Programmering av en webbapplikation med ramverket CodeIgniter

Daniel Sundgren

Examensarbete
Informations- och medieteknik
2015

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informations- och medieteknik
Identifikationsnummer:	5011
Författare:	Daniel Sundgren
Arbetets namn:	Programmering av en webbapplikation med ramverket CodeIgniter
Handledare (Arcada):	Göran Pulkkis
Uppdragsgivare:	Thomas Finne

Sammandrag:
Examensarbetet är en uppföljning av hur man skapar en webbapplikation med ramverket CodeIgniter. I arbetet beskrivs vilka moduler och bibliotek som behövs för att få en CodeIgniter-applikation att fungera självständigt för företagsbruk. Focus är på funktionella delar utan att det grafiska användargränssnittet beskrivs. Först motiveras och sedan presenteras de tekniska lösningar som användes. Arbetsrapporten går även in på hur man använder de olika modulerna och jämför dem med liknande moduler. Sedan beskrivs nyttan av att använda ramverk och MVC-designmönstrets grundidé presenteras. Planering av en webbapplikation, kravspecifikationens uppställning och uppdelningen av applikationen i MVC-designmönstret beskrivs. CodeIgniters tilläggsmoduler och sessionshantering presenteras. Dessutom förklaras vad som behövs för att skapa dynamiskt innehåll på webbsidor samt hur databasdesign och databashantering sker med CodeIgniter. Även validering och testning av en webbapplikation beskrivs. Svårigheter inom webbapplikationsutveckling och speciellt i skapandet av en webbapplikation med CodeIgniter samt när det lönar sig att använda ramverk dryftas.

Nyckelord:	applikationsramverk, databaser, dataöverföring, HTML, JavaScript, PHP, programmering, programutveckling, programvarudesign, relationsdatabaser, webbsidor, öppen källkod, phpBB
Sidantal:	40
Språk:	Svenska
Datum för godkännande:	

DEGREE THESIS	
Arcada	
Degree Programme:	Information and Media Technology
Identification number:	5011
Author:	Daniel Sundgren
Title:	Programming a Web Application with the CodeIgniter Framework
Supervisor (Arcada):	Göran Pulkkis
Commissioned by:	Thomas Finne

Abstract:
This thesis is a description of how a web application is created using the CodeIgniter framework. It contains a description of all the modules and libraries needed to develop an independent web application that can be used in a medium-sized company. The focus is mainly on the functional part. The graphical user interface is not described. All technical solutions are described and compared to other similar tools and techniques. The MVC design pattern is introduced and the benefits of using it. A development plan and a technical specification is described. How to set up CodeIgniter and how all modules are used in implementation of an application is thoroughly discussed. The three components in MVC are described as well as the usage of them. Setting up dynamic content on webpages as well as the design of relational databases and database management via CodeIgniter is explained. Testing and validation of a web application, problems that arise during the development stage, and under which circumstances a framework should be chosen for web application development is discussed.

Keywords:	applicationframework, databases, data processing, HTML, JavaScript, PHP, programming, application development, computer system design, relationaldatabases, webpages, open sourcecode, phpBB
Number of pages:	40
Language:	Swedish
Date of acceptance:	

INNEHÅLL

Figurer

Använda termer och förkortningar

- ACID = Atomicity, Consistency, Isolation, Durability
- Ajax = Asynchronous JavaScript and XML
- CDN = Content Delivery Network
- CSS = Custom Style Sheet
- CSV = Comma Separated Values
- Datatables.net = JQuery insticksmodul som förbättrar HTML-tabeller
- HTML = Hyper-Text Markup Language
- i18n = Internationalization
- ionAuth = Autentiseringsbibliotek för CodeIgniter
- JavaScript = Dynamiskt skriptspråk som används för att skapa funktioner på webbläsare
- JSON = JavaScript Object Notation
- jQuery = Bibliotek för att förenkla modifikation av HTML, CSS, DOM och Ajax-funktioner
- MVC = Model View Controller
- MySQL = Databashanterare med öppen källkod använder frågespråket SQL
- PHP = Hypertext Preprocessor, skriptspråk som körs på webservrar
- PhpBB = Diskussionsforumsprogramvara med öppen källkod
- phpMyAdmin = Program för att administrera databaser och tabeller via webbläsaren
- UML = Unified Modelling Language

1 INLEDNING

Detta examensarbete är en uppföljning på det som jag gjorde under min praktik som jag utförde för Arcada på deltid under vintern 2013 och sedan jobbade jag fulltid sommaren 2014. Uppdragsgivaren består av några Arcadas lärare, speciellt läraren i International Business Thomas Finne. Finne har gett idéer och hela konceptet är till stor del hans idé. Webbapplikationen är en prototyp som fungerar som ett verktyg för företag. Iden är att prototypen skulle fungera som en demoversion till en storskaligare applikation. Examensarbetet är en liten inblick i hur en webbapplikation programmeras med CodeIgniter som bas och hur jag upplevde att jobba i ett IT-projekt.

1.1 Bakgrund

Jag startade i slutet av 2013 ett projekt där min uppgift var att skapa en webbapplikationsprototyp. Själva projektets idé var att skapa en prototyp som skall fungera som ett webbverktyg för medelstora och stora företag. Webbapplikationen skall i företaget användas av en kärngrupp som består av olika människor med olika kunskap om företaget. Det var fullständigt nytt för mig att skapa en webbapplikation av denna skala på egen hand. När det här arbetet görs har prototypen gjorts klar redan men den väntar fortfarande på vidareutveckling eller på en slutlig version.

Trots att CodeIgniter är redan ett gammalt ramverk är dess dokumentation bra och många vanliga problem med detta ramverk har behandlats på nätet. Till CodeIgniter finns det många bibliotek och insticksmoduler som skapats av frivilliga. Detta gör det enkelt och effektivt att anpassa delar för just en specifik uppgift i en applikation.

1.2 Syfte och mål

Syftet med arbetet är att ge en inblick i vad som behövs för att skapa en webbapplikation där betoningen klart ligger på programmerandet av backend-delen. Även funktioner i användargränssnittet m.m. tas upp. Detta arbete kunde fungera som en guide till andra som tänker skapa en webbsida med dynamiskt innehåll med ett PHP-ramverk. Jag antar att läsaren har en viss grundkunskap inom informationsteknik för att förstå texten. Mera främmande termer förklaras kort i en liten lista som finns i början av detta arbete.

Målet är att få en bättre och sammanhängande text om de olika momenten i skapandet av en webbapplikation och om vad som behövs för att få CodeIgniter-ramverket att fungera i en dynamisk webbapplikation. Behandling av relationsdatabasens planering och konstruktion tas upp men behandlas inte utförligt. Jag behandlar också till en del hur man skall gå tillväga när man jobbar med ett IT-projekt. Jag kommer att berätta om de olika insticksmoduler som jag använt. De är sådana som inte behandlas frekvent. Information om dem finns huvudsakligen i modulernas egen dokumentation. Dels kan man även hitta information om dem på bloggar och diskussionsforum.

För att detta projekt är sekretessbelagt med tanke på själva webbapplikationens idé och funktion kommer det inte att handla om hurudan data matas in av slutanvändaren eller om själva processen i webbapplikationen.

Målet med just detta arbete är inte jordskakande men jag vill ändå svara på följande frågor:

- Hur skapar man en webbapplikation med dynamiskt innehåll med CodeIgniter?
- Vad behövs det för att få CodeIgniter att fungera med inloggningar och sessioner?
- Hur går man tillväga att skapa dynamiska webbsidor med flera olika språk?

1.3 Avgränsning

Eftersom själva webbapplikationen är sekretessbelagd behandlar jag inte det grafiska användargränssnittet eller informationsflödet i webbapplikationen. Jag kommer inte heller att behandla planeringen och implementeringen av webbsidornas grafiska gränssnitt, dels för att det skulle bli för omfattande och dels för att design inte är min starka sida. Jag låter bli att berätta om webbens historia samt om grunderna i HTML- och PHP-programmering.

2 TEORETISK BAKGRUND

För att skapa en fungerande webbapplikation, behövs en hel del olika verktyg trots relativt enkla funktioner och krav. I följande kapitel beskrivs de verktyg och program som jag övervägde att använda och de som jag sedan använde i skapandet av denna webbapplikation. En del program gjorde det lättare att arbeta och andra var nödvändiga för att få den önskade funktionaliteten. Till största delen kunde jag bestämma vad webbapplikationen bygger på. För att grunden var redan bestämd fanns det dock vissa begränsningar som jag måste anpassa utvecklingen till.

2.1 Server och grund till applikationen

Som grund till hela applikationen står företaget Nettihotelli Internet Oy. Enligt Nettihotelli har de servrar belägna inom goda förbindelser i Otnäs, Esbo och Drumsö, Helsingfors (Nettihotelli Internet Oy 2015). Servrarna kör operativsystemet Debian och har en Nginx webbserver. Redan

i tidigt skede var det klart att en databas behövs för denna applikation. En domänadress fanns redan färdigt och en tidigare prototyp av samma koncept under Nettihotellis tjänster. Denna webbhotelltjänst var dock den enklaste versionen som kallas ”Mini” (minsta). För att få i bruk en databas måste denna version uppgraderas till Nettihotellis version, som kallas ”Huone” (rum). Detta innebär 1 GB lagringsutrymme, 20 Gb/mån webbtrafik, 5 epostkonton och en databas MySQL eller PostgreSQL. Detta krävde bara ett mail till kundtjänsten på Nettihotelli och inom nästa dag var den nya tjänsten klar att användas. Till alla Nettihotellis tjänster hör som standard egna felmeddelandesidor, spamfilter för epost, ftp, PHP 5.4, phpmyadmin osv.

2.2 Val av databashanterare

I och med att det fanns endast två databashanterare att välja mellan var valet enkelt, MySQL. Båda två har öppen källkod dvs. de är gratis att använda och utveckla under förutsättningen att den utvecklade versionen också har samma definition för öppen källkod (Open Source Initiative 2015). En avgörande faktor var att jag i Arcada hade jobbat med MySQL, varvid specifika kommandon och phpmyadmin som verktyg är bekanta, vilket skulle medföra mindre problem under utvecklingsskedet.

Enligt Solid IT:s beräkningar har Oracles öppna källkodsvariant MySQL andra plats (24.2.2015) när PostgreSQL ligger på femte plats (24.2.2015). Samtidigt kan nämnas att Microsofts SQL Server ligger på tredje plats på listan (Solid IT 2015). Listan som Solid IT har gjort innehåller hundratals databashanterare så det finns alternativ att välja mellan. Som standard bjuder Nettihotelli dock bara på MySQL och PostgreSQL.

MySQL har som fördelar att den är driftsäker och mycket optimerad, har enkla säkerhetskopieringsmöjligheter och är flexibel för olika sorters datatyper. (Harkins & Reid 2001)

Båda databashanterarna är bra som sådana och de skiljer sig mera i detaljer och tillägg. Det handlar i stort om personliga preferenser och om behov av olika funktionaliteter. MySQL är snabbare och fungerar därför bättre i en webbapplikation (Hunter 2002).

2.2.1 MySQL

MySQL är som redan tidigare nämnt den mest använda öppna källkodens databashanterare. Olika versioner av MySQL har laddats ner eller distribuerats under dess livstid i 100 miljoner exemplar (Oracle 2015). Användarbasen som sådan har varit redan länge enorm och håller sin position stadigt (Solid IT 2015).

MySQL används i de största och mest framgångsrika webbsidorna och i företag runtom i världen bl.a. Google, Yahoo och Wikipedia. Den är också en del av den största öppna källkodens mjukvarupaket LAMP som består av Linux, Apache och PHP/Python/Perl. MySQL är känt för sin snabbhet, pålitlighet och användbarhet. (Oracle 2015)

2.2.2 PostgreSQL

Likt MySQL har PostgreSQL sina rötter redan under ett tidigt 1980-tal och en första version kom ut likväl år 1995. Postgre-projektet startades av amerikanaren Michael Stonebraker redan år 1986. (The PostgreSQL Global Development Group 2015b)

Databashanteraren är känd för att ha en bra datasäkerhet och för att vara pålitlig. Den fungerar på de kändaste operativsystemen och är likväl som MySQL fullt ACID-kompatibelt. PostgreSQL stöder förutom de vanligaste datatyperna också bild, ljud och video. (The PostgreSQL Global Development Group 2015a)

Jämfört med MySQL så framhäver PostgreSQL mycket mera om sina egenskaper på sina egna sidor. Där som man knappt får veta något om MySQL på Oracles webbsida utan att gå in i dokumentationen, så får man veta t.ex. vilka datatyper som stöds av PostgreSQL osv. Kanske Oracle inte har behov att marknadsföra sitt system där användarbasen redan är så stor. Där fungerar gratisversionen MySQL som en demo-version. Till dem som t.ex. kräver ännu mera funktioner kommer Oracle-databashanteraren med i bilden.

2.3 Applikationsramverk

Applikationsramverk är ett applikationsbibliotek med färdigbyggda komponenter för att bygga andra applikationer. I projektet fick jag rådet att använda ett ramverk för att underlätta utvecklandet av applikationen. Jag visste inte om ramverk så mycket men hittade ett intressant ramverk att lära mig använda.

Ramverket fungerar som hörnstenarna i applikationen och som ensamt skulle applikationen inte fungera utan ramverket. Ramverk används både till frontend- och till backend-utveckling. Meningen med ramverk är att minska på de problem som vanligen uppstår under applikationsutvecklingen. Med ramverk kan man göra utvecklingen av applikationer snabbare samtidigt som felen minimeras. Tack vare komponenterna kan programkod användas på nytt istället för att man duplicerar samma saker. Applikationen blir logiskt strukturerad så att underhåll och vidareutveckling kan ske smärtfritt. (Janssen 2015)

Applikationsramverk är ingen ny uppfinning, Apple skapade redan år 1985 ett av de första kommersiella ramverken, MacApp, som används ännu idag. På detta objekt-orienterade ramverk bygger också t.o.m. de första versionerna av Adobe Photoshop. (Feiler 2012, s.290)

För att hitta ett bra ramverk för utveckling, specifikt med PHP, krävdes lite undersökning, kanske just för att det finns en hel del öppna källkodens ramverk att välja mellan. Jag snubblade på CakePHP som verkade vara ett bra alternativ då och jag var ganska säker på att jag ville använda det. Senare hittade jag en webbsida (phpframework.com) som jämför de populäraste PHP-ramverken och där man enkelt kan värdera deras specifika egenskaper. Högt på listan fanns CodeIgniter som hade mycket röster och bra recensioner (Phpframeworks.com 2013). Det viktigaste för mig som nybörjare var att hitta ett ramverk med en stor användarbas, med bra dokumentation och med enkla utvecklingsmöjligheter. CodeIgniter verkade fylla mina krav så jag tog chansen att börja använda detta ramverk till webbapplikationen.

2.3.1 CodeIgniter

CodeIgniter är ett ramverk för att bygga dynamiska webbsidor, applikationer med serverskriptspråket PHP. CodeIgniter är i enkelhet ett paket med filer och kataloger som man flyttar över på servern. I rotkatalogen finns tre underkataloger, index.php-filen och en licenstext. De tre underkatalogerna är *application, system* och *user_guide*. Den som man använder mest är *application*. Via index.php-filen kommer man upp till en konfigureringsassistent som till en början innehåller felmeddelanden och hur man skall rätta till dem. I detta skede skall man öppna *application/config/config.php* och *application/config/database.php.* Dessa filer behöver redigeras enligt egna uppgifter. Efter att allt är konfigurerat kan man börja skapa den första sidan till sin applikation.

CodeIgniters första offentliga version kom ut 26 februari år 2006 av Rick Ellis. Den nyaste versionen kom ut i januari 2015 och en ny version är under utvekling som bäst. År 2008 blev CodeIgniter känd när Rasmus Ledorf, grundaren av scriptspråket PHP, berömde ramverket för att vara snabbare, lättare och minst som ett ramverk i jämförelse med de andra PHP-ramverken. (EllisLab 2015)

Ramverket har blivit känt under sin 7-åriga livstid för att vara kompakt, säkert och snabbt. Det är dessutom kompatibelt med både äldre som nyare tekniker. CodeIgniter används av både små och stora företag såsom Amerikas största teleoperatör AT&T och Dictionary.com samt som hörnsten i mobilapplikationer. (EllisLab 2013)

Företaget Ellislab, som fått sitt namn av grundaren Rick Ellis, beslöt år 2014 att sälja programmet. Orsaken var att företaget inte hade resurser eller passionen att fortsätta projektet med den entusiasm som det kräver. CodeIgniter passade inte in i företagets mål mer. Man ville inte över-

låta uppgiften till vem som helst utan till någon som håller projektet CodeIgniter i samma riktning som tidigare och dessutom har resurser att hålla upp det. Nu sköts CodeIgniter av British Columbia Institute of Technology (BIT) i Kanada. Där utbildas nu studerande att använda CodeIgniter. (EllisLab 2014b)

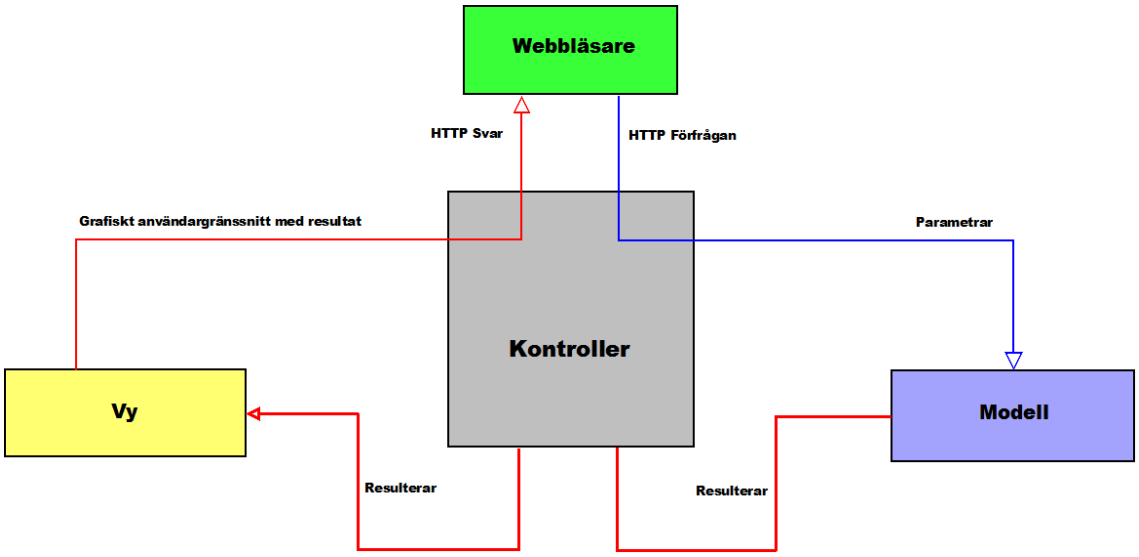
För slutanvändaren inverkar detta knappast på något annat sätt än att webbsidorna har flyttat över till CodeIgniter.com istället. Detta kan ju nog innebära förändringar i framtiden.

2.3.2 Model-View-Controller

De flesta ramverk baserar sig på MVC-standarden, där M står för datamodell (Model), V för presentationsvy (View) och C för kontroll (Controller) i enlighet med Conelly Barnes' definition av MVC: ”The model is the data, the view is the window on the screen, and the controller is the glue between the two” (Cunningham & Cunningham, Inc 2014). Likt de flesta ramverk baserar sig CodeIgniter också till en del på principen om MVC.

MVC fick sin början 1978 av norrmannen Trygve Reenskaug. Han utvecklade det första MVC-designmönstret till ett programs grafiska användargränssnitt med programmeringsspråket Smalltalk. Iden var att användaren skulle enkelt kunna granska och förändra bara den relevanta informationen för användaren. Reenskaug byggde ett mönster som representerar användarens mentala tankesätt, det skulle vara så gott som självklart att använda och bara det nödvändiga skulle vara synligt. Däremot det som inte är av intresse för användaren skulle inte vara synligt. (Reenskaug 2003)

Orsaken till uppdelningen till MVC kommer från det att först dela objekt i data och i en utgivare (Editor). Detta görs för att som tidigare nämnts ville man separera för användaren relevanta delar från resten av en mjukvara. Sedan märkte Reenskaug att inmatning av data och utdata inte hör ihop och är mycket olika. Han beslöt för att dela utgivaren i en vy som sköter om framställningen och i en kontroller som tolkar användarens inmatning. Se Figur 1 för hur de olika delarna fungerar ihop. (Reenskaug 2003)



Figur 1. Funktionsprincipen för designmönstret MVC.

Nu hade Reenskaug tydligt uppfunnit en lösning till att strukturera ett användargränssnitt, med att dela det i tre skilda delar som har sina egna roller. Den första är modell som står för strukturen och representerar användarens ”mentala modell” enligt Reenskaug. Den andra är vyn, den visuella representationen som hämtar data från ett eller flera modeller då den behöver uppdateras. Den sista är en kontroll som tar emot och tolkar data som matas in av användaren. (Reenskaug 2003)

Då man programmerar webbapplikationer är det vanligt att blanda in kod skapad med PHP eller motsvarande programmeringsspråk för databaskopplingar osv. Det är en bra vana att dela upp programkoden från själva användargränssnittet men det kräver arbete. Att skriva skilda filer för PHP som sedan körs i början av ett HTML-dokument är helt enkelt jobbigt. Med MVC-arkitekturen kommer uppdelningen automatiskt. Då behöver man inte duplicera lika mycket kod när man kan använda samma datamodell i många olika vyer. Om man gör förändringar i databasen eller byter databas fullständigt behöver man vanligtvis bara ändra på modellklassen. Samma sak gäller för kontroll- och vyfilerna. Kontrolldelen, som kan ses som en länk mellan vyerna och modellerna, fungerar enligt användarens önskemål och söker fram rätt modell och rätt vy. Man kan se MVC som tre olika, skilt för sig arbetande element som inte känner till detaljerna i varandras funktioner. Detta kan man ju också se som en negativ aspekt. (Kotek 2002)

När jag sökte efter lämpliga ramverk till projektet träffade jag ofta på debatter om det lönar sig för nybörjare att börja använda ett ramverk. Många tyckte att man först skall vara mycket bra på PHP innan man börjar med ramverk. Samtidigt underlättar dock ramverket, då man försöker få funktioner att fungera.

Att programmera i ett MVC-ramverk kräver mera planering och för med sig en större komplexitet än en enkel PHP-fil i en HTML-sida. När man separerar en sidas innehåll i minst två olika filer uppstår problem i att hitta felen i programmet. När man har mycket filer kan det bli jobbigt att hålla reda på alla filer i programmet. Det lönar sig att upprätthålla ett ordentligt system för namngivningen så att man hittar respektive vyers kontrollfiler osv. MVC är onödigt för små och kanske också för vissa medelstora applikationer. Mödan att starta en mindre applikation med en MVC-arkitektur kan helt enkelt vara för stor. (Kotek 2002)

2.4 JavaScript

För vissa funktioner var jag tvungen att använda ett klientbaserat programmeringsspråk. Klientbasering innebär funktioner som exekveras i webbläsaren utan att webbsidan behöver laddas på nytt och därmed t.ex. förstöra något som användaren har gjort på sidan. JavaScipt kan triggas med t.ex. en musklickning, ett tangenttryck eller om användaren lämnar sidan.

JavaScript skapades år 1995 på 10 dagar av amerikanaren Brendan Eich som jobbade på Netscape Communications Corporation. JavaScript blev en del av webbläsaren som sedan kom att heta Mozilla Firefox när Netscape beslöt att göra webbläsaren till ett öppet källkodsprojekt. Till en början kallade man JavaScript för Mocha sedan bytte man namn till LiveScript. I och med att Sun Microsystems var inkopplad i projektet gav man denna nya teknik namnet JavaScript, dels för att Java som programmeringsspråk var så populärt då. Två år senare gjordes JavaScript till en standard så att det kunde implementeras även i andra webbläsare. År 2005 började man skapa bibliotek med öppen källkod vilka använde JavaScript som bas. Hit hör sådana bibliotek som jQuery och Dojo. Med dessa kan man skapa dynamiska applikationer med ännu mera funktioner utan att webbsidor behöver omladdas fullständigt. (W3C 2012)

Enligt van der Schee (2013) finns det många orsaker att inte använda JavaScript. Speciellt ofta kom jag fram till det att många anser att JavaScript inte skall vara det som håller upp en webbsida utan skall mera förbättra upplevelsen på webbsidan. Mera om detta beskrivs i ett senare avsnitt. JavaScript fungerar enligt van der Schee (2013) inte så bra på en mobil, vilket är något att hålla i minnet. Enligt vad jag har prövat, fungerar de enklare egenskaperna bra och rätt så snabbt med en rätt modern mobil. Andra egenskaper som kan försämras är säkerheten, JavaScript kan orsaka mera sårbarheter, utvecklingstiden kan växa och funktionstestningen tar mer tid (van der Schee 2013).

Alternativen till klientbaserade programmeringsspråk är inte många och de kan nästan inte jämföras med JavaScript. JavaScript är överlägset mest använt (Q-Success 2015a).

2.4.1 jQuery

JQuery är ett JavaScript-bibliotek som gör funktioner och animeringar mycket enklare och snabbare än med att använda HTML och Ajax. Eftersom olika webbläsare inte fungerar på samma sätt kan man med jQuery få funktionerna att fungera så lika som möjligt i olika webbläsare. (The jQuery Foundation 2015)

JavaScript-bibliotekets viktigaste komponent är en fil med JavaScriptkod. För att använda jQuery behöver man endast ladda ner den minimerade versionen med storleken ca 83 KB och/eller länka den på sitt HTML-sidhuvud.

Det finns en del alternativa JavaScript-bibliotek till jQuery som tillsvidare dock är en marknadsledare. Hela 62,9% av alla webbsidor använder jQuery, när däremot 33,8% använder inte någon alls (Q-Success 2015b). De två andra som är näst mest använda är Modernizr och Bootstrap. Funktionsidén är lite lika mellan dessa men jQuery är den mest kompletta och avancerade. Bootstrap är ett bibliotek för responsiv webbdesign och mobilanvändning. Det fanns önskemål av uppdragsgivaren att göra webbsidan responsiv på mobilen, men tyvärr fanns det inte tillräckligt med tid för mig att bekanta mig med Bootstrap. Modernizer igen strävar till att göra CSS-användning enklare så att CSS fungerar på olika webbläsare på samma sätt.

JQuery är likt JavaScript en krydda för vissa funktioner på webbsidan. Det sägs att webbsidan inte skall vara beroende av JavaScript. För att uppfylla vissa krav som ställdes på webbapplikationen måste jQuery användas i detta projekt.

2.5 Insticksmoduler

En insticksmodul (eng. plug-in) är en mjukvara som ger mer funktioner till en annan mjukvara som denna insticksmodul installeras i. Man använder insticksmoduler i t.ex. webbläsare för att ge dem olika extrafunktionalitet eller stöd för olika audio- och videoformat. Med insticksmodul kan man också referera till hårdvara. Man blandar ofta mellan insticksmodul och tilläggsprogram, där tilläggsprogram kan ses mer som en modifierad version av det ursprungliga programmets funktion. (The Computer Language Company Inc. 2015)

2.5.1 ionAuth

Eftersom CodeIgniter saknar en inloggningsfunktion och för att webbapplikationen skulle ha en begränsad användargrupp behövdes ett autentiseringsbibliotek. IonAuth är ett autentiseringsbibliotek för CodeIgniter-ramverket. IonAuth är ett kompakt och lättanvänt bibliotek (Edmunds 2010), som har en katalogstruktur som är densamma som i CodeIgniter. Man behöver

bara kopiera filerna till respektive kataloger i ramverkets katalogstruktur för att börja använda det.

Amerikanaren Ben Edmunds jobbade med CodeIgniter-ramverket en tid och ville förbättra det dåvarande autentiseringsbiblioteket Redux Authentication 2. Han tyckte det var bra men det hade fel och saknade viss funktionalitet som ingen fixade så snabbt som han ville. Edmunds, med hjälp av några andra, omstrukturerade koden i programmet Redux Auth och satte till nya funktioner. (Edmunds 2015)

IonAuth innehåller det mesta som man kan behöva när det gäller användarhantering. Det första som man tänker på är in- och utloggning av en användare. För användarhantering krävs dock även en hel del annat. För att logga in måste man kunna skapa profiler. Tack vare att människan är glömsk till sin natur måste man kunna få lösenordet utbytt via sin e-post. IonAuth kan skilja på administratörer och vanliga användare osv.

2.5.2 I18n

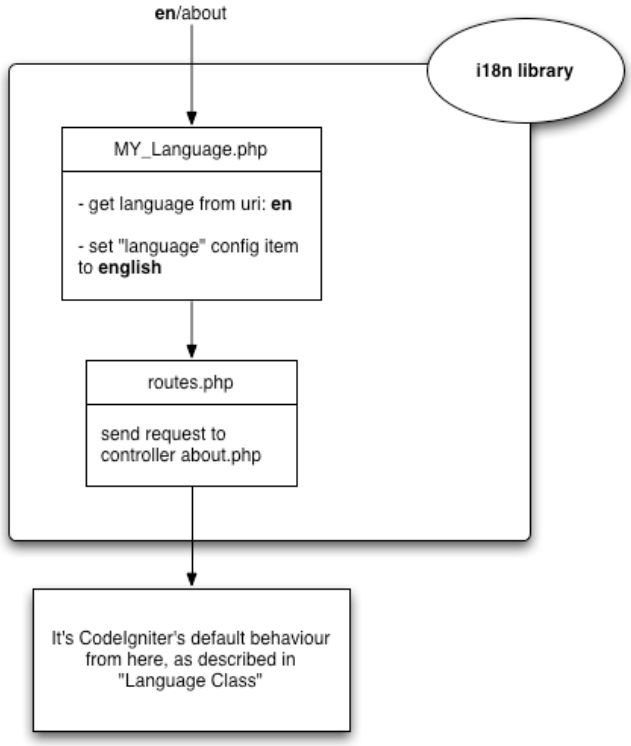
CodeIgniter 2.1 Internationalization i18n är utvecklad av Jérôme Jaglale med några andra utvecklares hjälp. Enligt Jaglale använder i18n CodeIgniters språk-klass enligt det språk som är valt i webbadressen. (GitHub, Inc 2014)

I18n är en sorts förkortning på internationalisering (eng. internationalization). Begreppet kommer från att det är i engelskan 18 bokstäver mellan tecknen i och n. Samma stil används för lokalisering (i10n). Uppdragsgivarens tanke var att visa i applikationen att det finns möjlighet att byta språk på webbsidan. Jag hittade detta program och ville gärna pröva använda det i applikationen. I18n är ett instickspogram utvecklat för just CodeIgniter. Idén är att använda skilda språkfiler som väljs på basis av användarens språkval. Webbsidan hämtar det valda språkets filer och fyller webbsidan med de rätta meningarna på rätt språk. På webbsidan programmerar man vanliga utskrifter av variabler på rätta ställen dit texten borde komma. För att ta i bruk i18n behöver man språkfiler. Engelska språkets filer för interna texter t.ex. formulärvalidering finns med som förhandsinställning i CodeIgniter. Det finns språkfiler till många olika språk att ladda ner via Github.com och övrigt på nätet för de mest allmänna funktionerna. T.ex. för ionAuth fanns det en finsk språkfil men ingen på svenska.

Språkfilerna är lagrade i *language*-katalogerna. Ramverket söker automatiskt efter språkfiler i katalogerna *application/language/* och *system/language/*. Varje använt språk skall ha en egen katalog i *language*-katalogen. För att internationaliseringen skall fungera måste språkfilerna ha ändelsen *_lang.php* för att igenkännas av systemet. Det rekommenderas att man som variabelnamn för varje mening i språkfilerna använder ett vettigt och enligt sammanhanget lämpligt namn med ett understreck och språkfilens namn. På det sättet undviker man samma namn för olika filer varvid krockar mellan två olika textsträngar med samma variabelnamn hindras. Om

man använder en formulärvalidering och filen heter t.ex. *validation_lang.php*, skulle variabeln för saknad av efternamn kunna heta *efternamn_saknas_validation.* På detta sätt kan man lättare göra felsökningar eller förändringar. I språkfilen skulle meningen kunna skrivas ut i stil med *$lang['efternamn_saknas_validation'] = 'Efternamnet saknas'*.

För att få språkfilerna att fungera måste de först laddas. I kontrollen sker denna förberedelse med koden *$this->lang->load('filnamn', 'språk').* Detta görs för varje språkfil som används. Alternativt kan man ladda dem automatiskt via *application/config/autoload.php.* Filnamnet skall skrivas utan ändelsen *_lang.php*. Man skulle som filnamn bara skriva *validation* för att enligt tidigare exempel ladda filen validation_lang.php. Som språk används det valda språket. Om man lämnar språket tomt används standardspråket som väljs i ramverkets konfigurering. På en webbsida utskrivs en textsträng med *echo $this->lang->line('variabelnamn').* I18n-bibliotekets funktion visas i Figur 2. (Juskewycz 2013)



Figur 2. Funktionsprincipen hos insticksmodulen i18n (Jaglale 2009)

2.5.3 DataTables

DataTables är ett projekt utvecklat med öppen källkod. DataTables är för jQuery en insticks-modul som är mycket flexibel och utvecklas hela tiden. Med DataTables får man nya funktioner och mer flexibilitet till HTML-tabeller. DataTables används av t.ex. Amazon.com, Los Angeles Times, Opelibrary.org och Cern.ch. (SpryMedia Ltd. 2015a)

Som relevanta funktioner kan räknas sökfunktion, sortering och möjlighet att dela tabellerna i sidor. Dessa är fullständigt valbara och man har möjlighet att anpassa DataTables på många olika sätt. Som förinställning innehåller DataTables en CSS-stilfil som man kan använda som bas.

Man hävdar på DataTables-webbsidan att man får insticksmodulen att fungera på sin webbsida på två minuter. Det kan stämma men för en oerfaren tar det längre. Naturligtvis måste man använda jQuery för att detta skall fungera. För att få DataTables att fungera behöver man en JavaScript-fil och en CSS-fil. Dessa filer kan laddas ner och länkas på webbsidan så att de laddas från servern varje gång sidhuvudet laddas. Alternativt kan man använda en CDN som DataTables erbjuder. För att sätta igång måste man förstås ha en rätt formaterad och fungerande HTML-tabell. Tabellen måste i alla fall ha elementen sidhuvud (thead) och body (tbody). På varje sida som DataTables används måste den initialiseras med en liten JavaScript-kod, se Figur 3.

```
$(document).ready(function() {
    $('#table_id').DataTable();
} );
```

Figur 3. Aktivering av jQuery-insticksmodulen DataTables.

Exemplet i Figur 3 är en vanlig JavaScript-kod som startar skriptet först efter att sidan har laddats. Därefter anger man vilken tabell skall använda DataTables. Med endast dessa steg skall DataTables fungera på webbsidan med all funktionalitet som erbjuds. För att anpassa olika funktioner och stilar krävs dock lite mera arbete. (SpryMedia Ltd. 2015b)

2.5.4 phpBB

PhpBB är en gratis diskussionsforumsmjukvara med öppen källkod. Namnet PhpBB kommer från Bulletin Board som kan översättas till elektronisk anslagstavla. PHP i namnet kommer naturligtvis från att phpBB är skriven i programmeringsspråket PHP. För att någon chatt eller diskussionsforum skulle ingå i projektet beslöt vi efter att ha prövat några andra alternativa chatt-program att använda phpBB som är den mest använda diskussionsforumsplattformen (phpBB 2014b). PhpBB används av många populära diskussionsforum såsom Openoffice.org, Joomla, Debian och Mozilla Add-on (PhpBB 2014c).

PhpBB kan användas för att upprätthålla en diskussion med en grupp människor eller som en fullständig webbsida. Tack vare olika anpassningsmöjligheter kan man lätt och snabbt skapa en personlig webbsida med phpBB. (phpBB 2014a)

PhpBB innehåller allt man kunde önska sig från privata meddelanden, användarikoner (avatar), bilduppladdning, röstningar till administratorpaneler. Med phpBB kommer en index.php-fil som vägleder användaren genom installationen samt går igenom och granskar att vissa inställningar på webbsidan är rätta. Sedan måste man skapa en administrator med lösenord. Därefter gäller det bara att börja skapa kategorier och rubriker till det nya diskussionsforumet.

3 IMPLEMENTERING

Implementeringen började i ett tidigt skede med en grovplanering av en webbsida som kom att få den största rollen i hela webbapplikationen. Före det hade vi möten med uppdragsgivaren för att klargöra vad webbapplikationen skulle handla om och vilka funktioner den skulle ha. Jag började jobba enligt idéer och färdiga skisser. Efter att en version var färdig, diskuterade vi med uppdragsgivaren om den var lämplig eller om man borde ändra på vissa saker. Under hela projektets utvecklingsprocess hade vi dessutom regelbundna möten med uppdragsgivaren för att lösa problem och behandla nya idéer. Vi använde inte mycket tid på planering och endast databasdesignen krävde en del förarbete och den måste ändras ännu i ett sent utvecklingsskede.

3.1 Kravspecifikation för webbapplikationen

En kravspecifikation behöver inte vara alltför komplicerad. Kravspecifikationen är en beskrivning på en hur en applikation ser ut, fungerar och hur användaren samverkar med den. För att hålla en planerad tidsram och en på förhand överenskommen budget behövs kravspecifikationer. De hjälper också utvecklaren i att konkretisera applikationen, hjälper att hålla den på rätt spår och hindrar att det skapas funktioner som inte behövs. En vanlig orsak till att ett projekt försenas eller i värsta fall misslyckas är att kravspecifikationen är bristfällig eller otydlig. (Görling 2009 s.78-79)

Till en stor del fick jag fria händer till att göra så som jag ansåg var bäst för applikationens olika delar. Vissa saker var nog klara redan från början, de funktionella kraven var till en del klara men nya kom upp under utvecklingsskedet. Det handlade ofta om att jag antecknade ner krav som vi kom överens om. Sedan fick jag ställa förslag till vad som kunde tas med i webbapplikationen.

3.1.1 Funktionella krav

Förutom det vanliga som nästan alla webbsidor med lite mera funktionalitet innehåller dvs. en hemsida med inloggning och utloggningsmöjlighet, en sida som beskriver webbsidornas ägare och idén bakom webbsidorna fanns det en del mera speciella krav. För att webbapplikationen

är stängd för yttre besökare skulle det finnas möjlighet för en administrator att skapa och redigera användare osv. Användaren skall kunna lagra text och dessutom data i form av filer av olika typ. Texten som inmatas lagras i databasen och visas också på webbsidan efter inmatning beroende på sammanhanget. Textinmatning krävde en fullständig möjlighet att redigera innehållet i vissa tabeller via webbsidan. Data från olika tabeller skulle kunna exporteras som en CSV-fil etc. En möjlighet för diskussion mellan användare skulle finnas. En stor del av funktionerna på webbsidan bygger på insamling av information om användare och sedan uträkning av statistik från tabeller för visning i form av grafer. Det skulle också finnas en möjlighet att skicka e-post som ett formulär.

Alla krav var inte klara från början och pga. projektets prototypnatur växte kraven i och med att applikationen började ta form.

3.1.2 Icke-funktionella krav

Vi kom överens att om det behövs finansiering för olika kommersiellt licenserade program så ordnas det nog. Med jag tyckte att man klarar sig med öppna källkodsprogram i en webbapplikation av denna typ, speciellt då det finns så mycket bra och välkända program med öppen källkod att välja mellan. I projektet gällde KISS-metoden (Keep It Simple Stupid), allt skulle hållas relativt enkelt och stilrent. Webbapplikationen skulle trots det vara social och uppmuntrande att använda både med en mobil och med en dator.

3.2 Planering

En plan är en beskrivning på vad som skall göras, hur det skall utföras och hur lång tid det skall ta att göra det. Med hjälp av planen kan man organisera arbetet och förtydliga de mål som finns. Samtidigt kan man enklare se om och när problem uppstår och vad det medför i form av förändringar i tidtabellen och kravspecifikationen. (Görling 2009 s. 44)

Tidsestimering ses som en konst i sig och är krävande även för erfarna människor. Man glömmer också ofta att t.o.m. en effektiv arbetsdag sällan når upp till dryga 8 timmar. En estimering är mer en gissning än något annat. Syftet med estimering är att få en bild för både uppdragsgivaren som utvecklaren hur stort projektet kommer att bli. För att få en hållbar tidsestimering måste man ha en tillräcklig kravspecifikation. Man skall sträva till att i alla fall inte underestimera eftersom det kan leda till stora problem mellan uppdragsgivaren och utvecklaren. Tidsestimeringen blir allt noggrannare ju mer tid man sätter på den, men det beror på projektets storlek om det lönar sig att göra en bra estimering eller inte. (Görling 2009 s. 99-114)

Jag uppgjorde med hjälp av uppdragsgivaren en plan om funktioner och egenskaper som borde finnas med i webbapplikationen. Tack vare en tillfredsställande kravspecifikation kunde en plan

göras för att underlätta tidsestimeringen. Jag gjorde estimeringar av antalet timmar som varje ny egenskap skulle behöva för implementering. Sedan prioriterade vi varje egenskap så att de viktigaste skulle ges mera tid. Om tiden inte räcker lämnar man bort det som inte är relevant. Vissa saker drog ut på tiden och andra saker behövde mycket kortare tid att få att fungera. Trots det hjälpte planen att planera tidsanvändningen och se helheten som ännu skulle byggas upp.

3.2.1 Webbsidornas första version

Jag började med att planera och bolla med idéer för en relevant webbsida i webbapplikationen. Detta var en uppgift i en grafisk designkurs men som det brukar vara så blev det inte så mycket kvar av den ursprungliga iden. Som utgångsläge hade jag skisser att använda med idéer som uppdragsgivaren gjort på förhand. Jag började skapa en webbsida med menyer och en skild CSS-fil. Efter att grunden var klar kunde man lätt kopiera de huvudsakliga komponenterna till nästa webbsida. Vi kom överens att jag börjar med det visuella utan funktioner för att visa hur det kommer att se ut. Det var ett bra alternativ att först få grunden att fungera utan att behöva jobba med dataöverföring osv.

I detta skede hade inga webbsidor ännu funktionalitet, men i stort sätt var basidén med mycket av innehållet redan färdigt.

3.2.2 Relationsdatabasdesign

En databasdesignprocess är enligt Connolly och Begg (1995) en trestegsprocess: den konceptuella, den logiska och den fysiska designen. Den konceptuella designen innebär en bild av en modell över den typ av information som databasen kommer att innehålla. Den logiska databasdesignprocessen däremot konstrueras mha. den konceptuella designen med att skapa en datamodell som också tar i beaktande hurdan typ av design man kommer att använda, t.ex. en relationsdatabas och konstruktion av ett ER-diagram (Entity Relationship). Slutligen behandlas alla fysiska begränsningar i den fysiska databasdesignen. I processen skapar man en beskrivning på relationer, filtyper, lagringsutrymmets storlek och säkerhet etc. (Connolly & Begg 1995 s. 419)

Den fysiska designprocessen börjar med insamling av all information från den logiska designprocessen och med att skapa basrelationer mellan tabeller. Hit hör bl.a. primärnycklar och främmande nycklar samt tabellelementens datatyp, längd och förvalda värden. Man måste veta noggrant vad man vill åstadkomma med databasen, t.ex. skall hela personalens månatliga löner räknas ihop redan i tabellen eller räknas de alltid vid en databassökning. För att bestämma storleken som reserveras per datatyp måste man samtidigt veta ganska exakt hur mycket information lagras i tabeller. (Connolly & Begg 1995 s. 479-482)

Jag började med att planera ett dokument med att skriva behövliga tabeller och kolumner. Efter en grov plan började jag använda UML-programmet DIA. I DIA uppställde jag de olika tabellerna och satte in alla kolumner som jag trodde att skulle behövas i det skedet. I samma plan skisserade jag relationerna mellan tabellerna. Att få ihop ett ER-diagram var en utmaning. Relationerna mellan databaserna, privata nycklar och index var en krävande del. Efter att jag trodde att jag hade fått ER-diagrammet klart var nästa steg att flytta över allting till databasen. Jag använde MySQL Workbench till en del men hade lite problem att lära mig använda den på rätt sätt. Jag använde Workbench till att göra samma plan som med DIA och sedan försökte jag exekvera planen med ett SQL-kommando som skapade tabellerna med MySQL Workbench. Detta fick jag inte att fungera utan det blev något fel i kommandot trots att programmet själv skapat det. Till sist använde jag bara Workbench till att skapa SQL-filer som jag sedan måste modifiera så att phpMyAdmin kunde importera dem. Efter många försök och modifieringar fick jag phpMyAdmin att skapa tabeller med SQL-filerna. Skillnaden mellan Workbenchens skapade filer och det som passade in i phpMyAdmin var att det ena programmet använde citattecken för kolumnnamn och det andra programmet ville inte ha citattecken.

3.3 Migrering till CodeIgniter

Efter att jag hade fått klart en hel del sidor skulle jag börja överflytta dem till CodeIgniter. Webbsidorna hade redan nått nästan det slutliga utseendet men så gott som all funktionalitet saknades ännu. En sak som jag måste förändra var sättet som jag gjort för att länka sidorna med varandra. Jag hade använt relativa länkar i stil med */filnamn.php* men det fungerade inte för att CodeIgniter använder länkar utan .php-ändelse.

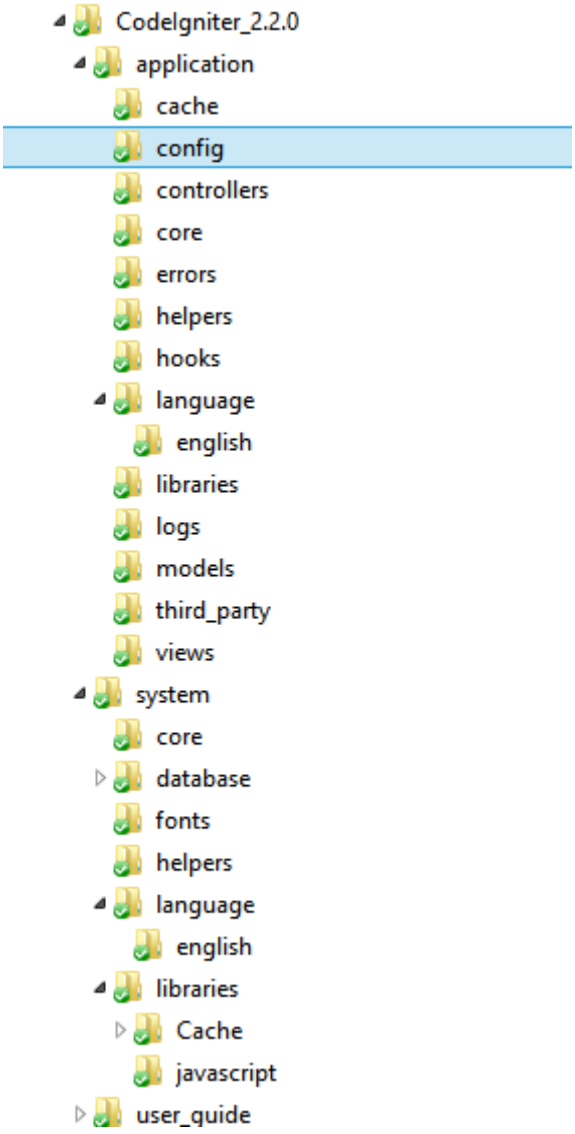
Förutom några små detaljer var det mest att klippa ut och klistra in i vyfiler det som jag hade gjort. Utan kontrollklasser får man ändå inte webbsidorna att fungera.

3.3.1 Katalogstruktur

CodeIgniter har en struktur som påminner om andra liknande ramverk eller bibliotek. Den består av tre kataloger: *application, system* och *user_guide*. Under utvecklingen använde jag inte den inbyggda lokala instruktionsguiden, jag glömde antagligen att den fanns överhuvudtaget. Samma guide finns på CodeIgniters hemsida, den använde jag frekvent.

Under *application*-katalogen finns de filer och kataloger som man behöver oftast tillgång till. Där finns katalogerna *models, controllers,* och *views*, vilka man arbetar mest med (se Figur 4). I *application*-katalogen finns också *logs*-katalogen, i vilken lagras filer i php-format med de olika fel som berör ramverket. Det är en bra vana att kolla katalogen alltid där när fel uppstår under utvecklingsprocessen. *Language*-katalogen innehåller språkfiler beroende på vilket språk man har valt till ramverket. Förutom *config*-katalogen är resten av mindre betydelse. *System-*

katalogen innehåller systemfiler som man inte behöver modifiera på något sätt, de fungerar som skelett för ramverket.



Figur 4. CodeIgniters katalogstruktur

3.3.2 Kontrollklasser och vyfiler

CodeIgniter använder sig av URL-segementering i MVC-stil. URL-segmentering fungerar så att t.ex. i adressen *exempel.fi/klass/funktion/ab* är klass namnet på en webbsida t.ex. sidan statistik och *funktion* är en funktion i kontrollen t.ex. *clear*. Med *ab* betecknas en eventuell variabel som flyttas från webbsidan till kontrollern.

Vyfilerna skapar den visuella presentationen på en webbsida vars beteende styrs av kontrollfilerna. Vyfilen är i all enkelhet en vanlig HTML-sida som inte behöver innehålla något annat. I regel kan man säga att varje webbsida har en kontroll men vyfilerna kan vara flera än så.

Kontrollklassen består av en PHP-klass som i mitt fall ärver och utvidgar en annan IonAuth-klass som hindrar åtkomst till webbsidan utan inloggning. I PHP-klassen finns en konstruktor med vilken man inladdar t.ex. modellklasser, formulärvalidering och andra inbyggda bibliotek före de andra funktionerna exekveras. Nästa funktion är en index-funktion som exekveras då man besöker webbsidan som kontrollen representerar. I sin enkelhet kan den bestå av endast en vyfil. I Figur 5 finns ett enkelt exempel på hur man skulle kunna ha denna indexfunktion på en ”About Us”-webbsida:

```
public function index() {

    $data['title'] = "About Us";
    $this->load->view('frontend/aboutpage_view', $data);

}
```

Figur 5. Index-funktionen för aboutpage-kontrollen

Variabeln data är en så kallad indexerad lista dit man kan lagra meningar etc. Istället för att använda platsnumror 0....n kan varje plats ha ett namn för lättare kontroll av listan. För att skriva ut ett specifikt index som finns i listan använder man *$data[’title’]*. I exemplet ovan skickar man allt som variabeln $data innehåller till webbsidan där det är fritt fram att använda listan på vilket sätt som helst. I detta fall innehåller variabeln endast titeln som man kan använda som webbsidans rubrik. Sista raden i exemplet laddar från katalogen *frontend* filen *about-page_view.php* som är en vanlig HTML-fil. CodeIgniter har försökt göra det lättare med att man inte behöver sätta ändelsen *.php* i koden men det kan även vara missledande att lämna bort något som i programmering ofta skall finnas i filnamnet.

Utöver den vanliga index-funktionen kan man ha funktioner av olika slag med uppgifter såsom att lagra eller hämta data från en databas osv. Till en stor del använde jag formulär för att interagera med användaren i form av olika datainsamlingar eller redigeringar av information. Dessa formulär är egentligen knappar som för användaren till undersidor. Som ett enkelt exempel kan nämnas en statistik-webbsida där administratorn har möjlighet att tömma sidans tabell. Med formuläret förs användaren till undersidan */statistik/clear*. Efter knappklickningen exekveras funktionen *clear* där metoden *clear_Table()* tömmer en tabell i databasen, se Figur 6. Vyn *clear_view* laddas sedan och den består endast av ett omdirigeringskommando (redirect) tillbaka till statistik-webbsidan. Före denna omdirigering exekveras metoden *clear_Table* och användaren ser bara en omladdning av webbsidan. Tidigare hade jag användaren att gå in på en ny webbsida som meddelade att åtgärden lyckades, men om man ville se statistik-webbsidan igen måste man navigera tillbaka till den. En omdirigering fungerar mycket bättre och avbryter inte användaren.

```php
public function clear() {

    $data['clear']=$this->statistics_model->clear_Table();
    $this->load->view('frontend/clear_view', $data);

}
```

Figur 6. En funktion som tömmer en tabell på statistik-webbsidans kontroll.

Alla kontrollklasser som representerar webbsidor med lite mer funktionalitet fungerar i stort sätt på samma sätt i webbapplikationen: en knapp etc. anropar en funktion i en modellklass varefter användaren förs till en webbsida som omdirigerar användaren tillbaka.

3.3.3 Modellklasser

Modellklasserna hör inte till de viktigaste komponenterna i ramverket för en webbsidas funktionalitet. De spelar trots det en stor roll i dynamiska webbapplikationer. Modellerna används för att kommunicera med databasen i applikationen. Med CodeIgniter kan man använda förenklade SQL-kommandon eller fullständiga SQL-satser för att manipulera tabeller, man kan använda sig av alla tänkbara SQL-kommandon.

En modellklass byggs upp på samma sätt som en kontrollklass. All programkod kommer innanför PHP-starttaggen, men en sluttagg kommer inte med i CodeIgniter-filerna. Alla modellklasser ärver CodeIgniters basmodell *CI_Model*. Klassnamnen skall alltid ha en stor begynnelsebokstav och övriga bokstäver skall vara små. Jag har skapat en *Team_model* vars uppgift är att lagra, redigera och hämta uppgifter om användare. I konstruktorn kan man ladda databasen med kommandot *$this->load->database()*. I konstruktorn blir databasen instansierad för alla funktioner i klassen. Detta kan göras också automatiskt i *application/config/autoload.php* beroende på om man använder databasen i många webbsidor.

```php
public function get_members() {

    $this->db->select('*');
    $this->db->from('users');
    $query = $this->db->get();

   return $query->result_array();
}
```

Figur 7. En modell-funktion som hämtad en användartabell.

En enkel get-metod som hämtar alla rader ur användartabellen visas i Figur 7. Med kommandot *select* specificerar man som vanligt vilka kolumner i tabellen tas med i SQL-förfrågningen, ’*’ betyder alla kolumner. Ordet *from* specifierar vilken tabell det är frågan om, enligt Figur 7 heter tabellen *users*. Databasförfrågningen initialiseras med *get*-funktionen och resultatet lagras i en variabel *query* i form av en lista. Funktionen *get_member* aktiveras i kontrollen med kommandot $data['users'] = $this->users_model->get_members(), som lagrar *query*-listan i *data*-listan med indexet *users*. För att använda tabellens innehåll hämtas listans innehåll i vyfilen. Ett sätt att göra det är att införa en *foreach*-slinga i vyfilen t.ex. enligt Figur 8.

```
foreach($users as $row):

    echo "<div>".$row['first_name']." ".$row['last_name']."</div> <br>";

endforeach;
```

Figur 8. Ett exempel på en slinga som itererar genom en användartabell och skriver ut förnamn och efternamn i en vyfil.

Slingan i Figur 8 radar alla förnamn och efternamn under varandra. För att specificera vilken kolumndata som kommer vart skall man använda samma kolumnnamn som i databasen, t.ex. *first_name*.

3.3.4 Mallar

Redan före CodeIgniter togs i bruk insåg jag att då man har ca 20 olika undersidor i webbapplikationen är det tidskrävande att ändra små saker som berör alla sidor. Ett sätt att komma över detta är att använda ramverk och med s.k. mallar (templates) för underlätta redigering av webbsidor. Med mallar behöver man ha endast en fil som fungerar som sidhuvud och kan användas på alla sidor. Mallar använde jag för både sidhuvud- och sidfot-sidor och därtill hade jag navigeringsmenyn som en mall. Det underlättade redigering av sidor, då man inte behövde göra förändringar i 20+ filer.

För att få en webbsida att se rätt ut måste de olika komponenterna laddas in i kontrollen i kronologisk ordning. Kontrollen skapar en HTML-sida av de laddade vyfilerna. Om man laddar sidfoten före sidhuvudet kommer sidfoten fysiskt att komma först i HTML-dokumentet. På samma sätt som en vy laddas av kontrollen skall även mallarnas vyfiler laddas. Figur 9 är ett exempel på hur kontrollen ser ut för webbapplikationens hemsida.

```php
public function index() {

    $this->load->view('templates/header');
    $this->load->view('frontend/home_view');
    $this->load->view('templates/footer');

}
```

Figur 9. Exempel på hur en hemsidas olika mallar laddas i en kontroll.

Mallarna har jag i en katalog med namnet *templates* för att skilja åt den från *frontend*-katalogen som innehåller de individuella vyfilerna. Efter att processen att dela upp en webbsida i vyfiler, mall- och kontrollklasser blev klart för mig gick det enkelt. Tidskrävande var det nog att först rensa bort det som var gemensamt med sidorna och sedan skapa kontroller för varje sida.

3.3.5 Sessioner

Sessionsklassen i CodeIgniter lagrar en användares tillstånd och aktivitet på webbsidan. Sessionslagring finns inbyggt i PHP men CodeIgniter använder en lagring som är mer flexibel. Man kan välja mellan att lagra sessionsdata som en *cookie* eller att lagra den i databasen. Sessionen kan även krypteras vid behov. Sessionsdata är egentligen en lista som innehåller användarens unika, tillfälliga ID, IP-adressen, webbläsarens identifikation och en tidsstämpel. Utöver detta kan man programmera egen sessionsdata i en lista. (EllisLab 2014a)

Orsaken till att jag måste använda sessionsdata var att jag hade på profilsidan skapat en möjlighet att ladda en profilbild. Filuppladdningen är en inbyggd funktion i ramverket. Själva bilden lagras på servern men filens sökväg lagras i användartabellen. Problemet var att om man gör ändringar i sin profil utan att byta profilbild blir fältet med bildens sökväg tomt för att den sätter in all information som finns i formulären varje gång man sparar. För att lösa problemet insatte jag en if-sats där bildens sökväg inte sätts i tabellen om sökvägen är tom. Detta fungerade delvis men beroende på hur man använder webbapplikationen kan profilbildens sökväg tömmas. För att lösa även detta problem måste jag lagra bildens sökväg som sessionsdata för att ha den lagrad tills användaren bestämmer att spara dvs. skicka all data till tabellen. Sessionsdata lagras med bildens uppladdningsprocess i kontrollen enligt Figur 10. Jag kunde använda en inbyggd funktion som lagrar information om bilden i en variabel under uppladdningsprocessen.

```php
$data = array('upload_data' => $this->upload->data());
$this->session->set_userdata($data);
```

Figur 10. Sessionslagring av uppladdningsdata.

Det var en utmaning att få sessionslagringen att fungera men till slut fick jag bildens sökväg lagrad i sessionen och endast den utskriven som i Figur 11. Bildens sökväg lagras på samma

sätt som all profildata med HTML-formulär, men skillnaden är att formuläret är gömt från användaren, samtidigt som värdet tas från listan som innehåller sessionsdata.

```
<input name='picurl' type='hidden' Value='".$sessionarray['upload_data']['file_name']."'>
```

Figur 11. Användning av sessionsdata i ett formulär.

3.4 IonAuth

För att börja använda ionAuth efter den s.k. installationen måste ionAuth laddas i *application/config/autoload.php* eller i konstruktorn av den kontroll där insticksmodulen används. Det finns mycket som man kan anpassa till egna preferenser t.ex. hur snabbt en användare loggas ut automatiskt eller hur bra kryptering används för lösenorden i databasen. För login surfar man till *exmpeladress.fi/login*. Färdiga mallar finns för alla funktioner. Jag ville inte associera webbapplikationen med stilar på inloggningsfönstret som syns för potentiellt vem som helst. Jag beslöt att bara använda ionAuths standardinloggningsformulär. För att logga ut en användare utförs kommandot *$this->ion_auth->logout()*. Eftersom det finns i ionAuth en kontroll med funktionen *logout* kan man logga ut en användare med */auth/logout*.

3.4.1 Åtkomst utan inloggning

Till webbapplikationen måste skapas en sida som gav åtkomst till en modul utan inloggning för ”gäster”, som inte hade åtkomst till resten av webbapplikationen. Sidan är likt en annan i webbapplikationen men jag kunde inte använda fysiskt samma sida utan måste skapa en version med färre funktioner. Kontrollen i Figur 12 saknar index-funktionens webbsida eftersom jag ville minska risken att någon besöker webbadressen av misstag. Avancerade användare, som tror att de genom att surfa till *exempel.fi/v7j2dwmdu0veoxq* skulle få åtkomst, dirigeras inte till inloggningssidan utan till en inbyggd felmeddelandesida. Kontrollen i Figur 12 har en konstuktor där en modellklass med vissa databasfunktioner laddas. Indexfunktionen ger felmeddelandet 404 medan *guest* öppnar en webbsida för gäster.

```php
<?php

class v7j2dwmdu0veoxq extends CI_Controller {

    function __construct() {
        parent::__construct();
        $this->load->model('guest_model');
     }

    public function index() {
    $this->load->view('templates/error');
    }

    public function guest() {
    $this->load->view('templates/headerrestrict', $data);
    $this->load->view('frontend/guest_view', $data);
    }
}
```

Figur 12. Del av en kontroll som jag programmerat för åtkomst utan inloggning.

3.4.2 Användare och användargrupper

Varje användare får en unik id. Därtill tilldelas alla användare en användarroll, antingen *user*, *facilitator* eller *administrator*. En admin har rättighet till alla funktioner som är skapade i programmet, facilitatorn har vissa rättigheter i applikationens interna dataadministration.

För att skilja åt olika användares rättigheter måste jag bygga olika vyer och i kontrollen specificera vilken användargrupp varje användare hör till. I kontrollen sker detta medan användaren klickar på länken till ifrågavarande sida. IonAuth har en metod för att kolla användarens roll. Med *$this->ion_auth->in_group("gruppnamn")* kan man kolla användarens grupp. Alternativt kan man kolla administratorrättigheter med *$this->ion_auth->is_admin().* Enligt Figur 13 har jag använt nämnda funktion för att ladda skilda vyer för olika användare med en if-sats i kontrollklassen för en profilsida. Skillnaden är att administratorn har rättighet att modifiera alla profiler medan en användare bara kan redigera sin egen profil.

IonAuth har en användartabell med endast få uppgifter. För webbapplikationen behövdes flere uppgifter så jag skapade en egen tabell istället för att modifiera den existerade tabellen. För att inte användaren skulle behöva själv fylla i sina uppgifter på två ställen införde jag i ionAuths programkod en funktion som använde samma formulär för personuppgifterna och insatte samma data i den nya tabellen.

```
public function index() {

    if($this->ion_auth->is_admin()) {
        $this->load->view('frontend/accountad_view')
    }
    else if($this->ion_auth->in_group("facilitator")){
        $this->load->view('frontend/accountfac_view')
    }
    else{
        $this->load->view('frontend/account_view')
        }
}
```

Figur 13. Implementationsexempel på rollfördelning av en profilsida som är unik enligt användargrupp.

```
    $additional_data = array(
        'first_name' => $this->input->post('first_name'),
        'last_name'  => $this->input->post('last_name'),
        'company'    => $this->input->post('company'),
        'phone'      => $this->input->post('phone'),
    );
}
if ($this->form_validation->run() == true && $this->ion_auth->register(
$username, $password, $email, $additional_data))
{
    $data['team'] = $this->team_model->set_members();

    $data['usersend'] = $this->account_model->send_notification();
    //check to see if we are creating the user
    //redirect them back to the admin page
    $this->session->set_flashdata('message', $this->ion_auth->messages());
    redirect("auth", 'refresh');
}
```

Figur 14. IonAuths egen kontroll auth.php med en markerad modifikation.

Funktionen som är markerad i Figur 14 är ett funktionsanrop som utnyttjar variabler som redan skapats för förnamn, efternamn, e-post, telefonnummer och företagsnamn. I modellen lagras de i den skapade användartabellen i exakt samma form.

3.5 Multispråkstöd

För att ge en bild att applikationen klarar av olika språk valdes de typiska språken engelska, svenska och finska att användas i webbapplikationen. Iden med biblioteket I18n är att språket syns i adressfältet som en förkortning. I detta fall skulle språken synas som en/swe/fin. För det måste man skapa en fil. I instruktionerna på webben finns en fil som man kan kopiera och modifiera enligt eget behov. Filen skall finnas i */application/core* och heter MY_Lang.php. Filen har som uppgift bl.a. att granska språket som används, byta språk och lägga till språket i webbadressen. Utöver detta skapar man på samma sätt en fil med namnet MY_Config.php.

Efter att filerna har skapats måste man till filen */application/config/routes.php* insätta några rader, som har att göra med adressen och att den fungerar trots att där finns en språkidentifiering. För att sedan ladda språkfilerna kan man antingen ladda språkfilerna på varje sida där de används eller så kan man använda en funktion, som laddar alla språkfiler. Funktionen i Figur 15 skall insättas i katalogen */application/hooks.*

```
function initialize() {
    $ci =& get_instance();
    $ci->load->helper('language');
    $site_lang = $ci->session->userdata('site_lang');

    if ($site_lang) {
        $ci->lang->load('form_validation', $site_lang);
    } else {
        $ci->lang->load('form_validation', 'english');
    }
}
```

Figur 15. En del av en funktion som laddar alla språkfiler.

Figur 15 som visar bara formulärvalideringens språkfil, men principen är densamma för alla språkfiler. Sessionen anger vilket språk användaren har valt och sedan laddar man språkfiler för det språket. Som standard väljs engelska språket om inget annat har valts.

3.5.1 Färdiga språkfiler samt översättning av filer

Det finns på Internet färdiga översatta språkfiler för svenska och finska men endast för autentiseringsbiblioteket ionAuth. Sedan hittade jag en svensk språkfil för formulärvalidering på en blogg. Formulärvalideringen är ett hjälpverktyg för användaren. Valideringen berättar för användaren vad denne har skrivit fel eller lämnat bort relevant information från formulären. Jag översatte formulärvalideringen till finska. De mesta måste göras från början, all text som finns på webbsidorna måste klippas ut och insättas i textfiler. Här var jag tvungen att skapa ett variabelnamn för olika textsträngar. Alla meningar som kommer efter varandra kan insättas till samma textsträng. Jag delade upp filerna så att t.ex. sidhuvudet och navigeringens texter kom i samma språkfil. Det beror förstås på hur mycket text man har men jag såg till att det inte skulle bli mer än en sida text per språkfil. Alla formulärs texter kom i samma fil, som jag gav namnet *formlabel_lang*.

3.5.2 Platshållare

För att kunna skapa språkfiler snabbare skapade jag enligt instruktioner en fil */application/helpers/MY-language_helper.php* som har som uppgift att ersätta en textsträng med en annan. Enligt Figur 16 är *%s* en s.k. platshållare (*placeholder*) som ersätts med en variabel i webbapplikationen.

```
$lang['valid_email']    =    "%s måste innehålla en giltig e-post.";
```

Figur 16. Exempel på en platshållare i en språkfil.

Platshållare används av formulärvalideringar. Annars använde jag inte platshållare i språkfiler. I kontrollen väljer man vilken data som skall valideras och vad platshållaren skall representera. I exemplet i Figur 17 väljs vilket fält skall valideras, i detta fall *email.* Nästa parameter berättar för platshållaren vad den skall ersättas med och sista parametern anger hur data skall valideras. Här granskas endast om det finns något i fältet *(required).*

```
$this->form_validation->set_rules('email', 'Email', 'required');
```

Figur 17. Exempel på en initialisering av en formulärvalidering.

3.5.3 Val av språk

Allt var klart efter att jag hade insatt en meny uppe på webbsidan där man kunde välja språk. Samtidigt som man ändrade språk skulle adressen e*xempel.fi/en/home* ändras. Om man valt svenska skulle adressen ändras till *exempel.fi/swe/home.* Jag fick inte detta att fungera. Problemet var kanske i sättet att välja språk. I sidhuvudet anropas funktionen *switchLanguage*, se Figur 18.

```
function switchLanguage($language = "") {
    $language = ($language != "") ? $language : "english";
    $this->session->set_userdata('site_lang', $language);
    $ref = $this->input->server('HTTP_REFERER', TRUE);
    redirect($ref, 'location');
}
```

Figur 18. Funktionen för byte av språk.

Funktionen *switchLanguage* skall byta språk och det gör den men fortfarande blir det */en/* i adressfältet för engelska. Hela multispråkstödet blev ett större projekt än vad jag förväntade, speciellt att skriva språkfilerna var en tidskrävande process. Eftersom hela språkstödet inte var så viktigt fick detaljen med språket i adressfältet förbli orättad.

3.6 Statistik

En viktig del i webbapplikationen var att få statistik på data som matats in av användare. Idén var att kunna följa upp resultaten för en noggrann analys. Därför implementerade jag en funkt-

ion som sparar grafer som bilder. I webbapplikationen finns en funktion där användare poängsätter (1…4 poäng) olika alternativ. Statistiken uträknades på basen av användares poängsättning som infördes i tabeller.

3.6.1 Val av graf

När behovet av grafer uppstod under våra möten med uppdragsgivaren tänkte jag genast på Google Charts. Googles alternativ var ett enkelt val för att dynamiskt och snabbt visa data som finns i databasen. Jag har i en kurs använt Google Charts som verkade riktigt enkelt att använda. Eftersom data som tas från tabeller för statistik ändras med tiden valde jag att räkna procentuella andelar och visa grafer på basen av de uträknade värdena varje gång webbsidan laddas. En annan möjlighet skulle ha varit att skapa en ny tabell med data i rätt form och bara utnyttja den informationen i grafen. Detta kändes dock som dubbelt arbete.

3.6.2 Datainsamling

Användarna fyller i en tabell där man måste välja en kategori. Den första grafen representerar dessa kategoriers fördelning. Eftersom kategorin är en skild tabell måste två tabeller paras ihop med en *join*-operation. I vyn uträknas sedan hur många rader av en specifik kategori finns i tabellen. Radantalen sparas i skilda variabler namngivna enligt kategori. I detta skede har jag lika många variabler som kategorier. Varje variabel innehåller antalet rader per specifik kategori. Problemet som jag först stötte på var att variablerna är i PHP-format och Google Charts använder JavaScript. Det var dock inte ett omöjligt problem, tack vare att PHP exekveras på servern och JavaScript på webbläsaren. Såsom visas i Figur 19 konverteras alla siffervärden till JavaScript-variabler. Därefter insätts variablerna i en lista med rubriker enligt Figur 20. Listan tas sedan med i funktionen som skapar ett cirkeldiagram. För att visuellt skilja åt olika kategorier har varje kategori en egen färg.

```
var categoryA = parseInt("<?php echo $categoryA; ?>");
```

Figur 19. Metoden som jag använde för att ändra PHP-variabel till JavaScript-format.

```
data.addRows([
    ['Kategorinamn A', categoryA],
    ['Kategorinamn B', categoryB]
]);
```

Figur 20. Insättning av värden med rubriker i en tvådimensionell lista.

För den andra grafen behövdes ett mellansteg som innebar anrop av en JSON-funktion. Denna graf är ett stapeldiagram som består av data inmatad av användare. Enligt en poängsättning får de inmatade datavärdena en procentuell fördelning. Antal poäng sparas i en tabell som är skild från övrig data. Kontrollen går igenom varje datarad som får den tillhörande poängsumman som värde. Se Figur 21.

```
$rows = 1+$this->stats_model->getrowcount();
        for($i = 1; $i < $rows; $i++)
{
    $data['data'.$i] = $this->stats_model->load_points($i);
}
```

Figur 21. Ett utklipp av en kontroll som skapar en lista enligt antal rader.

Sedan skapas två temporära listor, en för data och en för antal poäng. I datalistan med namnet *dataname* insätts rubriken och data med ett index-värde. I poänglistan med namnet *datavotes* sätts poängen med samma index-värde som *dataname*. Efter att båda listorna var i samma ordning måste jag konvertera dem till JavaScript. I Figur 22 visas tillvägagångssättet. Först måste numeriska värden hanteras med *array_map*-metoden som skapar en ny lista och konverterar numeriska värden till textsträngar. Sedan för jag över PHP-listorna till JSON-format där de sedan kan ges till vanliga JavaScript-variabler.

```
<?php

  $votes = array_map('intval', $datavotes);
 $jsdataarr = json_encode($datatname);
  $jspointsarr = json_encode($votes);

     echo "var Votes = ".$jsvotesarr.";\n";
     echo "var javaSdata = ".$jsdataarr.";\n";
?>
```

Figur 22. Principen för konvertering av numeriska värden (votes) och strängar (dataname) till JavaScript.

Google Charts har olika möjligheter för modifiering av grafers utseende, bl.a. kan man ändra på deras färg. För att få olika färger för varje diagram måste jag använda mig av en lista med färgvärden. Enligt Figur 23 insätts allt i en slinga som skapar en lista med alla datavärden, poänganatal och en färg per värde. Med Google Charts instruktioner är det sedan enkelt att skapa ett stapeldiagram för datat i Figur 23.

```
for(var i = 0; i < javaSdata.length; i++)
{
    data.addRow([javaSdata[i], Votes[i], colors[i] ]);
}
```

Figur 23. Adderar tre olika listor till grafens tabell.

Eftersom all statistik tas från databasen medan en webbsida laddas och sedan uträknas av webbläsaren tar det en längre stund att ladda webbsidan fullständigt. Jag anser ändå att det inte är något problem trots att det tar ca 2-4 s att ladda sidan. Det är dock nästan dubbelt mer än för en webbsida utan databas- eller JavaScript-funktioner.

3.7 Dokumentation

Dokumentation är en viktig del i applikationsutveckling. Under utvecklingsprocessen insatte jag kommentarer i programkoden, men jag märkte senare att sådan dokumentation var rätt bristfällig. Jag insatte för varje funktion en beskrivning av vad som kommer in i funktionen, vad den gör och vad den returnerar. Samtidigt började jag skriva en dokumentation som förklarar webbapplikationens funktionalitet och vad som behövs för att använda applikationen. Jag utgick i dokumentationen från att läsaren inte har en teknisk bakgrund. Jag skriver också i dokumentationen att den inte är riktad till programmerare, utan mera för att ge läsaren en bild av vad som gjorts samt var man hittar den information och de moduler som jag använt. Att göra en självständig dokumentation som innehåller information om alla källor som jag använt och om allt som jag har lärt mig är dock omöjligt.

Dokumentationen fick inte tillräcklig uppmärksamhet och jag insåg det allt för sent. Jag inser också att under detta examensarbete hade jag svårt att tolka hur jag programmerat vissa funktioner. Trots att de relevanta funktionerna har en del kommentarer är det svårt att hålla koll på allt när en helhet består av tre olika filer. Att någon skall ta över detta projekt känns skrämmande, för jag vet inte hur en annan programmerare med en helt annan bakgrund tolkar programkod skapad av mig. Programkoden som jag skapat har dels dock grunder från skolan där man försökt lära ut objektorienterad programmering. En erfaren programmerare kan tycka att min programkod ser enkel ut.

4 RESULTATREDOVISNING

Nu när webbapplikationen är färdig kan jag med gott samvete hävda att allt som har betydelse fungerar. Det som jag inte fick att fungera löste jag på ett annat sätt. Jag hade många problem med funktioner i frontenden. Jag löste problemen tack vare vänliga utvecklare på diskussionsforumet stackowerflow.com. Trots det hade jag ett annat problem som jag försökte lösa utan att lyckas, jag fick inte hjälp på nätet. Problemet var att i webbapplikationen skulle finnas en möjlighet att redigera fälten i en HTML-tabell före överföring till databasen. Det finns en insticksmodul *Jeditable* som jag försökte använda men jag fick den inte att fungera. Lösningen var att använda en liknande jQuery-dialog som jag redan använt i en annan del av applikationen. I jQuery-dialogen hämtas den valda radens data, sedan kan man redigera fritt och när man sparar lagras allt i databasen.

4.1 Testning och validering

Under utvecklingen använde jag dynamisk testning. Detta innebär att man testar webbapplikationen då den används. Jag behövde inte använda en testserver eftersom applikationen var lösenordsskyddad utan andra användare. Jag hade inte tid att stifta bekantskap med andra testmetoder. Testningen är en kontinuerlig process som kunde räcka lika länge som utvecklingen.

Jag försökte testa webbapplikationen efter att det mesta var klart men det blev inte så mycket tid över. Det är ett problem när man i slutet av projektet borde använda 10-20 procent av projekttiden till att testa och granska projektet (Görling 2009 s. 153).

Dels testades applikationen av uppdragsgivaren vilket var i många fall givande. Utvecklaren blir snabbt blind för sina egna metoder och något som verkar självklart för mig var kanske inte självklart för någon annan.

4.1.1 Valideringstjänsten W3C

W3C har för både HTML- och CSS-filer en egen valideringsfunktion som jag använt redan tidigare. HTML-valideraren går igenom filen för eventuella teckenfel osv. Man kan säga att valideringsprogrammet är smartare på HTML än webbutvecklaren (Dorward 2012). Iden är att man insätter en länk till den sida man vill validera i ett fält och därefter låta programmet validera webbsidan. W3C har också en valideringstjänst för mobilsidor, tjänsten validerar webbsidorna för enkelhet och att de inte innehåller sådant som mobilapparater inte klarar av.

Det första problemet var att alla mina webbsidor var bakom lösenord och därför kunde jag inte bara klippa ut och klistra in valideringslänken. Jag var tvungen att skapa en testsida som hade färdigt ett sidhuvud och en sidfot eftersom jag hade använt mallar hela tiden.

Under valideringsprocessen framkom några fel där jag hade en sluttagg eller starttagg som inte användes. Det kunde tidvis vara svårt att hålla reda på rätt tagg. Trots att man tog bort på rätt sätt, kunde sidan byta utseende. CSS-valideringen gick bra förutom att CSS3 inte ingår ännu i valideringen. Samma sak gällde många nya HTML5-tillämpningar som valideringen inte har beaktat. Vissa fel var relativt unika p.g.a. speciella tillämpningar på webbsidan vilka jag inte kunde redigera på annat sätt. Jag var t.ex. tvungen att ha variabelnamn skapade med PHP på HTML-element för att skilja åt dem för CSS.

Jag använde också valideraren för mobilanvändning men då kunde jag inte mera göra några stora förändringar i hela webbapplikationen. Egentligen är det mycket svårt att få en webbsida som inte är anpassad endast för mobiler att fungera effektivt på en mobil. En stor svårighet var

att valideringen klagade på bildstorlekar och på användningen av JavaScript. JavaScript fungerar hur bra som helst på min Android-telefon. Bildstorlekarna försökte jag minska men det lönade sig inte att minska dem så mycket att de blir obegripliga på en datorskärm.

5 SLUTSATSER OCH DISKUSSION

Att börja första gången i sitt liv med ett IT-projekt utanför skolan var en utmaning. Samtidigt som det var jobbigt att börja från noll kan man också se det som en möjlighet att få fria händer att bestämma hur och med vad en applikation utvecklas. Dels var det bra att jobba ensam men nog ställde det krav på mig som utvecklare. Trots det lyckades jag helt bra, jag har skapat en webbapplikation som kanske kommer någon dag i användning.

De krav som ställdes på webbapplikationen uppfylldes rätt bra, några saker kunde ha lagts till med lite mera tid. Den största saken som blev ogjord var en responsiv design. Skulle det ha funnits mer tid skulle jag ha troligtvis använt Bootstrap för att skapa en mobilvänlig sida. Med ramverket skulle det ha varit enkelt att skapa skilda vyer enligt en skärmstorlek som granskas i kontrollen. Multispråkstödet fungerar trots att det inte var nödvändigt, men en liten detalj med språket i webbadressen blev orättad.

Valet att använda ramverk var ett bra beslut. Utan ramverk skulle webbapplikationen inte ha blivit så avancerad på en så kort tid. Trots att CodeIgniter redan är ett gammalt ramverk och det finns modernare ramverk har CodeIgniter en stabil användargrupp och frivilliga som utvecklar moduler till det. Nu har en ny version 3.0 redan kommit ut med många nya förbättringar. Utan en stor mängd insticksmoduler skulle ramverket inte ha blivit så komplett på en så kort tid. Vissa saker hindrade ramverket dock. Funktioner i frontenden fungerade på en testsida men i ramverket slutade de att fungera. Sådana saker var jag tvungen att lösa på ett annat sätt.

Själva projektet gick relativt bra enligt planen och tidtabellen höll tillräckligt bra. Redan i ett tidigt skede märkte jag att jag måste minska på tiden som används för dokumentation och testning. För mig själv är det viktigt att dokumentera men jag förstod inte då hur viktigt det är. Speciellt dokumentationen borde ännu förbättras om man tänker att någon annan måste förstå det som jag skapat.

KÄLLOR

The Computer Language Company Inc. 2015 Computer Desktop Encyclopedia. Tillgänglig: <http://lookup.computerlanguage.com/host_app/search?cid=C999999&term=plug-in&lookup.x=36&lookup.y=21> Hämtad 23.4.2015

Connolly, Thomas & Begg, Carolyn. 1995 *Database Systems: A Practical Approach to Design, Implementation, and Management*. 3 uppl,. UK: Addison Wesley, 1236 s.

Cunningham & Cunningham, Inc. 2014 Model View Controller. Tillgänglig: <http://c2.com/cgi/wiki?ModelViewController> Hämtad 18.3.2015

Dorward, David. 2012 Help and FAQ for the Markup Validator, W3C. Tillgänglig: <http://validator.w3.org/docs/help.html> Hämtad 14.4.2015

Edmunds, Ben. 2010 Ion Auth Documentation. Tillgänglig: <http://benedmunds.com/ion_auth/> Hämtad 02.12.2014

Edmunds, Ben. 2015 Ion Auth 2. Tillgänglig: <https://github.com/benedmunds/CodeIgniter-Ion-Auth> Hämtad 24.3.2015

EllisLab Inc. 2013 EllisLab Seeking New Owner for CodeIgniter. Tillgänglig: <https://ellislab.com/blog/entry/ellislab-seeking-new-owner-for-codeigniter> Hämtad 10.3.2015

EllisLab Inc. 2014a Session Class. Tillgänglig: <http://www.codeigniter.com/userguide2/libraries/sessions.html> Hämtad 11.4.2015

EllisLab Inc. 2014b Your Favorite PHP Framework, CodeIgniter Has a New Home. Tillgänglig: <https://ellislab.com/blog/entry/your-favorite-php-framework-codeigniter-has-a-new-home> Hämtad 10.3.2015

EllisLab Inc. 2015 A Brief History of CodeIgniter. Tillgänglig: <https://ellislab.com/codeigniter> Hämtad 10.3.2015

Feiler, Jesse. 2012 *Sams Teach Yourself Core Data for Mac and iOs in 24 hours*, 2 uppl., USA:Sams publishing, 290 s.

GitHub Inc. 2014 Codeigniter 2.1 interntaionalization i18n. Tillgänlig: <https://github.com/bcit-ci/CodeIgniter/wiki/CodeIgniter-2.1-internationalization-i18n> Hämtad 2.12.2014

Görling, Stefan. 2009 *Att arbeta med IT-projekt*, 1 uppl., Lund: Studentlitteratur AB, 308 s.

Harkins, Susan & Reid, Martin. 2001 Many Web developers prefer MySQL, CBS Interactive. Tillgänglig: <http://www.techrepublic.com/article/many-web-developers-prefer-mysql/> Hämtad 26.2.2015

Hunter, Shylon. 2002 MySQL vs. PostgreSQL, CBS Interactive. Tillgänglig: <http://www.techrepublic.com/article/mysql-vs-postgresql/> Hämtad 25.2.2015

Jaglale, Jérôme. 2009 Internationalization (i18n) library for CodeIgniter 2. Tillgänglig: <http://jerome-jaglale.com/doc/php/codeigniter_i18n> Hämtad 25.3.2015

Janssen, Cory. 2015 Application Framework, Janalta Interactive Inc. Tillgänglig: <http://www.techopedia.com/definition/6005/application-framework> Hämtad 4.3.2015

The jQuery Foundation. 2015 What is jQuery? Tillgänglig: <https://jquery.com> Hämtad 24.3.2015

Juskewycz, Helmut. 2013 Internationalization How To for the 5 most popular PHP frameworks, Lingohub. Tillgänglig: <http://blog.lingohub.com/2013/07/internationalization-how-to-5-most-popular-php-frameworks/> Hämtad 30.3.2015

Kotek, Brian. 2002 MVC design pattern brings about better organization and code reuse, CBS Interactive 2015. Tillgänglig: <http://www.techrepublic.com/article/mvc-design-pattern-brings-about-better-organization-and-code-reuse/> Hämtad 17.3.2015

Nettihotelli Internet Oy. 2015 Webhotellit. Tillgänglig: <https://www.nettihotelli.fi/?Etusivu/Webhotellit> Hämtad 21.1.2015

Open Source Initiative. 2015 The Open Source Definition. Tillgänglig: <http://opensource.org/osd> Hämtad 21.1.2015

Oracle. 2015 About MySQL. Tillgänlig: <http://www.mysql.com/about/> Hämtad 3.3.2015

phpBB. 2014a The #1 Free, Open Source Bulletin Board Software. Tillgänglig: <https://www.phpbb.com/> Hämtad 31.3.2015

phpBB. 2014b About phpBB. Tillgänglig: <https://www.phpbb.com/about/> Hämtad 2.12.2014

phpBB. 2014c Site Showcase. Tillgänglig: <https://www.phpbb.com/showcase/> Hämtad 31.3.2015

PHPFrameworks.com. 2013 Top 10 Hot frameworks 2013. Tillgänglig: <http://www.phpframeworks.com/top-10-php-frameworks/> Hämtad 25.10.2014

The PostgreSQL Global Development Group. 2015a About. Tillgänglig: <http://www.postgresql.org/about/> Hämtad 3.3.2015

The PostgreSQL Global Development Group. 2015b History. Tillgänglig: <http://www.postgresql.org/about/history/> Hämtad 3.3.2015

Q-Success. 2015a Usage of client-side programming languages for websites. Tillgänglig: <http://w3techs.com/technologies/overview/client_side_language/all> Hämtad 3.11.2014

Q-Success. 2015b Usage of JavaScript libraries for websites. Tillgänglig: <http://w3techs.com/technologies/overview/javascript_library/all> Hämtad 24.10.2014

Reenskaug, Trygve. 2003 The Model-View-Controller (MVC), Its Past and present. Tillgänglig: <http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf> Hämtad 11.3.2015

van der Schee, Maurits. 2013 10 very good reasons to stop using JavaScript, LeaseWeb. Tillgänglig: <http://www.leaseweblabs.com/2013/07/10-very-good-reasons-to-stop-using-javascript/> Hämtad 23.3.2015

Solid IT. 2015 DB-Engines Ranking. Tillgänglig: <http://db-engines.com/en/ranking> Hämtad 21.1.2015

SpryMedia Ltd. 2015a DataTables. Tillgänglig: <http://datatables.net/> Hämtad 2.12.2014

SpryMedia Ltd. 2015b Installation. Tillgänglig: <http://datatables.net/manual/installation> Hämtad 30.3.2015

W3C. 2012 A Short History of JavaScript, World Wide Web Consortium. Tillgänglig: <https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript> Hämtad 23.3.2015