5.5 Administrationsverktyget

En annan utmaning inom projektet var att relativt mycket tid spenderades på att hitta ett färdigt CMS (content management system, se ordlistan) att anpassa till att kunna administrera ramverket. Sökningar efter ett mindre CMS med inriktning mot programmering gjordes, men alla CMS som hittades och testades byggde på idén att ett CMS skulle hjälpa en användare att bygga en fullständig webbsida med hjälp av inbyggda verktyg.

Eftersom CMS:et skulle vara inriktat mot programmerare fanns inget behov av traditionella CMS-funktioner, därför behövde ett nytt CMS skapas från grunden. Det behövde kunna skapa plugins enligt en given nivåstruktur samt uppfylla de krav som fanns på projektet, som till exempel att ett plugin skulle kunna utvärdera och godkänna svarsdata. Det enklaste blev alltså att utveckla ett pluginhanteringssystem, PMS.

5.6 Expansionsmöjligheter

Det finns två huvudsakliga förbättringar av ramverket som skulle kunna göra pluginsystemet mer användbart. Den ena är att bygga något slags bibliotek av standardmoduler som skulle underlätta skapandet av plugins. En standardmodul skulle kunna vara en metod som tar en textsträng som argument och returnerar en formaterad text.

Den andra stora förbättringen som kan göras är inte lika klar och tydlig, men baserar sig i att det är ganska omständigt att skriva hela hemsidor i Javascript. Därför bör Javascript bytas mot antingen html och Javascript kombinerat eller eventuellt ett wrapperspråk som genererar Javascript från betydligt enklare konstruktioner.

Säkerhet är en aspekt som inte har behandlats och det är något som skulle behöva göras för att skapa en användbar produkt utifrån ramverket.

Därutöver skulle databasen kunna utökas. Till exempel skulle information om varje spelare och deras respektive interaktion med webbapplikationen kunna sparas, för att hålla koll på hur många försök som gjorts på varje nivå och hur lång tid varje nivå har tagit. Det skulle kunna ge ytterligare kunskap om hur en spelare presterat.

Vidare skulle ramverket kunna generaliseras ytterligare, till att inte vara specialiserat på nivåbaserade spel. Istället skulle det kunna stödja konstruktionen av vilken webbapplikation som helst med hjälp av plugins. Det vore en vidareutveckling av dagens Ajax-baserade webbapplikationer som dynamiskt byter ut sitt innehåll att också dynamiskt byta ut logiken för varje delsida när den laddas.

6 Slutsatser

Målet med projektet var att skapa ett ramverk för nivåbaserade webbspel, och en spelprototyp i det. Ramverket och prototypen som vi skapat uppfyller alla mål och håller sig inom de avgränsningar som fanns på projektet. Det nivåbaserade ramverket förser klienten med utbytbara komponenter. De laddas in i presentationssidan och innehåller både logik och grafik som representerar en nivå. Prototypen påvisar att pluginkonceptet är användbart och fungerande.

Så vitt vi kan bedöma är ramverket unikt, och tack vare plugindesignen skall det vara möjligt att skapa alla typer av nivåbaserade spel i det.

7 Referenser

Adobe (2013) *Adobe Flash Player 11 / Features.* http://www.adobe.com/se/products/flashplayer/features._sl_id-contentfilter_sl_featuredisplaytypes_sl_new.html (Hämtad 2013-05-14)

Asp.Net. 2011. Asp.Net Page Life Cycle Overview. http://msdn.microsoft.com/en-us/library/ms178472(v=vs.100).Aspx (Hämtad 2013-05-03).

Asp.Net (1). 2011. Asp.Net Web Server Controls Overview. http://msdn.microsoft.com/en-us/library/zsyt68f1(v=vs.100).Aspx (Hämtad 2013-05-03).

Asp.Net (2). 2013. Online Documentation - Developer Express Inc. http://documentation.devexpress.com/#AspNet/CustomDocument7873

Brogi et al. 1994. Modular Logic Programming. *ACM Transactions on Programming Languages and Systems,* vol 16, nr 4, ss 1361 - 1398.

Crockford, Douglas. 2004. *Javascript: The good parts*. O'Reilly media. E-bok.

CLR. 2013. Common Language Runtime. http://msdn.microsoft.com/en-us/library/ddk909ch(v=vs.71).Aspx (Hämtad 2013-05-17).

Diaz, Dustin och Harmes, Ross. 2008. *Pro Javascript Design Patterns*. Apress. E-bok.

Ecma International. 2011. ECMA-262 Standard. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf (Hämtad 2013-06-01)

Flanagan, David. 2006. *Javascript: The Definitive Guide, 5th Edition : Activate Your Web Pages*. O'Reilly Media. E-bok.

Flanagan, David. 2010. *Canvas Pocket Reference : Scripted Graphics for html5*. O'Reilly Media. E-bok.

Incardona, Marie-Francoise et al. 2009. EDNA: a framework for plugin-based applications
applied to X-ray experiment online data analysis, page 873
http://journals.iucr.org/s/issues/2009/06/00/wa5014/wa5014.pdf

Internet world stats. 2013. *Internet Growth Statistics - the Global Village Online.*
http://www.internetworldstats.com/emarketing.htm (Hämtad 2013-05-06).

Microsoft. 2003. *Internet Explorer History.*
http://web.archive.org/web/20030704024306/http://www.microsoft.com/windows/WinHistoryIE.mspx (Hämtad 2013-05-10).

Powell, Thomas och Schneider, Fritz. 2001. *Javascript: The Complete Reference*. McGraw-Hill/Osborne. E-bok.

Statcounter Globalstats. 2013. *Top 5 Browsers from Apr 2012 to Apr 2013.*
http://gs.statcounter.com/#browser-ww-monthly-201204-201304 (Hämtad 2013-05-10).

Stefanov, Stoyan. 2010. *Javascript Patterns - Build Better Applications with Coding and Design Patterns*. O'Reilly Media. E-bok.

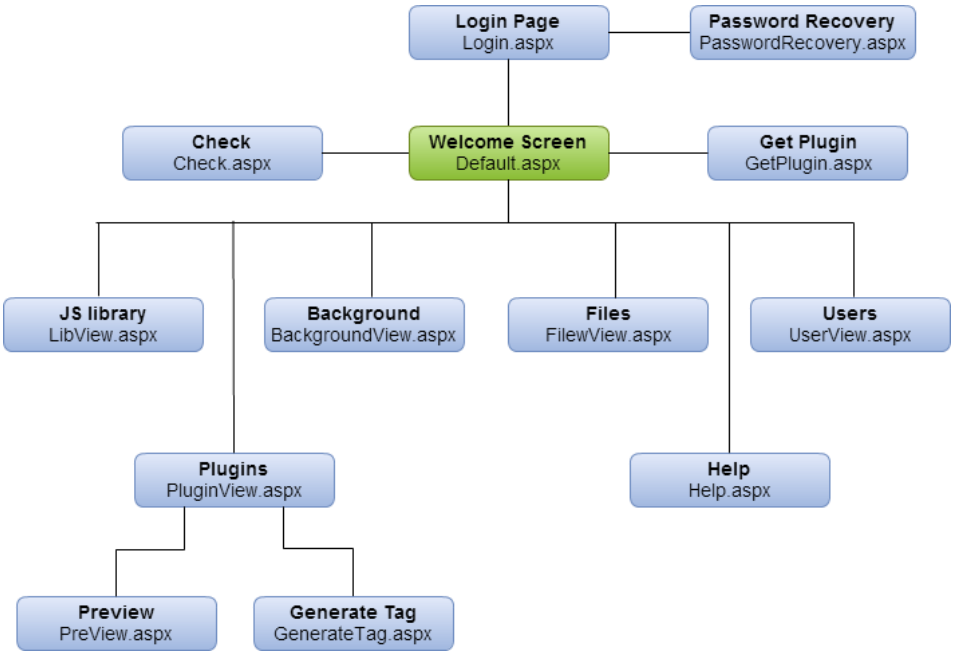
Vandevoorde, Daveed. 2006. *Plugins in C++, page 1.*
http://www.open-std.org/Jtc1/SC22/wg21/docs/papers/2006/n2074.pdf
(Hämtad 2013-05-01).

Wordpress Codex. 2013. *Plugins.*
http://codex.wordpress.org/Plugins (Hämtad 2013-05-14).

WordPress Codex. 2013. *Writing a Plugin*.
https://codex.wordpress.org/Writing_a_Plugin (Hämtad 2013-05).

8 Appendix

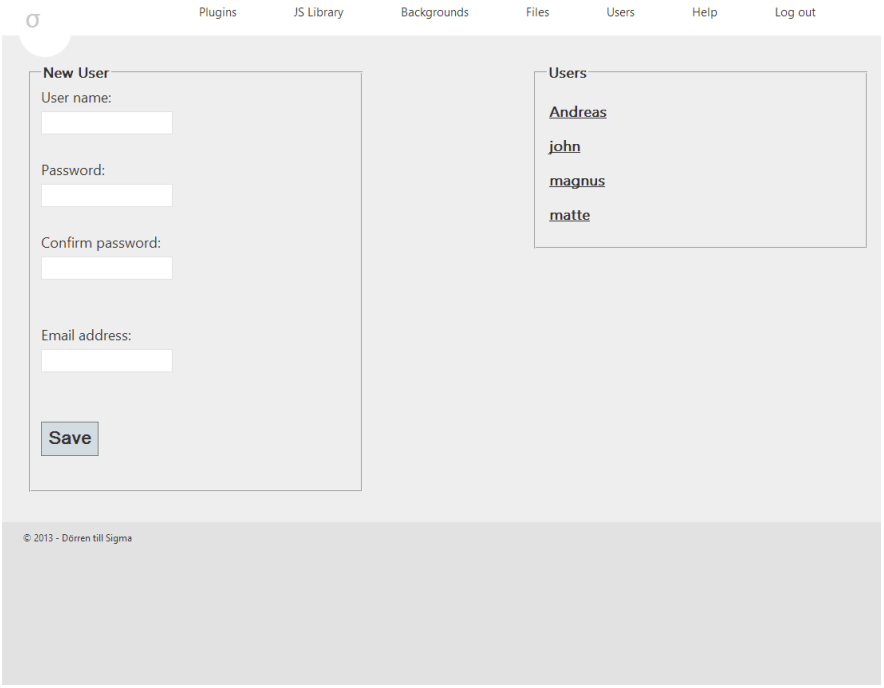
8.1 Översikt över pluginhanteringssystemet



Figur 8. Översikt över de sidor som bygger upp PMS:et

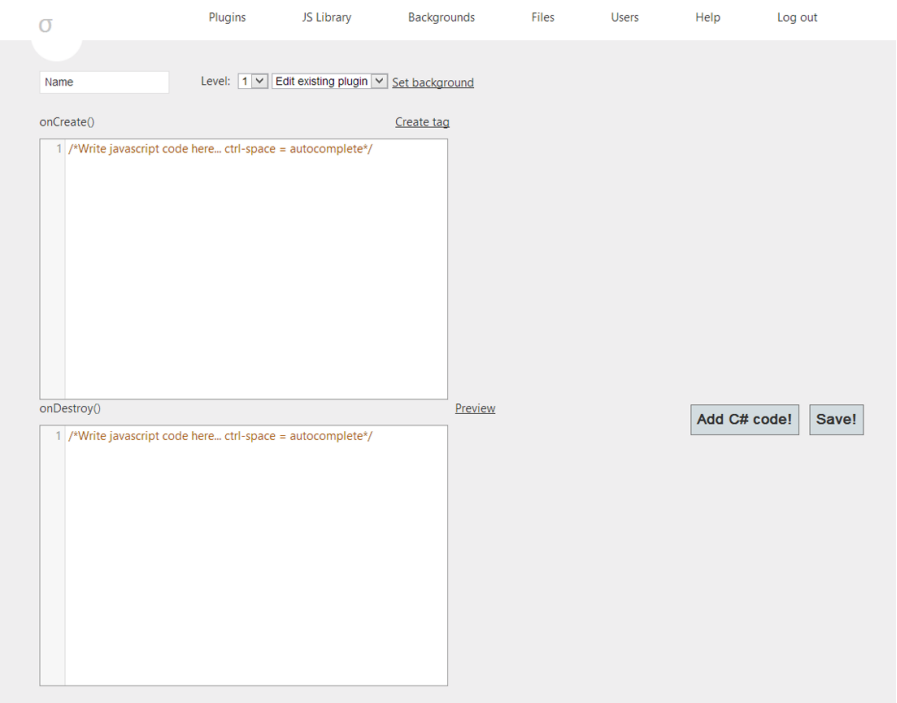
8.2 Grafiskt gränssnitt

Det grafiska gränssnittet till PMS:et utformades först genom att skapa enklare mockups av de olika sidor som skulle vara med. Dessa mockups låg sedan till grund för det grafiska gränsnittets uppbyggnad. Nedan följer ett urval av den nuvarande designen för några av de sidor som skapades.



Figur 9. På denna sida hanteras användarna till PMS:et.

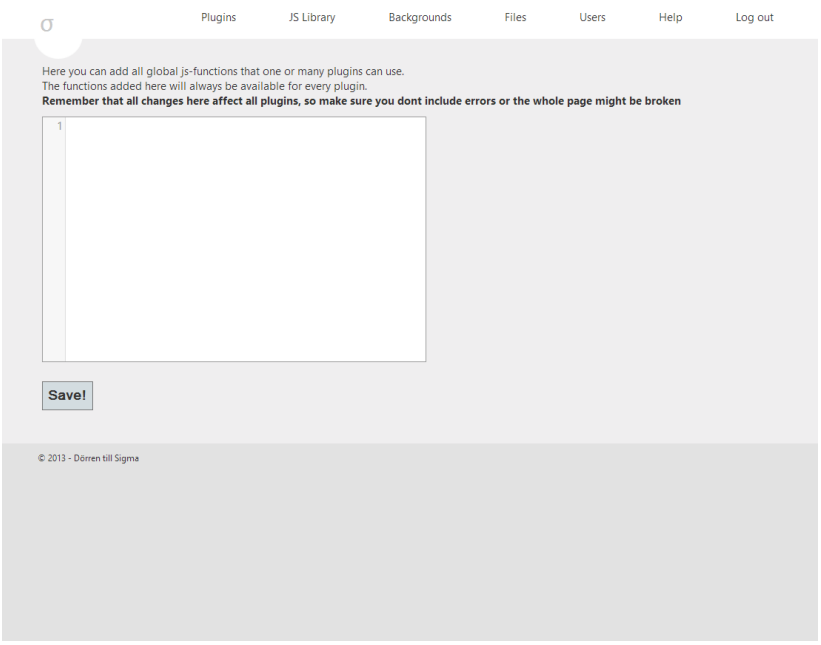
Alla användare av PMS:et har fullständiga rättigheter. Detta betyder att varje användare kan lägga till, ändra och radera användare.



Figur 10. På denna sida hanteras plugins.

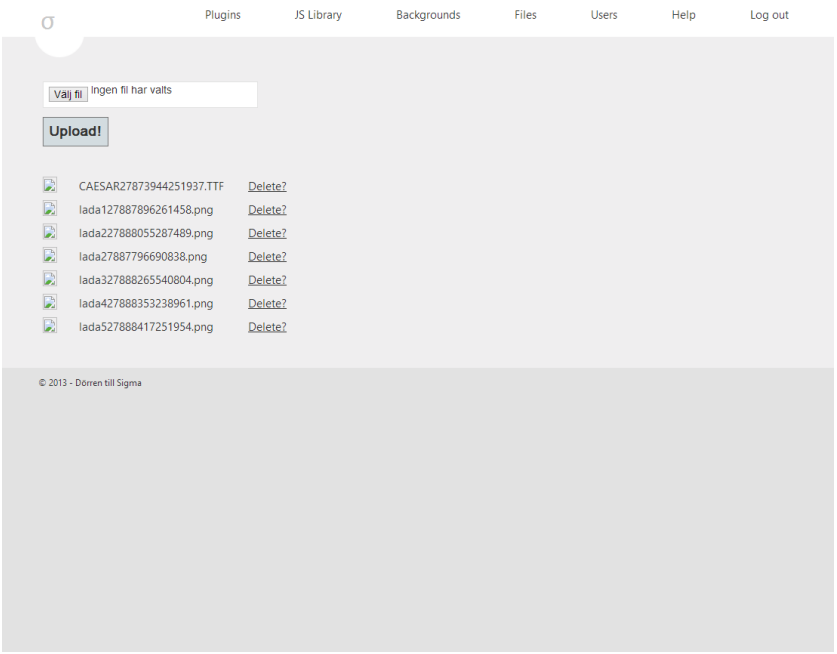
Varje plugin innehåller namn, nivå och bakgrund. Om namn inte fylls i används det standardnamn som redan står i textfältet. Om nivå eller bakgrund inte specifieras väljs nivå 1 och den första bakgrunden som finns tillgänglig. Det finns två textfält där kod för initierings- och avslutningskod skrivs, ovan benämnda som onCreate och onDestroy.

Användaren kan välja att visa en förhandsvisning av den kod som skrivits genom att klicka på länken “Preview”. Klickar användaren på “Create Tag” öppnas en dialog som hjälper användaren att skapa olika html-element.



Figur 11. På denna sida kan användaren skapa metoder som kan nyttjas av samtliga plugins.

I det globala biblioteket lagras metoder och variabler som kan användas av alla plugins i systemet.



Figur 12. På denna sida kan användaren ladda upp filer som kan nyttjas av samtliga plugins.