CHALMERS

Utveckling av ett webbaserat administrativt system

En webbapplikation implementerad med ramverket Symfony

Kandidatarbete inom Informationsteknik

DANIEL GUNNARSSON
HEDVIG JONSSON
SEBASTIAN LJUNGGREN
BOEL NELSON
JENNIFER PANDITHA
ISAK STRÅVIK

Chalmers tekniska högskola
Göteborgs universitet
Institutionen för Data- och Informationsteknik
Göteborg, Sverige, Juni 2013

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet. The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Utveckling av ett webbaserat administrativt system
En webbapplikation implementerad med ramverket Symfony

DANIEL GUNNARSSON,
HEDVIG JONSSON,
SEBASTIAN LJUNGGREN,
BOEL NELSON,
JENNIFER PANDITHA,
ISAK STRÅVIK

© DANIEL GUNNARSSON, June 2013.
© HEDVIG JONSSON, June 2013.
© SEBASTIAN LJUNGGREN, June 2013.
© BOEL NELSON, June 2013.
© JENNIFER PANDITHA, June 2013.
© ISAK STRÅVIK, June 2013.

Examiner: SVEN-ARNE ANDRÉASSON

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

This report describes the implementation of a web-based administrative system for the non-profit organization Intize. It was constructed using an agile work process along with a close cooperation with Intize. Based on requests from Intize, a requirement specification was formulated which was updated as the requests changed. In parallel with the formulation of the requirement specification, functionality was implemented in order of assigned priority. User testing was conducted continuously throughout the project to test new or modified functionality. The result was an administrative web application for Intize, implemented in PHP using the framework Symfony. A total of 69.9 percent of requirements were met. The agile work process proved to alleviate the implementation of functionality because of its flexibility, but work on functional tests and the database became more difficult.

Sammanfattning

Rapporten redogör för implementeringen av ett webbaserat administrativt system åt den ideella föreningen Intize. Detta konstruerades med hjälp av en agil arbetsprocess och ett nära samarbete med Intize. Baserat på önskemål från Intize togs en kravspecifikation fram, som sedan uppdaterades allteftersom önskemålen ändrades. Parallellt implementerades funktionalitet efter tilldelad prioritet. För att testa ny eller förändrad funktionalitet genomfördes användartester kontinuerligt under hela projektets gång. Resultatet blev en administrativ webbapplikation till Intize implementerad i PHP med ramverket Symfony. Totalt har 69,9 procent av kraven uppfyllts. Den agila arbetsprocessen visade sig i stor utsträckning underlätta implementeringen av funktionalitet på grund av dess flexibilitet, men arbetet med funktionella tester och databasen försvårades.

Ordlista

Agila arbetsmetoder Samlingsnamn för flexibla arbetsmetoder som främst används inom mjukvaruutveckling.

Användbarhet Härstammar från engelskans *usability* och syftar på hur användbar en produkt är.

Användningsfall En lista av olika steg som behöver genomföras i en applikation för att utföra en handling eller uppnå ett mål.

Apache HTTP-server med öppen källkod, är en av de vanligaste webbservrarna som används.

Branch Ett begrepp som används i versionshanteringssystem. En *branch* är en arbetskopia för att kunna arbeta parallellt med olika funktioner. Då en funktion i en *branch* är färdig kan den sammanfogas med en annan *branch*.

Django Webbutvecklingsramverk till programmeringsspråket Python.

Doctrine ORM implementerad i PHP som används i Symfony.

Entitet Föremål i databasen, kan liknas vid ett objekt i en objektorienterad applikation.

Funktionella tester Test med uppgiften att kontrollera att en viss funktion i ett system fungerar. Ofta kan detta innebära att testa ett helt användningsfall.

Git Decentraliserat versionshanteringssystem.

Iteration Tidsperiod under vilken en utvald mängd funktionalitet ska implementeras.

JavaScript Skriptspråk som exekveras i webbläsaren och som ger mer interaktivitet med användaren.

MVC *Model-view-controller*, ett objektorienterat designmönster där ett program delas in i modell, vy och *controller* för att separera hanteringen av affärslogik, presentation och användarinmatning från varandra.

OAuth 2.0 Specifikation för *single sign-on* som utöver autentisering även tillåter åtkomst av material sparat hos tredje parten som autentiseringen sker med.

OpenID Specifikation för *single sign-on* som hanterar autentisering med hjälp av en länk från användaren.

ORM *Object-relational mapping*, ett abstraktionslager mellan databasen och den objektorienterade delen av applikationen.

Ramverk Bibliotek med färdiga funktioner.

Redmine Ett ärendehanteringssystem.

Row striping Designmönster för användargränssnitt där raderna i en tabell skiftar mellan två färger för att göra separationen tydligare mellan raderna.

Ruby on Rails Ramverk till programmeringsspråket Ruby, skrivs ofta som enbart Rails.

Sekventiella arbetsmetoder Arbetsmetoder som har ett flöde med olika faser som sker i en sekventiell ordning.

SSO *Single sign-on*, tillåter autentisering av en användare genom en tredje part.

Symfony Ramverk till programmeringsspråket PHP.

Template engine Mjukvara som producerar webbdokument och således kan generera innehåll dynamiskt.

Vagrant Mjukvara som med hjälp av virtuella maskiner möjliggör användandet av samma utvecklingsmiljö oberoende av operativsystem.

Vattenfallsmetoden En typ av sekventiell arbetsmetod.

Visual framework Designmönster för användargränssnitt där samma layout och färger används på alla sidor vilket gör det enkelt för användaren att hitta på sidan.

Webbapplikation Programvara som är tillgänglig med hjälp av en webbläsare.

Öppen källkod Källkod som släpps med en licens som tillåter modifiering och spridning. I vilken grad detta tillåts beror på vilken licens som används.

Innehållsförteckning

1 | Inledning

IT-system används idag av både stora och små organisationer och bidrar till att effektivisera verksamheten. Att skräddarsy ett system för en organisations specifika behov kan bli en ekonomisk börda för mindre organisationer. Detta kan leda till att de använder IT-system som är mer generella och inte helt uppfyller organisationens behov. En konsekvens kan bli att flera olika tjänster måste användas, vilket kan resultera i att det blir svårt att få en överblick över all information.

1.1 Bakgrund

Den ideella föreningen Intize verkar för att främja intresset för matematik. De har över 100 engagerade personer som hjälper gymnasieelever med matematik. De använder sig idag av en ad hoc-lösning för att utföra sitt administrativa arbete. Då Intize är en ideell förening är ekonomin en styrande faktor när val av tjänster görs. Detta har lett till att föreningen använder sig av flera olika gratistjänster istället för att betala för en komplett lösning.

Arbetet sköts med hjälp av tjänster såsom Google Apps, Google Drive och Dropbox. Google Apps används för att hantera kontaktlistor och för att skapa mejladresser till användare. För att hantera medlemmar och anmälningar till aktiviteter används Google Drive. Anmälningar görs med dokument som är tillgängliga för alla medlemmar i föreningen. Dropbox används som säkerhetskopieringslösning.

Ett stort problem som Intize har i dagsläget, till följd av användningen av flera system, är att data dupliceras. Att använda flera system gör det även tidskrävande för administratörer att uppdatera informationen, då ansvariga i de flesta fall måste ändra i flera olika dokument för att göra en uppdatering. Detta är en ineffektiv lösning då den inte främjar användarvänlighet samtidigt som risken att information dupliceras eller inte stämmer överens med information i andra dokument ökar. Intize nuvarande IT-lösning är därför begränsad av de IT-tjänster som används.

Ytterligare ett problem med föreningens nuvarande IT-lösning är att den är sårbar. Ett exempel på detta är att engagerade i föreningen kan avanmäla sig från aktiviteter och manipulera data utan att upplysa den ansvariga om förändringen. Detta medför att data i systemet inte alltid är tillförlitlig.

Eftersom systemet inte är användarvänligt läggs mycket arbetstid på att administrera mejlutskick samt sammanställa information från olika dokument. Detta kostar i nuläg-

et värdefull arbetstid som föreningen skulle kunna lägga på andra delar av verksamheten.

Följaktligen har Intize ett stort behov av ett specialanpassat system för att administrera föreningens data. Eftersom föreningen även har en hög omsättning av personal måste systemet vara lätt att lämna över. En möjlig lösning på detta problem är att utveckla ett webbaserat administrativt system för Intize räkning.

1.2 Syfte

Syftet med detta arbete är att konstruera ett administrativt webbaserat system utformat för den ideella föreningen Intize. Detta system ska uppfylla de önskemål som Intize har.

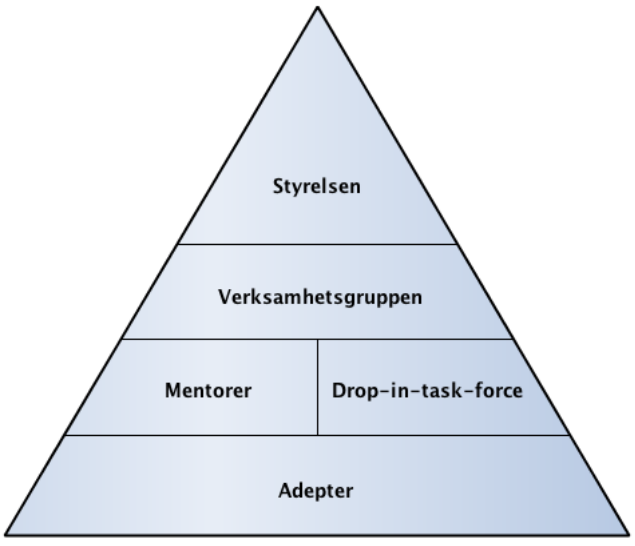
1.3 Metod

Under detta arbete konstruerades ett webbaserat administrativt system i form av en webbapplikation. En kravspecifikation för webbapplikationen arbetades fram med utgångspunkt från Intize önskemål. Användartester utfördes kontinuerligt under utvecklingen. Som underlag för konstruktionen genomfördes en litteraturstudie som berör webbapplikationer, databaser, säkerhet och användartester. Litteraturstudien presenteras i kapitel 3.

2 | Intize

Intize ger högskolestudenter möjligheten att fungera som mentor för en grupp gymnasieelever, kallade adepter, som träffas två timmar i veckan. Mentorerna kan även välja att läsa kursen 'Matematik och Samhälle' där mentorskap ingår. Gymnasieelever har möjlighet att komma till Chalmers på drop-in-tillfällen för att få hjälp med matematik. Vid varje tillfälle finns ett antal mentorer, varav en är huvudansvarig, på plats för att hjälpa eleverna. Föreningen anordnar dessutom evenemang som medlemmar kan anmäla sig till. Evenemang är aktiviteter som inte nödvändigtvis sker regelbundet. Utöver evenemangen anordnas även en fredagslunch en gång i veckan.

Medlemmarna i föreningen är uppdelade i en styrelse, en verksamhetsgrupp och mentorer enligt figur 2.1. Intize styrelse har i uppgift att planera föreningens framtid och besluta i strategiska frågor. Det operativa arbetet sköts däremot av fem anställda som utgör verksamhetsgruppen. Föreningen består till största del av mentorer som har en grupp adepter. Till skillnad från mentorer har drop-in-task-force inte adepter utan närvarar endast på drop-in-tillfällen. Adepter är emellertid inte medlemmar i föreningen utan tar endast del av de aktiviteter som föreningen erbjuder.



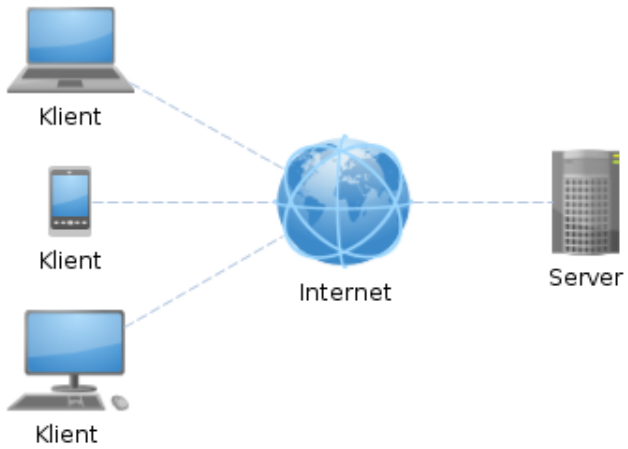
Figur 2.1: Intize interna föreningsstruktur representerad hierarkiskt.

3 | Teknisk bakgrund

Detta kapitel är baserat på den litteraturstudie som utförts för att kunna ta beslut i tekniska frågor. De undersökta teknikerna presenteras nedan.

3.1 Webbapplikationer

En webbapplikation är en distribuerad applikation som en klient får tillgång till via en server. Flera klienter kan, med en webbläsare, ansluta till applikationen över ett nätverk, i regel Internet. Klientens webbläsare skickar meddelanden med en begäran om data eller att få utföra en åtgärd till en server. Servern besvarar sedan dessa genom att returnera önskad data eller utföra åtgärden. Flera klienter kan hanteras samtidigt av servern. Applikationen blir lättillgänglig då det enda som krävs för att använda applikationen är en webbläsare. Figur 3.1 illustrerar hur olika typer av klienter ansluter till en server.



Figur 3.1: En överblick över kommunikationen mellan flera olika typer av klienter och en server i en webbapplikation.

För kommunikation mellan klientens webbläsare och servern över Internet används protokollet *Hypertext Transfer Protocol* (HTTP) [1]. HTTP är ett tillståndslöst protokoll, vilket innebär att all kommunikation till servern är oberoende av tidigare kontakt

med klienten [1]. Protokollet tillhandahåller även mekanismer för att använda webbläsarens cache [1]. Detta görs genom att servern och klienten jämför innehåll, och om innehållet fortfarande är aktuellt behöver inte samma data skickas igen. En server som implementerar HTTP-protokollet kallas för en HTTP-server.

En webbapplikation är oftast uppdelad i tre skikt: presentation, logik och data. Presentationsskiktet ansvarar för hur innehållet visas för användaren. Logikskiktet ser till att rätt åtgärder sker när klienten skickar och begär data. I datalagret sparas persistent data som behövs vid ett senare tillfälle. Vanligtvis är webbläsaren hos klienten ansvarig för presentationen medan servern ansvarar för större delen av logiken och dataskiktet.

De tre viktigaste kriterierna för en webbapplikations framgång är enligt Offutt [2]: driftsäkerhet, användarvänlighet och säkerhet. Driftsäkerhet syftar på att applikationen alltid ska vara tillgänglig. Det andra kriteriet, användarvänlighet, syftar på användarnas förväntningar om att webbapplikationen ska vara lätt att använda. Säkerhet, som är det sista kriteriet, framhäver vikten av ett säkert system för ägarens trovärdighet, speciellt när känslig användardata sparas.

3.1.1 HTML, CSS och JavaScript

De tekniker som används på serversidan i webbapplikationer varierar. På klientsidan används däremot i princip alltid standarderna HTML, CSS och JavaScript för att presentera innehållet.

Hypertext Markup Language (HTML) är ett märkspråk för innehåll på Internet [3]. Med hjälp av HTML struktureras innehållet i ett dokument in i en trädstruktur [3]. Noderna i detta träd kallas för element och representerar exempelvis rubriker, länkar eller listor [3]. En webbläsare kan med hjälp av denna information sedan rendera en hemsida. HTML är en standard som specificeras och underhålls av World Wide Web Consortium (W3C). Standarden specificerar, utöver själva språkets semantik, hur HTML ska renderas och integreras i webbläsaren.

Tillsammans med HTML används *Cascading Style Sheets* (CSS) för att kunna anpassa layouten och utseendet på sidan [4]. Istället för att exempelvis lagra bakgrundsfärger inbäddat i HTML-källkoden möjliggör CSS att denna information lagras i separata filer [4]. Detta gör det enklare att byta till ett nytt utseende på hemsidan [4]. Även CSS är en standard som underhålls av W3C.

Då HTML enbart är ett märkspråk utan stöd för logik behövs en teknik för att dynamiskt uppdatera innehållet på en sida. En sådan teknik är JavaScript som är ett interpreterat programmeringsspråk som vanligtvis exekveras i webbläsaren [5]. Med hjälp av JavaScript kan HTML uppdateras dynamiskt utan att hela sidan behöver laddas om [5].

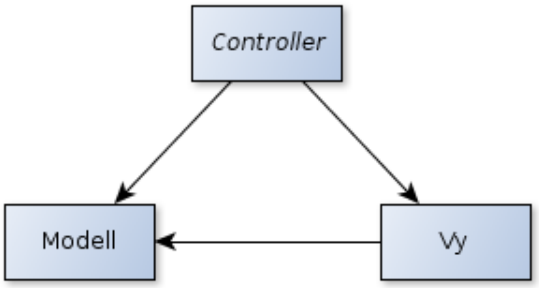
Standarderna för HTML och CSS uppdateras kontinuerligt för att lägga till ny funktionalitet och anpassningar för nya behov. De senaste versionerna är HTML5 och CSS3, vars

specifikationer ännu inte är helt färdigställda. Ett exempel på vad dessa nya tekniker möjliggör är responsiv design. Detta innebär att en hemsida med samma HTML-källkod kan anpassa sitt utseende till olika enheter beroende på upplösning och orientering. Responsiv design implementeras genom att skriva CSS som enbart aktiveras när vissa kriterier uppfylls.

Idag används dessa standarder i allt högre utsträckning i nya webbapplikationer. Däremot har inte alla moderna webbläsare stöd för all ny funktionalitet som följer med HTML5 och CSS3.

3.1.2 MVC i webbapplikationer

Model-view-controller (MVC) är ett designmönster där en applikation delas in i tre olika lager: modell, vy och *controller*. Modellen ansvarar för affärslogik och vyn för presentationen av innehållet. *Controller*-lagret ansvarar för hantering av användarinmatning. Figur 3.2 ger en översikt över hur dessa lager kommunicerar. Gemensamt för olika varianter av MVC är att modellen ej har några beroenden på vy- eller *controller*-lagret.



Figur 3.2: Översikt över hur olika komponenter i MVC har möjlighet att kommunicera med varandra.

I webbapplikationer är en möjlig variant av MVC att låta *controller*-lagret ta emot klientens anrop. Sedan tolkar *controller*-lagret anropet och kallar på modellen för att hämta eller lagra data. Därefter kallar *controller*-lagret på vylagret som genererar HTML med hjälp av data från modellen. Slutligen besvarar *controller*-lagret klientens anrop med vyns HTML. Denna typ av MVC är vanlig i webbapplikationer och möjliggör en tydlig separation av ansvar.

En användares interaktioner med en webbapplikation består ofta av en kombination av att skapa, visa, redigera och ta bort data. Därför kan *controller*-lagret delas upp i flera mindre enheter med ansvar för en viss typ av data. Exempelvis kan en *controller* ges ansvaret för att skapa, visa, redigera och ta bort nyheter. Denna *controller* anropar sedan modellen för att hämta nyheter som därefter genereras till HTML av en nyhetsvy. Resultatet blir att applikationen dels delas upp i MVC-komponenter och dels efter en

specifik typ av funktionalitet eller data. Detta resulterar i en applikationsstruktur med tydlig ansvarsfördelning.

3.1.3 Webbapplikationsramverk

Inom programmering är ett ramverk en samling av generiska komponenter som kan anpassas till ett specifikt syfte. Detta gör det möjligt att återanvända dess generiska källkod i flera olika applikationer. Genom att använda ett ramverk kan en utvecklare snabba upp utvecklingsprocessen och dra nytta av vältestad och prestandaoptimerad källkod.

För att göra det enklare att följa konventioner och att strukturera en webbapplikation används ofta ett webbapplikationsramverk. Ett sådant ramverk tillåter utvecklaren att fokusera på den problematik som är specifik för domänmodellen. Utvecklaren undviker att skriva generisk källkod för att exempelvis koppla anrop till en specifik *controller* eller hantera undantag. Den typ av MVC som beskrevs i det tidigare avsnittet är ett exempel på en struktur som blir enklare att uppnå med hjälp av ett webbapplikationsramverk.

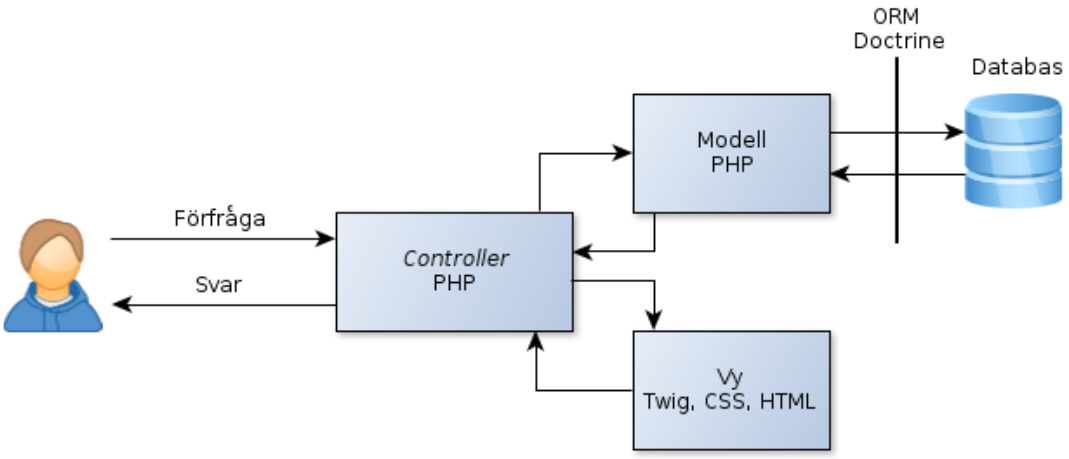
3.1.4 PHP och Symfony

PHP är ett dynamiskt typat programmeringsspråk med stöd för flera programmeringsparadigmer, däribland den objektorienterade. PHPs objektorientering inkluderar arv [6], polymorfism [7] och stöd för namnrymder [8]. PHP har flera användningsområden men används framförallt i logikskiktet i webbapplikationer. Det är exempelvis möjligt att bädda in PHP-källkod i HTML för att skapa en dynamisk hemsida.

I en webbapplikation som är byggd med PHP används en HTTP-server för att ta emot anrop. Därefter skapar servern en process för PHP. Sedan ansvarar PHP för att dynamiskt generera innehåll i HTML eller ett annat format. Det är inte nödvändigt att skapa en PHP-process för innehåll som inte är dynamiskt, exempelvis bilder och CSS-filer.

Symfony är ett webbapplikationsramverk för PHP med öppen källkod. Symfony är uppbyggt av löst sammansatta och fristående komponenter [9]. Detta gör det möjligt att byta ut och anpassa olika delar av ramverket. I många fall har Symfonys utvecklare valt att använda andra ramverk, såsom Doctrine för databashantering och Twig för att skapa vyer.

Grundstrukturen i Symfony är att följa HTTP-protokollets uppbyggnad där systemet tar in en förfrågan och skickar tillbaka ett svar [10], se figur 3.3. Symfony ger en *controller* ansvar för att ta emot en förfrågan och sedan översätta den till ett svar. Denna *controller* hämtar data från modellen för att sedan skicka den till vyn för rendering.



Figur 3.3: Exempel på en applikationstruktur med Symfony.

En Symfony-applikation består av flera *bundles*. En *bundle* är en slags modul som ansvarar för en viss typ av funktionalitet i applikationen [11]. I en Symfony-applikation ingår all källkod i *bundles*, inklusive själva ramverket [11]. Detta gäller för Symfonys komponenter, externa ramverk som Symfony använder sig av och för källkod som är specifik för en applikation. Till exempel kan en *bundle* bestå av funktionalitet för att hantera användare eller bygga applikationens menystruktur. En *bundle* med applikationsspecifik källkod har ofta en egen MVC-modell. Separationen av en applikation i *bundles* medför att en *bundle* endast är löst kopplad till resten av applikationen. Detta gör att man enkelt kan byta ut eller lägga till *bundles* med ny funktionalitet.

Twig är en *template engine* som är väl integrerad med Symfony [12]. Twig är anpassat för att skriva mallar och är ett alternativ till att använda PHP som *template engine* [12]. Twig tillhandahåller även funktionalitet för att kunna utöka och inkludera redan existerande vyfiler för att minska dupliceringen av källkod [13].

För att underlätta hanteringen av databaser är Doctrine, en *object-relational mapper*, väl integrerad med Symfony [14], mer om *object-relational mapping* i avsnitt 3.2.3. I Doctrine används en objektorienterad SQL-dialekt kallad *Doctrine Query Language* (DQL) för att skicka frågor till databasen. Denna dialekt är mycket lik *Hibernate Query Language* som används i det populära Java-biblioteket Hibernate [15].

3.2 Databaser

Databasers underliggande logik skiljer sig åt, vilket gör att olika typer av data hanteras olika bra. Nedan presenteras två databastyper, samt ett abstraktionslager.

3.2.1 Relationsdatabaser

Relationsdatabaser är en väletablerad typ av databas som är ett av de vanligaste sätten att lagra data på. Denna bygger på relationsmodellen, vilken i sin tur baseras på relationsalgebra. För att gruppera data använder sig relationsdatabasen av mängder bestående av tupler. Dessa tupler innehåller attribut som, själva eller tillsammans med andra attribut, bildar en unik nyckel. Eftersom nycklarna är unika kan dessa användas för att para ihop tupler i så kallade relationer. Detta görs genom att den ena tupeln känner till den andras nyckel [16]. Om man vill skapa en relation mellan flera tupler måste dessa sparas i en separat mängd [16].

Relationsdatabaser är anpassade för att hantera komplexa frågor väl, vilket är en av dess största fördelar [17]. En av dess största nackdelar är att den använder *join*-operationer, vilket är en kostsam operation att utföra då databasen först måste slå ihop och sedan sortera två mängder [18].

Det kan uppstå problem för utvecklaren om ett objektorienterat språk används i samband med en relationsdatabas. Problemen bottnar i att relationsdatabaser hanterar data som mängder medan objektorienterade språk hanterar data som objekt. Det problem som uppstår brukar samlas inom begreppet *object-relational impedance mismatch* eller *impedance mismatch*. Några av de problem som uppstår är exempelvis hur arv ska realiseras eller hur inkapsling ska ske [19].

3.2.2 Objektorienterade databaser

Objektorienterade databaser använder sig av en objektorienterad modell. Den största skillnaden i design mellan en relationsdatabas och en objektorienterad databas är hur relationer hanteras. I en objektorienterad databas sparas identifierare i objekt för att hitta relaterade objekt. Detta skiljer sig från en relationsdatabas eftersom relationer mellan flera tupler sparas i en mängd. Den objektorienterade databasen ger stöd för arv och inkapsling [18].

I jämförelse med relationsdatabaser är objektorienterade databasers svaghet hanteringen av komplexa frågor. Dess styrka är att de undviker *join*-operationer då det är möjligt, samt att komplexa objekt hanteras med bättre prestanda än i relationsdatabaser [18].

3.2.3 Object-relational mapping

En *object-relational mapper* eller *object-relational mapping* (ORM) är ett abstraktionslager mellan objektorienterad källkod och en relationsdatabas [17]. Attribut i klasser associeras till tupler i databasen [17]. Med hjälp av dessa associationer kan sedan ORM:en översätta metodanrop på objekt till SQL-frågor. Detta underlättar skapandet av en

domänmodell som är fristående från lagringsmodellen. En ORM ger alltså en alternativ lösning till de problem som uppstår vid *impedance mismatch*.

Då SQL-frågor måste genereras när ORM används påverkas prestandan negativt. Varje gång data ska hämtas från databasen måste metodanropen först översättas till SQL. I jämförelse med att skriva SQL-frågor direkt ökar tiden det tar att interagera med databasen.

Trots detta problem har ORM flera fördelar. Många ORM-implementeringar skyddar mot SQL-injektioner genom att uppmuntra till användandet av *prepared statements* [20], det vill säga att sanera inmatad data innan SQL-frågor utförs. Som tidigare påpekat minskar även ORM problemet med *impedance mismatch* och gör det enklare att skapa en databasoberoende domänmodell.

3.3 Säkerhet

För att konstruera en säker webbapplikation, som per definition är konstant tillgänglig för attacker, måste säkerhetsproblem tas i beaktande. Om applikationen även hanterar flera olika aktörer, som har olika rättigheter, bör anpassad autentisering och auktorisering implementeras.

3.3.1 Vanliga säkerhetsproblem i webbapplikationer

Den ideella organisationen Open Web Application Security Project (OWASP) arbetar med att uppmärksamma och informera om de vanligaste säkerhetsproblemen i webbapplikationer. Nedan beskrivs några av de vanligaste problemen som är relevanta för en webbapplikation, dessa är baserade på en lista[21] publicerad av OWASP.

Ett av de vanligaste säkerhetsproblemen i webbapplikationer är SQL-injektioner. SQL-injektioner uppstår genom att en hackare skriver SQL-frågor i inmatningsfält. Frågorna inkluderas i systemets redan definierade frågor vilket får databasen att exekvera dem. Anledningen till att system är sårbara för SQL-injektioner beror oftast på att de konstruerar sina frågor dynamiskt med hjälp av användarinmatning.

Ett annat vanligt säkerhetsproblem, främst mot användarkonton med administratörsrättigheter, är attacker mot autentiseringen och användares sessioner. Detta är möjligt om systemet har bristande hantering av användaruppgifter. Det kan även vara att autentiserade användare inte blir utloggade korrekt, oavsett om det är en medveten utloggning eller automatisk efter en bestämd tid. En användares session kan även kapas vilket innebär att en hackare kan agera som användaren utan dennes vetskap.

Det finns även en metod som kallas *cross site scripting* (XSS). Den möjliggörs av att användare inte hindras från att skriva in HTML- eller JavaScript-källkod som sedan

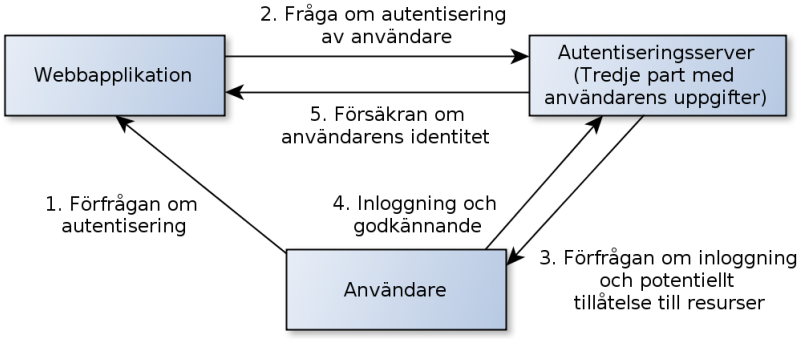
presenteras. Konsekvensen av detta kan till exempel bli att layouten ändras eller att användare omdirigeras till en annan hemsida, vilket kan resultera i att känslig information skickas till hackare.

3.3.2 Single sign-on

Single sign-on (SSO) är en term som beskriver en teknik som tillåter en användare att med endast ett konto, med en uppsättning inloggningsuppgifter, använda sig av flera tjänster. Det bygger på att det finns en tjänst som agerar som tredje part i autentiseringen och som säkerställer att användaren är legitim.

Ett problem som uppkommer när en användare måste komma ihåg ett flertal lösenord är risken att säkerheten blir försämrad [22]. Detta sker eftersom användare, för att komma ihåg lösenord, antingen blir oförsiktiga vid hanteringen av lösenorden eller använder mindre komplicerade lösenord. Även användarvänligheten blir lidande då det sätts ett krav på användare att memorera fler inloggningsuppgifter. SSO är konstruerat för att lösa dessa problem.

En implementering av SSO behöver minst tre komponenter: en webbapplikation, en autentiseringsserver och till sist en användare som initierar autentiseringen. I figur 3.4 visas hur kommunikationen mellan parter i en implementering av SSO kan se ut. Användaren har sina inloggningsuppgifter sparade på autentiseringsservern och vill bli autentiserad mot webbapplikationen. Applikationen i sin tur kommunicerar med autentiseringsservern för att säkerställa användarens identitet. Det finns därmed ingen kommunikation av inloggningsuppgifter mellan webbapplikationen och autentiseringsservern. Detta betyder att kommunikationen endast sker mellan användaren och autentiseringsservern.



Figur 3.4: Ett exempel på kommunikation vid autentisering med hjälp av SSO.

Det finns ett flertal tekniker med fastställda specifikationer som stödjer autentiseringen, men flera av dem är proprietära och därför limiterade till endast ett fåtal tjänster. Teknikerna OpenID och OAuth 2.0 är populära lösningar som utvecklas av organisationer utan ett vinstdrivande syfte och är även licensierade för fritt användande. De två

teknikerna var från början skapade för olika syften, OpenID var för autentisering medan OAuth var för åtkomst av resurser hos en tredje part. Därefter har teknikerna kommit att utvecklats till att erbjuda både autentisering och resursåtkomst.

Specifikationen för OAuth är framtagen av Internet Engineering Task Force (IETF) och i version 2.0 infördes möjligheten att även tillhandahålla autentisering [23]. Organisationen OpenID som arbetar med specifikationer för SSO rekommenderar OpenID 2.0 och OpenID Connect. Skillnaden är att OpenID 2.0 endast tillåter autentisering medan OpenID Connect inkluderar OAuth 2.0 [24].

OAuth 2.0 har en mer komplicerad specifikation än sin företrädare. Det innebär att OAuth 2.0 är svårare att förstå för dem som ska implementera och använda protokollet. Den nya versionen medförde även ett större utrymme för utbyggnad, vilket ger implementeringar en större variation [23]. Detta gör det svårt att hantera flera implementeringar på samma sätt vilket går emot en av anledningarna till att skapa en standard.

3.3.3 Rollhantering

För att avgränsa vad en användare har tillgång till i system, måste det finnas något som identifierar dennas rättigheter. Rollbaserad åtkomstkontroll är en metod för att applicera rättigheter på användaren genom att placera roller på dem. För att få tillgång till olika delar av systemet görs en kontroll av användarens roll som verifierar om denna har tillräckliga rättigheter.

Hanteringen av roller, användares rättigheter och åtkomstkontroller av systemets resurser kan hanteras med obligatorisk åtkomstkontroll. Detta innebär att det finns en auktoritet som kan modifiera användares rättigheter samt vilka rättigheter som behövs för att komma åt resurser.

3.4 Användartester

Enligt Sharp, Rogers och Preece [25] finns det tre olika typer av utvärderingsmetoder för användbarhet. Två av dessa typer genomförs med användare, medan den tredje typen utförs av experter inom interaktionsdesign [25]. Tester med användare leds av en eller flera testledare. Dessa utförs antingen i kontrollerade former, såsom i ett laboratorium, eller i användarens naturliga arbetsmiljö [25]. Det finns olika metoder för att samla data under ett testtillfälle och resultatet från dessa kan vara kvantitativa eller kvalitativa [25].

3.4.1 Utvärderingsmetoder

Tester som genomförs i kontrollerade former kan fokusera enbart på användbarheten av systemet [25]. Däremot är det svårt att hitta problem som beror på omgivningen där systemet används [25]. Utav de presenterade utvärderingsmetoderna passar denna bäst för att undersöka om systemet är anpassat för den avsedda målgruppen [25].

För att upptäcka problem som beror på omgivningen kan tester genomföras i användarens naturliga arbetsmiljö [25]. Dessa tester har även fördelen att oväntad information kan erhållas [25]. Nackdelen med sådana tester är att dessa är mycket tidskrävande då de innebär observation av samspelet mellan användaren, slutprodukten och externa faktorer som kan påverka användaren [25].

Den tredje typen av utvärderingsmetoder utförs av experter inom interaktionsdesign [25]. Metoder i denna kategori kan till exempel användas i de fall det inte finns möjlighet att testa på användare. De går även att använda om målet med testet är att analysera systemet utifrån bestämda riktlinjer [25].

Sharp, Rogers och Preece [25] anser att användare som deltar i användartest, och därmed är delaktiga i utvecklingsprocessen, ofta är mer benägna att använda produkten efter slutförandet.

3.4.2 Datainsamling

Det finns olika metoder för att samla data under ett testtillfälle. En av de viktigaste metoderna är tänk-högt-metoden som innebär att användaren hela tiden ska berätta vad han eller hon tänker [26]. Detta tillvägagångssätt ger testledaren en inblick i tankegången hos användaren, vilket gör det lättare att upptäcka eventuella missuppfattningar om funktioner hos grafiska element [26]. Fördelen är att testledaren får reda på varför användaren tycker vissa moment är svårare respektive enklare med systemet [25]. Testledaren kan även genomföra intervjuer med användaren, både före och efter tillfället. Sådana intervjuer kan antingen vara strukturerade, vilket innebär att testledaren strikt följer intervjufrågorna, eller halvstrukturerade, där passande följdfrågor kan ställas av testledaren beroende på användarens svar [25].

Resultatet från en utvärderingsmetod kan antingen vara kvantitativ, kvalitativ eller en kombination av dessa. Fördelen med kvantitativ data är att det är lätt att sammanställa resultatet [25]. När resultatet innehåller kvalitativ data kan det ta längre tid att sammanställa och analysera svaren [25]. Detta beror på att det inte alltid är lätt att tolka svaren, men däremot kan det sammanställda resultatet spegla verkligheten bättre än vad kvantitativ data kan göra [25].

4 | Realisering

Nedan beskrivs processen från framtagandet av kravspecifikationen till den slutgiltiga implementeringen av systemet. Applikationen designades för att uppfylla de kriterier som introducerades i avsnitt 3.1. Övriga val som gjorts under framtagandet av slutprodukten presenteras och motiveras.

4.1 Arbetsmetod

Arbetet valdes att utföras agilt på grund av att det ger flexibilitet i arbetsprocessen. Därmed kunde implementeringen påbörjas tidigare än om en sekventiell process, till exempel vattenfallsmodellen, använts. Detta var fördelaktigt eftersom alla krav inte behövde vara fastställda innan implementeringen påbörjades. Krav kunde dessutom ändras när som helst under utvecklingsprocessen på grund av den agila arbetsprocessen.

Projekttiden delades in i totalt sju iterationer som hade olika mål för vad som skulle uppnås inom den bestämda tidsperioden. Iterationerna hade varierande längd, de kortaste varade en vecka och de längsta två veckor. Den sista var endast avsedd till åtgärdandet av buggar i systemet.

4.2 Kravspecifikation

Med utgångspunkt i önskemålen från Intize, se bilaga B, togs ett första utkast till en kravspecifikation fram. Då många av Intize önskemål var otydliga, och troligtvis skulle ändras under utvecklingsprocessen, var det passande med en agil arbetsprocess. Utöver Intize önskemål låg även deras gamla system till grund för att kartlägga vilken funktionalitet som behövdes. Under arbetets gång har kravspecifikationen kontinuerligt uppdaterats efter att föreningens önskemål ändrats eller blivit tydligare.

Önskemålen som Intize tillhandahöll var vaga då en stor del av funktionaliteten var ospecificerad. Arbetet med att kartlägga föreningens behov kunde, tack vare den agila arbetsprocessen, fortsätta parallellt med implementeringen. Detta medförde att mer tid fanns till att konstruera en stabil grund för systemet innan avancerad funktionalitet började implementeras. Hade arbetet istället skett sekventiellt hade det inte varit möjligt att göra stora ändringar i kravspecifikationen under arbetets gång. Det hade med andra ord behövts göra en grundlig undersökning av föreningens behov redan innan im-

plementeringen påbörjades, vilket hade lett till att den försenats och därmed lämnat mindre tid för utvecklingen.

För att få en tydligare bild av hur mentorerna, och inte bara verksamhetsgruppen, använde systemet utfördes en intervju med de aktiva mentorerna. Intervjun innehöll en mängd frågor om hur de uppfattade det gamla systemet och hur de exempelvis anmälde sig till drop-in-tillfällen. För fullständiga intervjufrågor, se bilaga A. Efter att intervjun utförts sammanställdes svaren, vilket resulterade i en klarare bild av hur systemet användes och vilka funktioner som mentorerna önskade. Med detta som underlag kunde kravspecifikationen uppdateras för att även ta hänsyn till mentorernas åsikter och behov.

4.3 Utvecklingsprocess

När implementeringen av systemet påbörjades diskuterades det vilken prioritet de olika kraven skulle ha, för att säkerställa att den mest vitala funktionaliteten implementerades först. Kraven implementerades sedan i prioritetsordning i olika iterationer, där det även var möjligt att ändra prioritet på funktionalitet om det visade sig att något hade blivit felprioriterat. Att arbeta i iterationer ger även fördelen att användartester är lätta att integrera med denna arbetsprocess då funktionalitet implementeras inkrementellt [27].

All implementering av ny funktionalitet upphörde fem veckor innan produkten skulle levereras. De sista veckorna användes enbart till att förbättra existerande funktionalitet, samt rätta till buggar.

4.3.1 Processtöd

För att möjliggöra utveckling från flera olika datorer parallellt användes Git, som är ett decentraliserat versionshanteringssystem. När ny funktionalitet skulle implementeras skedde denna utveckling isolerat i egna *branches* för att undvika att störa andra delar av utvecklingen. När utvecklingen i en *branch* var klar och körbar, sammanfogades den nya källkoden med *development-branch*:en. Denna arbetsprocess medförde även att den källkod som låg på *development-branch*:en alltid var en körbar version av applikationen.

Ärendehanteringssystemet Redmine användes under implementeringen för att underlätta arbetet med att införa ny funktionalitet. Med Redmine gick det lätt att tilldela uppgifter och arbetet kunde effektiviseras då inte mer än en utvecklare jobbade på samma funktionalitet. Detta gjorde även att antalet konflikter i källkoden kunde minimeras då alla var medvetna om vad resterande utvecklare arbetade med.

Redmine gjorde det även enklare att hantera buggar och ny funktionalitet som uppkommit vid användartesterna. Nya önskemål utreddes först för att se om de skulle im-

plementeras eller ej, och därefter lades de in i ärendehanteringssystemet medan buggar däremot alltid lades in direkt. Däremot lades buggar inte till som mål för iterationer, vilket medförde problem i hanteringen då det var svårt att avgöra vad som skulle implementeras först. Detta berodde främst på att det inte fanns någon inbördes prioritering mellan buggar och funktionalitet.

4.3.2 Utvecklingsmiljö

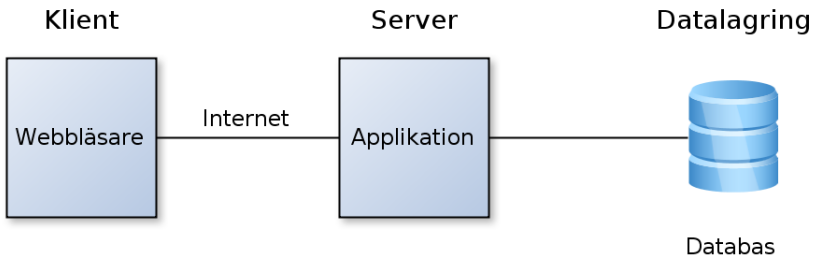
Eftersom gruppmedlemmarna använde olika operativsystem blev det svårt att sätta upp likvärdiga utvecklingsmiljöer. Ett verktyg som kunde ha underlättat detta var Vagrant, och det beslutades därför att Vagrant skulle undersökas. Vagrant gör det möjligt att med hjälp av virtuella maskiner sätta upp samma utvecklingsmiljö oavsett vilket operativsystem som används. Efter att utvecklingsmiljön skapats går det sedan att återskapa identiska miljöer på andra datorer.

Vagrant visade sig inte vara ett lämpligt verktyg på grund av att det tog lång tid att ladda webbapplikationen. Detta berodde på prestandan i det filsystem som delades mellan värddatorn och den virtuella maskinen. När antalet filer som behövde delas blev stort, vilket var fallet i detta projekt, försämrades hastigheten snabbt på läs- och skrivoperationer. Resultatet av denna undersökning var att tid som kunde ha lagts på att arbeta med projektet istället lades på att konfigurera och försöka använda Vagrant.

Istället för att använda Vagrant fick gruppmedlemmarna installera sina utvecklingsmiljöer på egen hand. Det ledde till att tid behövde läggas på att ta reda på hur utvecklingsmiljön skulle sättas upp på olika plattformar. Ett ännu större problem var att utvecklingsmiljöns prestanda varierade stort mellan de olika operativsystem som användes.

4.4 Systemarkitektur

Systemet är implementerat som en klient-server applikation med en databas, se figur 4.1. Klienten använder en webbläsare för att komma åt applikationen på servern. Applikationen använder sig av databasen för att hantera persistent data.



Figur 4.1: Översikt av systemets arkitektur. Flera klienter kan ansluta till servern samtidigt.

För att presentera innehåll för klienten används språken HTML, CSS och JavaScript som körs i webbläsaren.

På serversidan behövdes ett val av programmeringsspråk göras. Valet stod mellan PHP, Python och Ruby med respektive ramverk Symfony, Django och Ruby on Rails. Efter mycket diskussion föll valet på PHP med Symfony. Anledningen till detta beslut var att Intize webbhotell, där deras nuvarande hemsida är belägen, endast hade stöd för PHP. Det fanns däremot tre olika alternativ för att inte vara låst till PHP:

- Två webbhotell
- Byta webbhotell
- Egen server

Det första alternativet var att Intize skulle använda sig av två olika webbhotell. Detta skulle däremot inneburit att en oönskad månadsutgift tillkom.

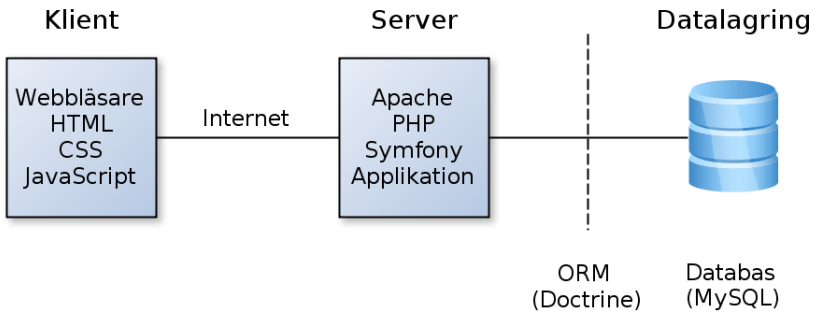
Det andra alternativet hade inneburit att byta webbhotell. Detta hade behövt ha stöd för minst två programmeringsspråk, annars hade Intize nuvarande hemsida behövt konverteras till det nya programmeringsspråket. Denna konvertering uteslöts då de skulle ta för mycket tid från projektets huvudmål.

Det sista alternativet hade varit att Intize införskaffade en egen server. Denna lösning var inte tänkbar för Intize, då det skulle kräva mer förkunskaper hos den som är IT-ansvarig i föreningen.

Då de alternativ som presenterades ovan visade sig ha sina nackdelar bestämdes det att Intize nuvarande webbhotell skulle behållas. Det webbhotellet använder Apache som webbserver och MySQL som databashanterare. Eftersom applikationen är belägen på samma webbhotell, körs även den på Apache. MySQL valdes eftersom den databashanteraren har öppen källkod.

Som tidigare beskrivits har Intize begränsade ekonomiska medel och därmed har valet av tekniker och externa bibliotek fallit på alternativ med öppen källkod. Detta har inneburit att Intize inte behövt betala för licenser och support.

Figur 4.2 visar en slutgiltig bild av systemets arkitektur. Klienten kommer åt applikationen på servern med hjälp av en webbläsare där HTML, CSS och JavaScript används. På servern används Apache som webbserver, och applikationen är implementerad med PHP och Symfony. Applikationen får tillgång till databasen, som använder MySQL, genom Doctrine som ORM-lager.



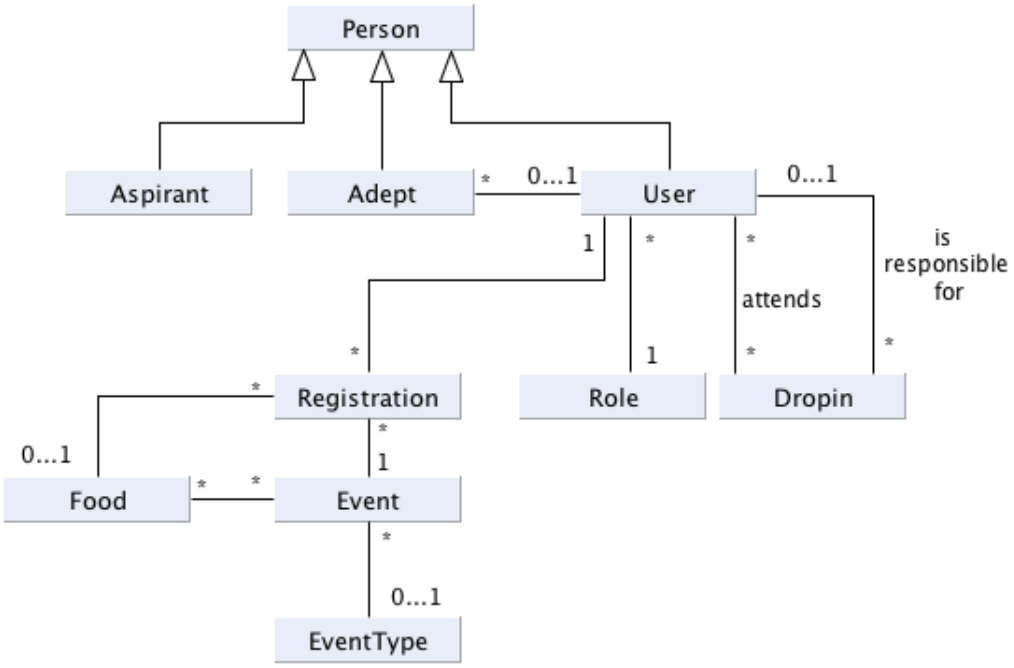
Figur 4.2: Översikt av de tekniker som används i systemet.

4.5 Systemdesign

Systemets kärna är en objektorienterad domänmodell som representerar delar av Intize organisation. Denna är enligt Symfonys konventioner indelad i *bundles*. Doctrine används för att förenkla lagringen av domänmodellen i en relationsdatabas.

4.5.1 Domänmodell

Systemets domänmodell abstraherades från Intize nuvarande föreningsstruktur för att passa en objektorienterad modell. På grund av detta har även ytterligare klasser behövts skapas. Figur 4.3 visar de centrala klasserna i systemets domänmodell.



Figur 4.3: En överblick av systemets domänmodell. Klassen *User* är central och representerar systemets användare.

Klassen *Person* implementerades som superklass till klasserna *Adept*, *Aspirant* och *User*. Den innehåller gemensamma instansvariabler som exempelvis namn, mejladress och telefonnummer. *Adept* innehåller information om adepter och en länkning till en *User*. Klassen *Aspirant* representerar aspirerande mentorer, dessa uppgraderas sedan till *User* om aspiranten blir medlem i föreningen. *Adept* och *Aspirant* representerar inte användare i systemet och tillåts därför inte att logga in.

User representerar en användare i systemet och en medlem i Intize. En association till *Role* avgör vilken roll medlemmen har i Intize. Vilken funktionalitet som medlemmen ges tillgång till i systemet avgörs av dess roll. Notera att en *User* endast kan ha en roll, då dessa är hierarkiska.

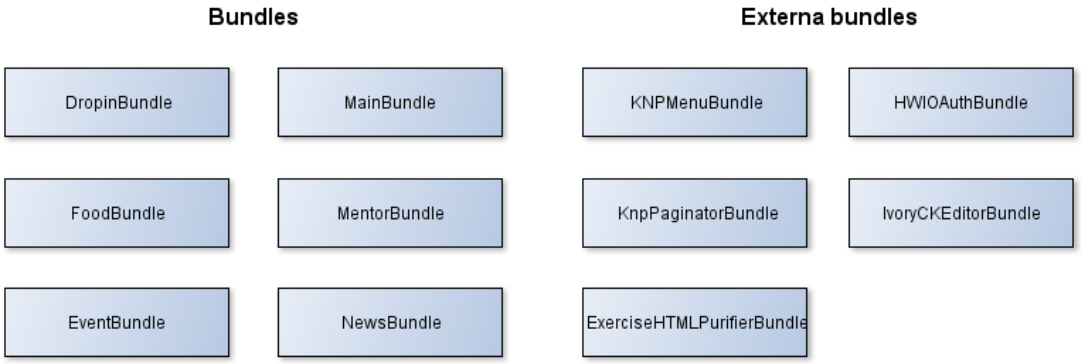
Intize drop-in-tillfällen representeras av klassen *Dropin*. *User* kan anmäla sig till dessa antingen som ansvarig eller närvarande. Endast en *User* är ansvarig för ett drop-in-tillfälle, men flera instanser av *User* kan närvara för att hjälpa gymnasiestudenter.

Intize väljer att särskilja på evenemang och fredagsluncher, däremot gör systemet inte det genom att representera båda som klassen *Event*. Denna särskiljning görs med hjälp av klassen *EventType*, vilket medför att systemet är utbyggbart. På grund av att fredagsluncher och evenemang har många aspekter gemensamt har de valts att representeras som samma klass, trots att Intize själva inte lagrat det så tidigare.

Klassen *Registration* används för att kunna spara en användares matpreferenser vid anmälan till ett *Event*. Denna klass skapades för att användare ska kunna ha olika matpreferenser för olika tillfällen, istället för att alltid ha samma. Matpreferenserna representeras av klassen *Food*. Notera att de ej inkluderar allergier för att användaren inte ska behöva ange dem vid varje anmälan. En användares allergier sparas istället i klassen *User*.

4.5.2 Applikationsstruktur

Webbapplikationen följer Symfonys konventioner med MVC och *bundles*. Exempel på *bundles* är *DropinBundle* som hanterar skapande, redigering och anmälan till drop-in-tillfällen och *MentorBundle* som ansvarar för hanteringen av mentorer och adepter, se figur 4.4. Dessa *bundles* har sina egna MVC-implementeringar med *controllers* som tar emot förfrågningar och talar med modellobjekt i samma *bundles*, och i vissa fall andra *bundles*, för att slutligen rendera dem med en vy.



Figur 4.4: *Bundles* som används i applikationen.

Utöver dessa *bundles* som tagits fram till Intize har även externa *bundles* använts. Exempelvis användes *KnpMenuBundle* för att bygga applikationens menyer och *HWIOAuthBundle* för inloggning med Google. Detta har möjliggjort ett större fokus på den egna domänen istället för att ägna tid åt att lösa vanligt förekommande problem inom webbutveckling.

4.5.3 Datalagring

Ett av de största problemen med Intize nuvarande lösning är avsaknaden av en centraliserad datalagring. Därför lades stor vikt på att modellera en lösning som innebar att data endast lagras på en plats i systemet.

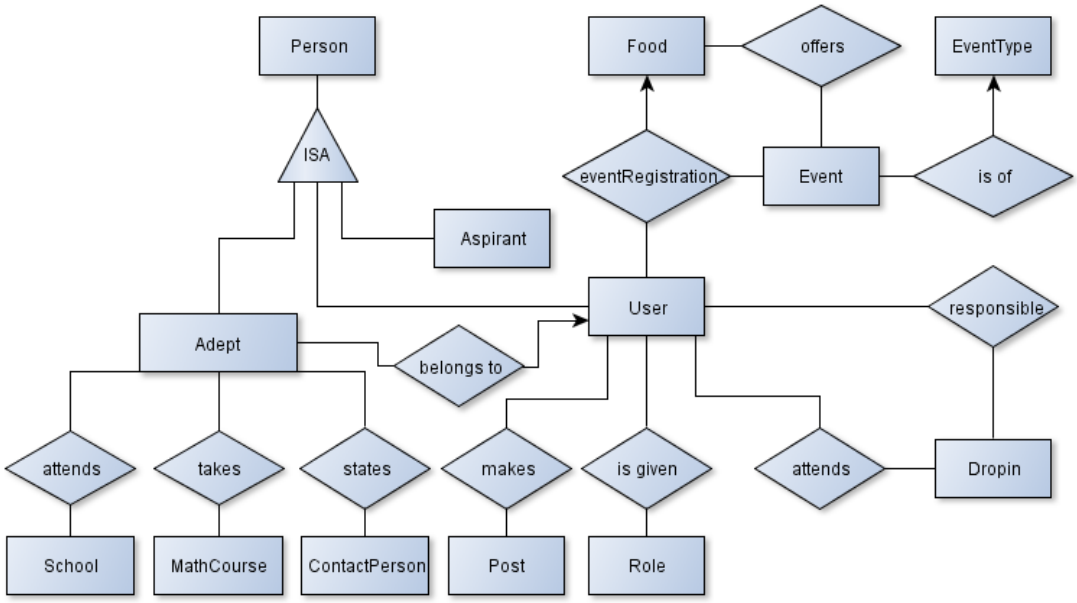
Utöver praktiska detaljer måste systemet kunna utföra filtrering av data. Därför var det viktigt att databasen effektivt kunde hantera komplexa frågor, då dessa används vid filtrering. Valet av databas baserades därför på två faktorer:

- Komplexa frågor måste hanteras väl.
- Intize webbhotell måste ha stöd för databasen.

Som tidigare nämnts i avsnitt 3.2.1 hanterar en relationsdatabas komplexa frågor väl, vilket talade för att en relationsdatabas var att föredra. Det visade sig även att Intize webbhotell endast erbjöd stöd för relationsdatabaser vilket blev en avgörande faktor.

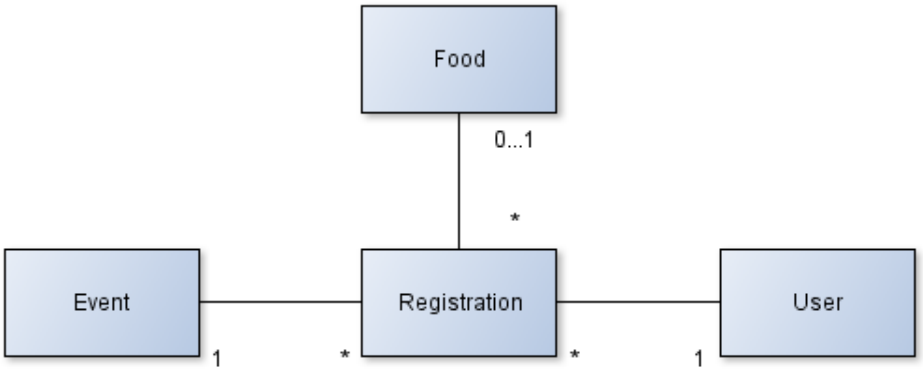
MySQL används som relationsdatabashanterare och som ORM-lager används Doctrine. Valet att använda en ORM berodde på att Doctrines fördelar ansågs väga upp för de brister med ORM som omnämns i avsnitt 3.2.3. Fördelarna är bland annat skydd mot SQL-injektioner samt att det minskar problemen som kan uppstå med *impedance mismatch*. Dessutom medför användandet av en ORM att det blir enklare för Intize att byta databashanterare i framtiden, då källkoden inte är implementeringsspecifik.

Den logiska strukturen modellerades med hjälp av ett *entity-relationship* diagram, som låg till grund för den första versionen av databasen. Modellen har sedan uppdaterats när problem eller nya tankar uppkommit för att förbättra den. I figur 4.5 visas *entity-relationship*-diagrammet som ger en översikt över databasens slutgiltiga implementering. Attribut har plockats bort då dessa gjorde det svårt att se grundstrukturen. I bilaga D visas varje entitet med attribut var för sig.

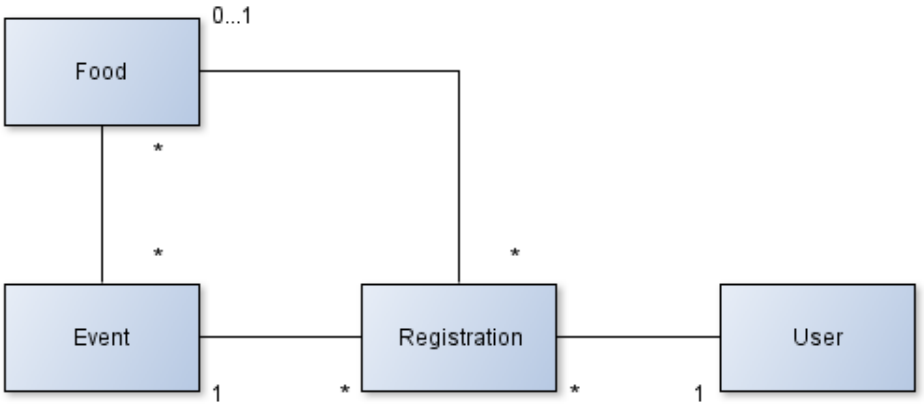


Figur 4.5: Den slutgiltiga databasmodellen. Entiteternas attribut har plockats bort för att ge en tydligare översikt.

Vid modelleringen av databasen uppstod ett problem med hur relationen mellan *Food*, *Event* och *User* skulle se ut. Eftersom ingen uppenbar lösning fanns arbetades två förslag på lösningar fram. Dessa förslag visas i figur 4.6 och figur 4.7. Det konstaterades att båda lösningarna skulle representeras med samma antal rader i databasen. Däremot skulle lösningen i figur 4.6 använda sig av fler *join*-operationer. Därför valdes lösningen i figur 4.7.

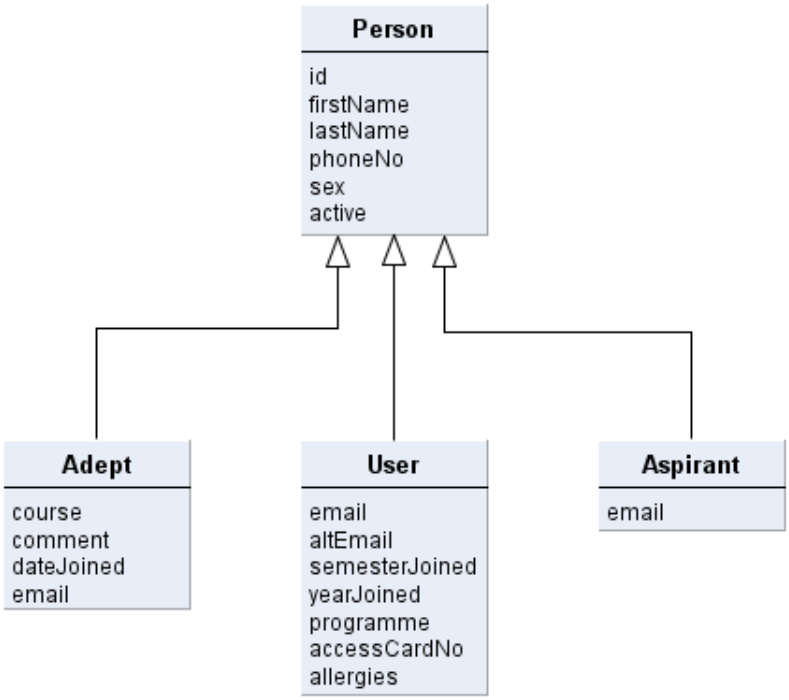


Figur 4.6: En UML-representation av hur användare som registrerat sig på ett evenemang skulle kunna sparas internt i databasen. Notera att endast ett registreringsobjekt skapas per matpreferens.



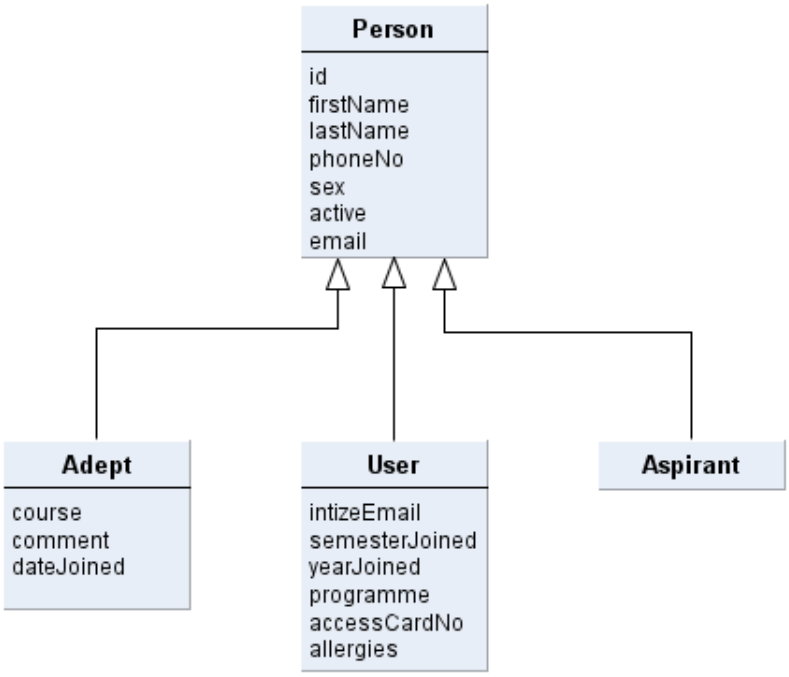
Figur 4.7: En UML-representation av hur användare som registrerat sig på ett evenemang sparas internt i databasen. Notera att ett nytt registreringsobjekt skapas per anmäld användare.

Under implementeringen upptäcktes att entiteterna *Person*, *User*, *Adept* och *Aspirant* hade implementerats på ett redundant sätt. Detta på grund av att ändringar i kravspecifikationen inte speglades i databasen, vilket var en följd av den agila arbetsprocessen som användes. Entiteten *Person* är en superentitet och har som funktion att hålla gemensamma attribut som används av subentiteterna *User*, *Adept* och *Aspirant*. Det upptäcktes att attributet *email* upprepades i alla entiteterna, se figur 4.8. Anledningen till detta var i första hand att attributet *email* i *Adept* var tvunget att vara unikt och inte fick lämnas tomt. Detta stod i strid med att de andra subentiteterna inte hade något krav att tillhandahålla en privat mejladress. Ytterligare en anledning till implementeringen i figur 4.8 var att entiteten *User* hade ett attribut *email* som skulle agera som inloggningsnamn i systemet. Följaktligen kan inte denna ärvas till varje subentitet då *Adept* och *Aspirant* inte skulle ha inloggningsmöjligheter. Det resulterade i att *User* fick två attribut för mejladresser, *email* för inloggning och en privat mejladress *altEmail*.



Figur 4.8: UML-diagrammet första versionen av hur relationer implementerats i systemet.

Det visade sig att kravet på att *Adept*:s *email* skulle vara obligatorisk försvann under implementeringen. Således fanns det ingen anledning till att skilja på *User*:s *altEmail* och *Adept*:s *email*. Därför ändrades strukturen så att *Person* fick attributet *email* som alla subentiteter ärvde, som tillåts vara tom. Inloggningsnamnet i *User* ändrades sedan till att representeras av *intizeEmail*. Genom att ändra strukturen försvann upprepningen av attribut. Lösningen som implementerades i systemet kan ses i figur 4.9.



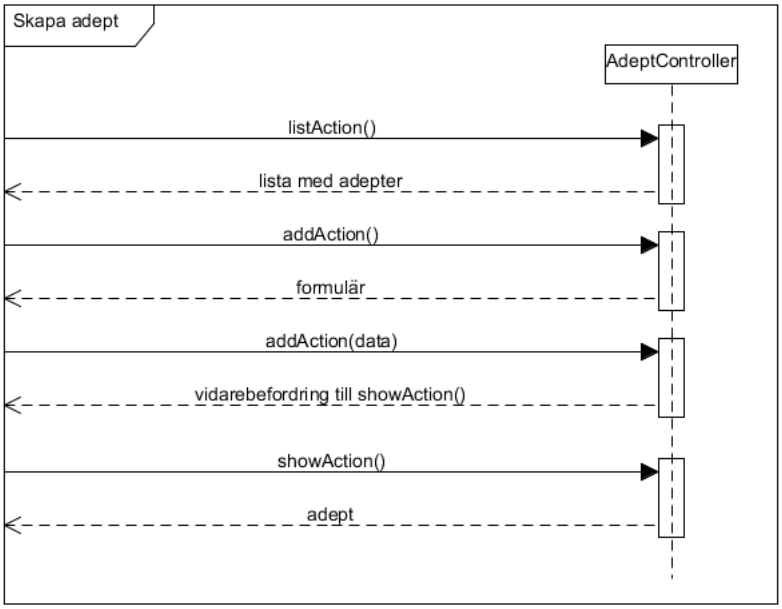
Figur 4.9: UML-diagrammet som beskriver den slutgiltiga relationen mellan superentiteten *Person* och dess subentiteter *Adept*, *User* och *Aspirant*.

4.5.4 Applikationsflöde: ett exempel

En av verksamhetsgruppens uppgifter är att hantera adepter och tilldela dem till mentorer. För att kunna göra det behöver de lägga till nya adepter i systemet.

I systemet utförs detta användningsfall i tre steg med en verksamhetsgruppmedlem som aktör. Det första är att visa sidan för adepthantering och göra valet att skapa en ny adept. Nästa steg är att fylla i och skicka formuläret för att skapa adepter. Hur detta formulär ser ut visas i avsnitt 5.2. Slutligen skickas aktören till en sida som visar den nya adepten samt en bekräftelse på att åtgärden lyckats.

Figur 4.10 visar ett sekvensdiagram över hur klassen *AdeptController* hanterar användningsfallet *Skapa adept*. Först görs ett anrop till metoden *listAction*() som returnerar en lista med adepter. Utöver listan ges aktören möjligheten att skapa en adept. När aktören väljer att skapa en adept anropas *addAction*(). Detta leder till att formuläret för att skapa adepter visas. Efter att ha fyllt i och skickat formuläret anropas *addAction*() igen, denna gång med data från formuläret. Om denna data är giltig skapas en adept och slutligen vidarebefordras aktören till visningssidan. I *AdeptController* innebär detta ett anrop till *showAction*().



Figur 4.10: Sekvensdiagram över de anrop som görs på *AdeptController* för att skapa en adept. Varje metodanrop initieras av ett anrop från klientens webbläsare till servern.

Metoden *showAction*() visas i källkodsexempel 4.1. Med hjälp av ett ID anropas modelllagret, Doctrine, för att hämta en adept från databasen. Varje adept ges i databasen ett unikt ID. Om den efterfrågade adepten inte existerar visas en felsida. Om en adept hittas skickas resultatet till vylagret, Twig, som genererar HTML. Denna HTML skickas till klienten för att slutligen renderas i webbläsaren.

```
public function showAction($id)
{
    $em = $this->getDoctrine()
               ->getManager();
    $adept = $em->getRepository('IntizeMentorBundle:Adept')
                ->findOneBy(array('id' => $id));

    if (!$adept) {
        throw $this->createNotFoundException(
            'Unable to find adept.'
        );
    }

    return array('adept'=>$adept);
}
```

Källkodsexempel 4.1: Implementering av *showAction*() i *VGAdeptController*.

Det exempel som givits i detta avsnitt är typiskt för hur andra användningsfall har implementerats i systemet.

4.6 Säkerhetslösningar

Eftersom säkerhet är en central del i implementeringen av webbapplikationen har flera säkerhetsåtgärder vidtagits, detta för att garantera att endast behöriga användare kan få tillgång till applikationen. Nedan presenteras hur autentisering och auktorisering har implementerats. Några vanliga säkerhetsproblem kommer att adresseras. Även en kortare beskrivning över hur inmatningskontroll hanteras kommer att ges.

4.6.1 Injektioner, kapning och cross site scripting

Symfony erbjuder flera hjälpmedel för att skydda mot de vanligaste säkerhetsproblemen. Ett av dessa tillhandahålls av Doctrine och är en metod för att motverka SQL-injektioner. Följer man Doctrines riktlinjer är det enkelt att undvika risken att injektioner sker. Däremot går det fortfarande att utföra osäkra SQL-frågor genom att konkatenera input från användaren. Rekommendationen är att man ska utföra SQL-frågor med *prepared statements* som visas i källkodsexempel 4.2 [28]. När denna typ av frågor utförs saneras inparametrarna för att se till att de är säkra att exekvera. För att eliminera risken för SQL-injektioner har webbapplikationen designats så att alla SQL-frågor utförs med *prepared statements*.

```
$dql = "SELECT u FROM User u WHERE u.username = :name";
$query = $em->createQuery($dql);
$query->setParameter("name", $username);
$data = $query->getResult();
```

Källkodsexempel 4.2: Exempel på en SQL-fråga med *prepared statements*. Frågan är skriven i Doctrine Query Language (DQL).

Genom att använda Google som autentiseringstjänst finns ingen förvaring av inloggningsuppgifter i webbapplikationen. Ansvaret för säker förvaring har istället placerats på Google. När det gäller kapning av sessioner finns det två potentiella säkerhetsrisker i applikationen. Dessa är kommunikationen med Google och förvaringen av sessionen i applikationen. Kommunikationen med Google sker med SSL, vilket är ett säkert transportprotokoll, som krävs av Googles autentiseringsserver. Sessioner som skapas genom autentiseringen håller endast en timme innan de blir ogiltiga. Detta görs för att minska risken att någon obehörig får tillgång till en behörig användares session. Resterande hantering av sessionen görs av Symfonys inbyggda säkerhetskomponent. Den korrekta hanteringen av sessionen baseras på att denna komponent är säker.

XSS är ett bekymmer som applikationens *template engine* Twig tillhandahåller en lösning på. Med rätt standardinställningar sanerar Twig allt som skrivs ut genom att avlägsna vissa tecken från utmatningen. För att tillåta viss källkod att skickas till webbläsare har det konstruerats ett eget filter genom att lista taggar som tillåts. På detta sättet kan webbapplikationen vara säker mot XSS medan den fortfarande tillåter att innehåll formateras.

4.6.2 Autentisering

Användare i systemet har användarkonton på Google som är kopplat till Intize Google Apps-konto. För alla sådana konton erbjuder Google SSO-tjänster baserade på OAuth 2.0, OpenID 2.0 och hybriden OpenID/OAuth [29]. Därför gjordes ett aktivt val att inte sköta autentisering i systemet utan istället använda en tjänst som erbjuds av Google. Detta innebär att en introduktion av fler lösenord undvikits. SSO medför även fördelen att sparandet av lösenord och en säker autentisering sköts av en trovärdig tredje part.

Valet av protokoll vid autentisering gjordes för att se till att systemet ska kunna utökas och att det finns ett långvarigt stöd. Därför valdes OAuth 2.0, då det stödjer både autentisering samt åtkomst av användares innehåll på Google. Google har, i dokumentation[30], uttryckt att denna lösning möter specifikationen för OpenID Connect vilket försäkrar om ett bra tekniskt stöd.

Extern autentisering med Google medför att övergången till det nya systemet blir lättare för användarna. Detta var i enighet med mentorernas önskan om att inte behöva ha flera

användarkonton. Under samtalet med mentorerna i början av projektet var nämligen flera mentorers åsikt att systemet skulle ha samma inloggning som de har hos Google, se bilaga A.

Inloggning tillåts enbart med mejladresser som har Intize domän, och mejladressen måste även tillhöra en användare i systemet för att inloggningen ska lyckas. Alternativet, att låta användare logga in utan att de finns i systemet visade sig vara besvärligt då konton behövde skapas automatiskt och information ändå kompletteras. Istället intygar inloggningen att både mejladressen existerar och att användaren tillåts använda systemet.

Tack vare stödet för att hämta innehåll från Google skulle en utökning av systemet kunna göras för att skaffa tillgång till information och resurser som är kopplat till användaren hos Google. Detta kan enkelt utföras genom att utöka omfånget av resurser som systemet ber om tillgång till av användaren vid autentisering.

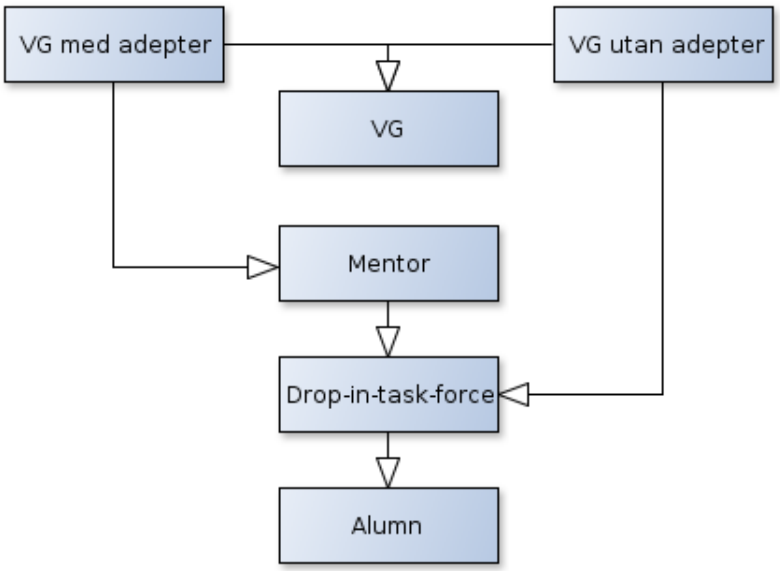
4.6.3 Auktorisering

Vilka rättigheter en användare har i systemet beror på vilken roll den är kopplad till i databasen. Implementeringen av auktorisering baseras på rollbaserad åtkomstkontroll som är realiserad genom att en administratör har det yttersta ansvaret för vilken användare som har vilka privilegier i systemet. Rollhanteringen påbörjades med två roller i åtanke, verksamhetsgruppmedlem, förkortat VG, och mentor. Under arbetsprocessen separerades logik för att försäkra att all åtkomstkontroll kunde appliceras på detaljnivå. Vid senare iterationer började ytterligare roller undersökas.

Eftersom Intize redan har definierat roller och vad de har tillgång till i föreningen var det enklast att översätta dessa till roller i systemet. Däremot är systemets interna roller inte en komplett spegling av föreningsstrukturen. En användare, i systemet, är begränsad till endast en roll. Roller en användare kan ha i systemet är följande:

- Verskamhetsgrupp med adepter
- Verskamhetsgrupp utan adepter
- Mentor
- Drop-in-task-force
- Alumn

Rollerna är baserad på en hierarkisk modell som är framtagen utifrån Intize nuvarande struktur av användare, se figur 4.11. Hierarkin definierar vilka rättigheter som ärvs, en mentor är alltså inte en drop-in-task-force men den har samma rättigheter. Rollen VG finns endast för att gruppera rättigheter för verksamhetsgruppen.



Figur 4.11: Den hierarkiska strukturen av roller som implementerats i applikationen. Arv av rättigheter visas med pilar, det är den mottagande rollens rättigheter ärvs.

Åtkomstkontrollen i webbapplikationen är implementerad på tre nivåer. Den första nivån är ett global nekande av tillträde till alla sidor förutom inloggningssidor för alla som inte är autentiserade. Den andra nivån sker hos grupper av *controllers*, dessa är logiskt placerade efter huruvida deras logik ska vara tillgänglig för en grupp användare. Den lägsta nivån sker på enskilda *controllers* och därmed på enskilda sidor.

4.6.4 Inmatningskontroll

Då Intize IT-lösning hade en osäker inmatningskontroll behövde detta åtgärdas i det nya systemet. Inmatning i systemet kontrolleras först vid inmatningen i formulär och senare vid insättning i databasen med hjälp av restriktioner.

Med inmatningskontrollen går det till exempel att byta ut användargränssnittet utan att lämna databasen sårbar för otillåten data. Då användaren troligtvis inte skulle förstå felmeddelanden direkt från databasen valdes det att även implementera felmeddelanden integrerat i användargränssnittet. På så sätt kan användaren få felmeddelanden i klartext, vilka gör det lättare att förstå.

4.7 Systemavgränsningar

Systemet som konstruerats i samband med rapporten är inte anpassat för att ha adepter och aspiranter med som användare i systemet. Systemet är inte heller integrerat med några externa tjänster förutom Google Apps.

Webbapplikationen är byggd för att fungera i webbläsare som följer HTML5- och CSS3-standarderna. Sidan fungerar även i Internet Explorer 8 med undantag för vissa grafiska effekter. Då stödet för olika delar av HTML5 och CSS3 varierar beroende på vilken webbläsare och upplösning på webbläsaren kan variationer i utseendet på systemet förekomma.

4.8 Användargränssnitt

För att skapa en första version av användargränssnittet togs en prototyp fram. Sedan uppdaterades användargränsnittet allteftersom användartester utfördes eller ny funktionalitet lades till.

Eftersom en webbapplikation skulle implementeras var valet att använda HTML, CSS och JavaScript givet. För att göra systemet lättillgängligt på mobiler, surfplattor och datorer gjordes systemet responsivt, det vill säga att layouten anpassas efter skärmstorlek. Att göra systemet responsivt innebar en extra komplikation under utvecklingen av gränssnittet då nya vyer hela tiden behövde testas på olika upplösningar.

Att anpassa det grafiska gränssnittet beroende på vilken roll användaren är inloggad som var en av de större utmaningarna i projektet. Samtidigt som menyn och vyerna behövde anpassas för olika roller skulle så lite källkod som möjligt dupliceras i vyer och *controllers*. Under en period i utvecklingen visades en uppsättning av helt olika menyer och vyer beroende på roll. Då detta innebar mycket duplicering av källkod löstes problemet genom att låta vy-lagret undersöka vilken roll användaren har och sedan anpassa den HTML som skrivs ut. Denna ändring eliminerade nästan all duplicering i *controllers* och minimerade dupliceringen i vyerna.

JavaScript används för att underlätta bland annat inmatning från användare. Exempelvis för att visa kalendrar i datumfält, göra det möjligt att lägga till flera kontaktpersoner i samma formulär och andra förbättringar av användargränssnitt. Det används dock sparsamt för att en användare utan JavaScript aktiverat ska ha möjlighet att använda all funktionalitet.

4.9 Användartester

Användartester genomfördes kontinuerligt under projektets gång för att få respons på det arbete som utförts, samt för att upptäcka problem med webbapplikationen så tidigt

som möjligt [25]. De tre kategorier av tester, som beskrevs i avsnitt 3.4, övervägdes i planeringsfasen av projektet.

Beslutet föll på att genomföra tester i en kontrollerad omgivning med en användare åt gången. Detta för att förhindra att personer eller oväntade händelser i omgivningen skulle störa användaren och därmed påverka utförandet. Målet var att testa användbarheten i applikationen och inte hur den fungerar i en naturlig miljö [25]. Genom att utföra testerna med en person i taget, skilt från resten av verksamhetsgruppen, förhindrades användarna från att påverka varandra. Användare inkluderades i testerna för att göra dem mer delaktiga i utvecklingsprocessen. Precis som Sharp, Rogers och Preece [25] hävdar ökar det sannolikheten att de senare är nöjda med produkten. Det viktigaste var dock att få respons på användargränssnittet och den funktionalitet som implementerades.

Testerna utfördes på Intize verksamhetsgrupp och mentorer. Totalt genomfördes tre stycken testserier på varje person i verksamhetsgruppen. Tre mentorer testade systemet i slutet av projektet. Alla tester leddes av två testledare för att öka chansen att fånga upp så mycket som möjligt av användarnas respons.

Testtillfällena utfördes genom att användaren fick ett antal uppgifter att slutföra och ombads att tänka högt under genomförandet. Den här metoden tillämpades för att få en uppfattning om användarens tankar kring applikationens funktioner och det grafiska gränssnittet. Testledarna tog anteckningar på vad användaren tyckte och hur navigationen genom applikationen gick till för varje uppgift. Användaren fick även uttrycka sina synpunkter och idéer under testtillfället. Utifrån användarens åsikter ställde testledarna frågor för att upprätthålla en dialog och samtidigt låta användaren motivera sina tankar. Testledarna studerade dessutom hur funktioner och grafiska element fungerade under användning. Detta för att upptäcka tänkbara problem. Efter varje testtillfälle utvärderades den respons som gavs för att göra eventuella omprioriteringar av kraven.

Inför det sista användartestet gjordes ett antal förändringar i användargränssnittet där funktionalitet flyttades mellan sidor. Det resulterade i att testpersoner från verksamhetsgruppen, som testat systemet tidigare, hade svårighet att guida sig fram då de vant sig vid det tidigare utseendet. Den nya verksamhetsansvariga, som aldrig sett applikationen innan, medverkade också i testet men hade inga problem med navigeringen.

4.10 Funktionella tester

Webbapplikationen genomgick funktionella tester för att säkra kvalitén, då de kunde ge svar på om ny källkod fungerade korrekt. Samtidigt förebyggde testerna att ny källkod förstörde den existerande funktionaliteten, som redan testats och bedömts som fungerande. Det underlättade även arbetet med att hitta buggar och producera ett stabilt system. Dock uppstod det problem med att hålla testerna i fas med implementeringen, eftersom ny funktionalitet lades till snabbare än testerna konstruerades. Detta berodde till stor del på att endast en utvecklare i taget skrev tester.

5 | Resultat

Det slutgiltiga resultatet är ett webbaserat administrativt system implementerat i programmeringsspråket PHP med ramverket Symfony. Systemet är implementerat enligt Jeff Offutts kriterier, som presenterades i avsnitt 3.1. Funktionaliteten följer den kravspecifikation som arbetats fram efter Intize önskemål, som återfinns i bilaga B.

En sammanfattning av antalet krav som har uppfyllts och implementeras i systemet presenteras nedan. Även skärmdumpar från tre olika plattformar visas för att demonstrera webbapplikationens responsiva design. Dessutom presenteras en översikt av användargränssnittet.

5.1 Uppfyllda krav

Av de 83 funktionella krav som återfinns i kravspecifikationen i bilaga C har 58 stycken blivit implementerade i webbapplikationen. Detta innebär att 69,9 procent av alla krav har uppfyllts. De krav som inte blivit implementerade är bland annat de som innefattar hantering av kurser. Dessa krav utelämnades då de hade låg prioritet, samt att det var oklart hur Intize ville att denna funktionalitet skulle se ut.

Exempel på funktionalitet som implementerades är hantering av användare, adepter, drop-in, evenemang och fredagsluncher. Hantering är i den här bemärkelsen möjligheten att lägga till, visa, redigera och ta bort medlemmar och aktiviteter. Applikationen erbjuder även publicering av nyheter som är tillgängliga för alla medlemmar i föreningen. Dessutom finns möjligheten att redigera globala inställningar.

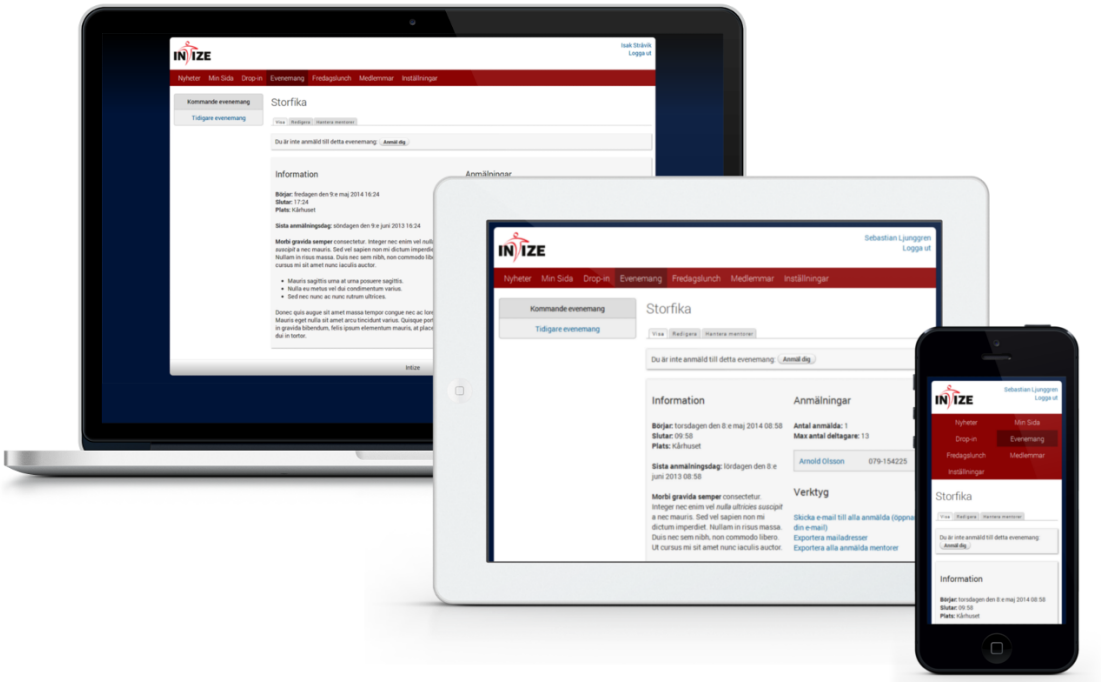
Utav de 5 icke-funktionella krav som listas i kravspecifikationen har 5 stycken uppfyllts, det vill säga 100 procent.

5.2 Användargränssnitt

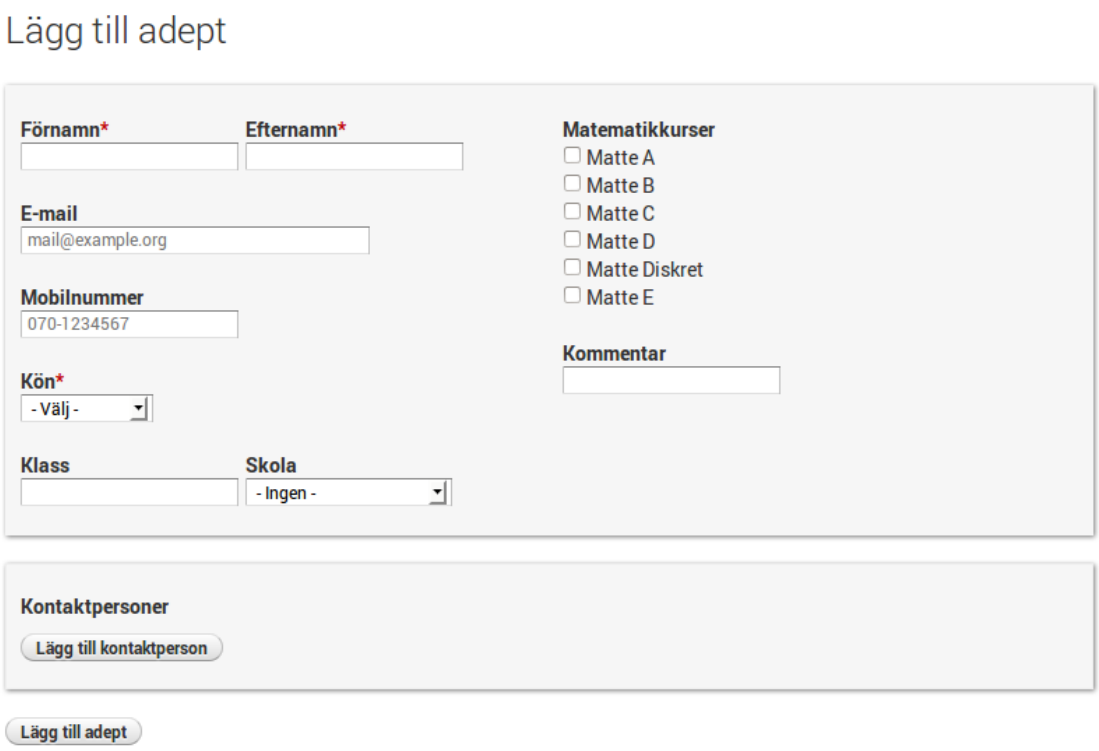
Användargränssnittet är, som figur 5.1 visar, byggt med responsiv design. Exempelvis anpassas de kolumner som visas i figur 5.4 och 5.2 vid olika skärmstorlekar. Öppnas sidan på en mobil visas allt innehåll i en kolumn. Ett annat exempel är att vid lägre sidbredder döljs mindre viktig information i tabeller för att göra tabellerna läsliga även vid lägre upplösningar.

I användargränssnittet har olika designmönster använts för att öka användarvänligheten och dra nytta av konventioner som redan är vanliga i webbapplikationer. Ett exempel på detta är sidomenyn som visas i figur 5.2 och figur 5.5 samt flikarna i figur 5.4 som använder sig av mönstret *visual framework*. Detta ger applikationen ett enhetligt och konsekvent utseende.

Figur 5.5 visar hur listor ser ut i webbapplikationen. Mönstret *row striping* används för att göra tabellen läsligare. I de listor där det är relevant visas en filterpanel för att kunna specificera vilken information som ska visas.

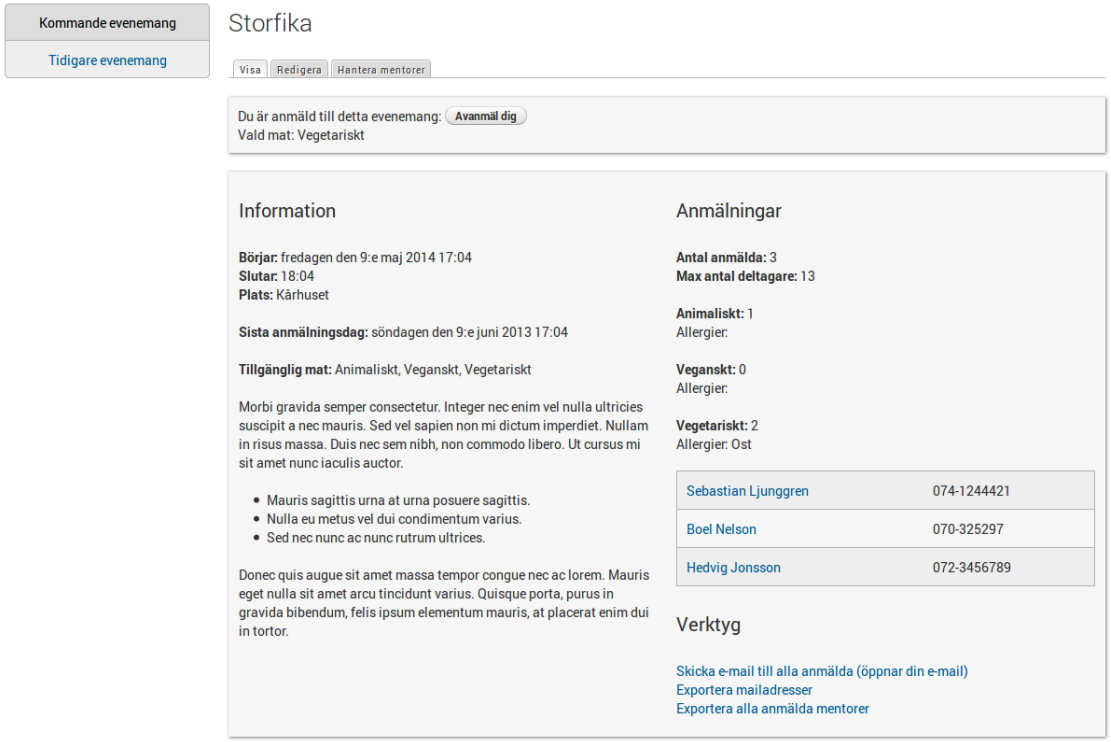


Figur 5.1: Demonstration av hur webbapplikationen med hjälp av responsiv design anpassar sig på olika enheter.

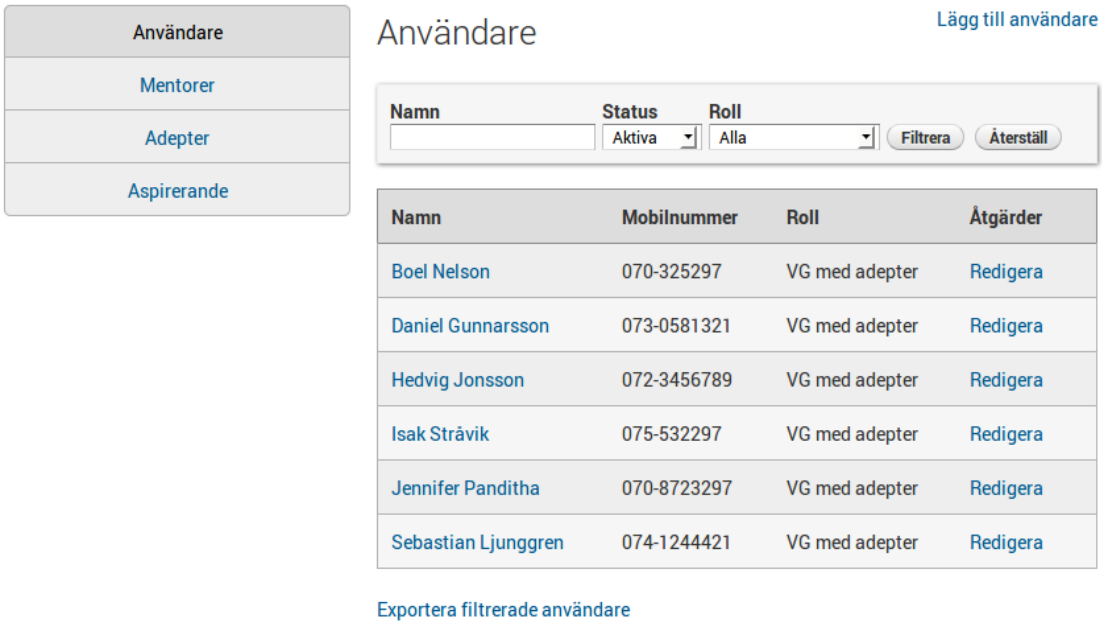


Figur 5.2: Formulär för att lägga till en adept. Kontaktpersoner läggs till dynamiskt med hjälp av JavaScript. Andra formulär i webbapplikationen har samma upplägg.

Figur 5.3: Export av adepter. Det är möjligt att anpassa vilken data som exporteras.



Figur 5.4: Vy för ett evenemang. Information om tid, plats och anmälda användare samt en beskrivning av evenemanget visas.



Figur 5.5: Listning av systemets användare med information och åtgärder.

6 | Diskussion

Vid planeringen och implementeringen av det administrativa systemet har flera val gjorts och argumenterats för. Dessa val har i stor utsträckning influerat slutprodukten, men även den agila arbetsprocessen har varit av betydelse. Då systemet kunde implementerats på flera olika sätt kunde systemet sett annorlunda ut. Därför diskuteras hur en alternativ implementering och arbetsprocess kunde ha påverkat slutresultatet.

6.1 Den agila arbetsprocessen

Om en annan arbetsmetod hade använts istället för den agila arbetsprocessen, som exempelvis en sekventiell arbetsmetod, skulle vissa problem kunnat ha undvikts. Ett sådant problem är hur modelleringen av databasen hanterats. Den agila arbetsprocessen medförde att krav på funktionalitet uppdaterades kontinuerligt. Det gjorde att problem uppstod då databasmodellen inte uppdaterades i samma takt som kravspecifikationen, vilket gjorde att modellen inte gav en korrekt bild av systemet. Om en sekventiell arbetsmetod använts istället skulle kravspecifikationen inte ändrats, och databasmodellen hade aldrig behövt ändras efter att den implementerats.

Ett annat problem som skulle kunnat ha undvikts med en sekventiell arbetsmetod är hur implementeringen av de funktionella testerna utförts. Då krav uppdaterades förändrades även delar av systemet vilket gjorde att tester behövde skrivas om. Då ny funktionalitet tillkom i snabbare takt än testerna uppdaterades, fungerade inte testerna som de skulle under vissa perioder. Om en sekventiell arbetsmetod tillämpats hade testerna inte blivit inaktuella i samma grad.

Detta talar emellertid inte för att den sekventiella arbetsmetoden är en bättre arbetsprocess. Detta på grund av att fördelarna med den agila arbetsprocessen, som beskrivs i avsnitt 4.1, väger tyngre än nackdelarna med den. Problemen har främst uppstått på grund av att den agila arbetsprocessen inte använts på rätt sätt.

För att bättre ta del av fördelarna hos agila metoder hade en specifik metod behövt appliceras strikt. Ett exempel hade varit att välja en testdriven arbetsprocess. På så sätt hade testerna skrivits före ny funktionalitet implementerades, vilket hade kunnat förhindra testerna från att komma ur fas. En nackdel hade dock varit att det hade krävts en ordentlig studie i ämnet innan arbetet hade kunnat påbörjas, vilket hade tagit längre tid.

6.2 Val av språk och ramverk

När valet av webbutvecklingsramverk skulle göras stod det mellan tre ramverk konstruerade i olika programmeringsspråk. I slutändan föll valet på Symfony på grund av att det är implementerat i PHP. Detta val baserades på både de praktiska och ekonomiska skäl som diskuterats i avsnitt 4.4.

Istället för att undersöka flera olika språk borde fokus legat på att jämföra olika PHP-ramverk, eftersom valet i slutändan ändå visade sig vara låst till PHP. Då kunde valet av ramverk ha baserats på dess tekniska meriter istället för vilket språk det är implementerat i.

Slutsatsen av den jämförelse av Symfony, Ruby on Rails och Django som genomfördes var att skillnaderna i funktionalitet och utvecklingsplattform var små. Valet skulle helt enkelt inte ha lett till stora skillnader i projektet. Dessutom hade skillnaden varit större mellan språken än mellan ramverken. Däremot kunde valet sett annorlunda ut om inte de ekonomiska och praktiska faktorerna varit av betydelse.

6.3 Prestanda i databasen

Systemets prestanda har inte varit en central fråga. Skulle den däremot fått högre prioritet hade en objektorienterad databas varit ett intressant alternativ. Den objektorienterade databasen, som nämns i avsnitt 3.2.2, har nämligen visat sig ha god prestanda i jämförelse med en relationsdatabas. Ett annat alternativ hade varit att endast ta bort ORM-lagret som systemet använder sig av. Detta skulle emellertid göra att systemet förlorar en del av sin flexibilitet då det inte skulle gå att byta databashanterare lika enkelt. Dessutom hade problem med *impedance mismatch* kunnat uppstå.

Eftersom systemet inte hade några krav på en hög prestanda begrundades dessa lösningar inte djupare. Begränsningarna med Intize webbhotell gjorde även att det inte var aktuellt att implementera systemet med en objektorienterad databas.

6.4 Brister i systemdesignen

Den applikationsstruktur som presenteras i avsnitt 4.5.2 skapar problem med onödiga beroendeförhållanden. De olika *bundles* som finns i systemet har nämligen ett flertal kopplingar till varandra. Ett sätt att lösa det skulle vara att strukturera om systemet för att använda färre *bundles*.

I det nuvarande systemet hanterar *controllers* nästan all logik vilket gör att dessa klasser är väldigt komplexa. En mer lättunderhållen arkitektur kunde erhållits om logik sorterats ut till modellen eller att *controllers* delats upp.

Vissa val i systemdesignen var från början ogenomtänkta. Detta var en effekt av att majoriteten av gruppen inte hade någon tidigare erfarenhet av Symfony eller webbutveckling. Med den kunskap som utvecklarna idag besitter hade bättre och mer informerade beslut om systemets arkitektur och design kunnat tas.

6.5 Ärendehanteringssystem

Ärendehanteringssystemet Redmine har på många sätt underlättat arbetet under implementeringen. Det går däremot i efterhand att diskutera om buggar och funktionalitet hanterades på ett passande sätt i Redmine. Då buggar och funktionalitet inte prioriterades inbördes var det svårt att avgöra i vilken ordning de skulle behandlas. Detta slutade med att buggar inte lades in i iterationer utan istället löstes efterhand.

En bättre lösning hade varit att ta både buggar och funktionalitet i beaktelse vid varje ny iteration och gett dem prioritet därefter. På så sätt skulle buggar kunna lösas mer effektivt istället för att sporadiskt lösas i en godtycklig ordning allteftersom projektet fortgick.

6.6 Dynamiska roller

Eftersom Intize verksamhetsgrupp innehåller flera olika tjänster skulle det ha varit intressant att tillåta administratörer att konfigurera nya roller dynamiskt. På så sätt skulle administratörer ha större kontroll över vilka användare som kan göra vad. Detta skulle även betyda att alla medlemmar i verksamhetsgruppen inte skulle behöva ha samma roll, utan dessa skulle kunna justeras efter ansvarsområde för att bättre spegla den interna hierarkin.

Dynamiska roller hade dock krävt att Intize verksamhetsgrupp hade mer insikt i webbapplikationens interna struktur för att kunna underhålla dem. Det hade även krävt mer tid under utvecklingsfasen för att producera ett avancerat gränssnitt och mer komplex åtkomstkontroll. Denna lösning hade inte varit möjlig att implementera under den begränsade tidsperioden för detta projekt. Däremot hade dynamiska roller sannolikt medfört en säkrare hantering av Intize data genom mer avgränsade ansvarsområden.

6.7 Användartester

För att komplettera de användartester som genomförts kunde även tester i den naturliga miljön utförts. Detta skulle eventuellt ge en utökad bild av hur systemet skulle användas i praktiken. Som nämnts i avsnitt 3.4 har tester i naturlig miljö flera fördelar. Denna typ av tester är emellertid mycket tidskrävande och hade inte kunnat utföras inom projektets begränsade tidsram. Dessutom var målet med testerna att undersöka användbarheten

hos systemet, vilket tester i kontrollerad miljö är bättre anpassade för. Det ansågs därför att dessa tester inte skulle bidra med ny information.

I efterhand insågs det att genomförandet av användartesterna kunde förbättrats. Först och främst upptäcktes det att testerna borde ha förberetts bättre. Det förekom att en funktion som skulle testas inte fungerade eller att den nödvändiga data som användaren skulle manipulera saknades. Istället borde testet kontrollerats mer noggrant och verifierats så att all data existerade innan testerna påbörjades. Ett annat problem var svårigheten för testledarna att veta när de skulle hjälpa användaren. Då användaren stötte på ett problem eller inte hittade en funktion var det oklart ifall testledarna skulle ingripa eller inte. Dessutom togs det inte ett beslut om vem av testledarna som eventuellt skulle ingripa, vilket kunde ge en uppfattning av ett ostrukturerat testtillfälle.

Trots en del brister i genomförandet har testerna tillfört mycket till utvecklingen. Tack vare testerna har missförstånd, brister och buggar tidigt kunnat upptäckas och åtgärdas. Responsen från användarna har gjort applikationen bättre anpassad till målgruppen.

7 | Slutsats

Webbapplikationen som konstruerats uppfyller en stor del av Intize önskemål. Målet har inte varit att implementera alla krav, utan fokus har legat på att implementera de viktigaste väl.

Det som är osäkert med systemet är om det kommer användas av Intize. Då önskemålen varit vaga och systemet ännu inte är i bruk, är det oklart hur väl det uppfyller Intize behov i praktiken.

Sammanfattningsvis kan webbapplikationen ersätta stora delar av den gamla lösningen, tack vare kontinuerligt testande och ett nära samarbete med Intize.

Källförteckning

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach och T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", juni 1999, RFC 2616. URL: http://www.ietf.org/rfc/rfc2616.txt (hämtad 2013-05-06).

[2] J. Offutt, "Quality attributes of web software applications", *IEEE Softw.*, vol. 19, nr 2, s. 25–32, mars 2002.

[3] R. Berjon, T. Leithead, E. Doyle Navara och S. Pfeiffer, "HTML 5.1: A vocabulary and associated APIs for HTML and XHTML, W3C Working Draft 17 December 2012", W3C, dec. 2012. URL: http://www.w3.org/TR/2012/WD-html51-20121217/ (hämtad 2013-05-16).

[4] World Wide Web Consortium. (2013). CSS Introduction, World Wide Web Consortium, URL: http://www.w3schools.com/css/css_intro.asp (hämtad 2013-05-20).

[5] ——, (2013). JavaScript Introduction, World Wide Web Consortium, URL: http://www.w3schools.com/js/js_intro.asp (hämtad 2013-05-20).

[6] The PHP Group. (2013). Object Inheritance, The PHP Group, URL: http://www.php.net/manual/en/language.oop5.inheritance.php (hämtad 2013-05-10).

[7] S. Guidetti. (2010). Understanding and applying polymorphism in php, Nettuts+, URL: http://net.tutsplus.com/tutorials/php/understanding-and-applying-polymorphism-in-php/ (hämtad 2013-05-17).

[8] The PHP Group. (2013). Namespaces overview, The PHP Group, URL: http://www.php.net/manual/en/language.namespaces.rationale.php (hämtad 2013-05-10).

[9] Symfony. (2013). Symfony components, Symfony, URL: http://symfony.com/components (hämtad 2013-05-13).

[10] F. Potencier. (2013). What is Symfony2?, URL: http://symfony.com/doc/2.2/quick_tour/the_architecture.html (hämtad 2013-05-06).

[11] Symfony. (2013). The architecture, Symfony, URL: http://symfony.com/doc/2.2/quick_tour/the_architecture.html (hämtad 2013-05-13).

[12] ——, (2013). Creating and using templates, Symfony, URL: http://symfony.com/doc/current/book/templating.html (hämtad 2013-05-19).

[13] Twig. (2012). Twig for developers, Twig, URL: http://twig.sensiolabs.org/doc/api.html (hämtad 2013-05-18).

[14] Symfony. (2013). Databases and Doctrine, Symfony, URL: `http://symfony.com/doc/current/book/doctrine.html` (hämtad 2013-05-19).

[15] The Doctrine Project. (2013). Welcome to the Doctrine Project, The Doctrine Project, URL: `http://www.doctrine-project.org/` (hämtad 2013-05-19).

[16] H. Garcia-Molina, J. D. Ullman och J. Widom, *Database Systems: The Complete Book*, 2. internationella utg. New Jersey: Prentice Hall, 2008.

[17] P. van Zyl, D. G. Kourie och A. Boake, "Comparing the performance of object databases and orm tools", i *Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, ser. SAICSIT '06, Somerset West, South Africa: South African Institute for Computer Scientists och Information Technologists, 2006, s. 1–11.

[18] K. Hedström och K. Larsson, "Objektorienterade databaser", examensarb., Lunds Universitet, Sverige, 2006.

[19] C. Ireland, D. Bowers, M. Newton och K. Waugh, "A classification of object-relational impedance mismatch", i *Advances in Databases, Knowledge, and Data Applications, 2009. DBKDA '09. First International Conference on*, 2009, s. 36–43.

[20] The Open Web Application Security Project (OWASP). (2012). Testing for ORM Injection (OWASP-DV-007), URL: `https://www.owasp.org/index.php/Testing_for_ORM_Injection_%28OWASP-DV-007%29` (hämtad 2013-05-16).

[21] ——, "OWASP Top 10 - 2010. The Ten Most Critical Web Application Security Risks", tekn. rapport, 2010. URL: `http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202010.pdf`.

[22] A. Adams och M. A. Sasse, "Users are not the enemy", *Commun. ACM*, vol. 42, nr 12, s. 40–46, dec. 1999.

[23] E. D. Hardt, "The OAuth 2.0 authorization framework", okt. 2012, RFC 6749. URL: `http://tools.ietf.org/html/rfc6749` (hämtad 2013-05-10).

[24] OpenID Foundation. (2013). Welcome to OpenID Connect, OpenID Foundation, URL: `http://openid.net/connect` (hämtad 2013-05-15).

[25] H. Sharp, Y. Rogers och J. Preece, *Interaction Design: Beyond Human-Computer Interaction*, 3. utg. Chichester: Wiley, 2011.

[26] J. Nielsen. (2012). Thinking Aloud: The #1 Usability Tool, Nielsen Norman Group, URL: `http://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/` (hämtad 2013-05-06).

[27] N. Ferreira, J. Biddle Jennifer och Robert, "Agile development iterations and ui design", *Software, IEEE*, s. 50–58, aug. 2007.

[28] The Doctrine Project. (2013). Security, The Doctrine Project, URL: `http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/security.html` (hämtad 2013-05-16).

[29] Google Developers. (2012). Choosing an Auth Mechanism, Google, URL: `https://developers.google.com/accounts/docs/GettingStarted` (hämtad 2013-05-10).

[30] ——, (2013). Using OAuth 2.0 for Login, Google, URL: `https://developers.google.com/accounts/docs/OAuth2Login` (hämtad 2013-05-12).

A | Intervjufrågor samt svar

Intervjuer genomfördes med mentorer i början av utvecklingsprocessen. Frågorna och resultatet av intervjuerna är presenterade här. Resultaten är sammanställda åsikter och kan representera åsikten av flera mentorer.

Hur tycker ni att det fungerar med anmälan till fredagsluncher?

- Kan vara jobbigt att byta till Intize inloggningen för att anmäla sig.
- Kan ej se om man är anmäld.
- Bra att man blir påmind av mejl att man ska anmäla sig.
- Avanmälning är jobbigt, även innan deadline och måste ske genom mejl.
- Enkelt.
- Automatiskt utskick så att verksamhetsgruppen slipper skriva varje gång.

Hur tycker ni att det fungerar med anmälan till drop-in?

- Otydlig med avanmälning till drop-in, mentorer avanmäler sig sent.
- Man kanske kan låsa schemat för att mentorer sen ska behöva ringa för att avanmäla sig.
- Påminnelse att man har registrerat sig.
- Bra att man kan se hur många som redan anmält sig.
- Svårt att hitta länken till anmälan.
- Otydligt med max/min antalet mentorer som kan anmäla sig.
- Otydliga avanmälan.
- Ingen deadline.

Har ni varit drop-in-ansvariga? Om ja, hur tycker ni det fungerar att rapportera till Intize efter tillfället?

- Förslag: Utvärderingsformulär för ansvariga efter drop-in där antal närvarande och andra detaljer ska fyllas i.
- Tyckte det var bra och fick en mall för inrapporteringen, tyckte dock att det nog var jobbigt för mottagande part.

- Måste manuellt skickas ut av drop-in-ansvarig.

Läser ni/Har ni läst kursen Matematik och Samhälle? Om ja, hur tycker ni det fungerade med administrationen av kursen?

- *Inlämning av arbeten?*
  - Inlämning var smidigt, fick respons och bekräftelse på kurshemsidan.
  - Om man får retur borde nästa deadline tydligt visas.
  - Vill ej ha fler inloggningar att komma ihåg.
  - Kan tänka sig annat system än mejl.
- *Rapportering av timmar?*
  - Smidigt med formulär, lättanvänt.
  - Inget konstigt.
- *Få information?*
  - Hemsidan uppdateras inte så ofta, man får mer information i mejlen än vad som finns på hemsidan.
    - Om man har missat en föreläsning borde info om den finnas på hemsidan.
    - Dålig uppdatering på sidan, information kommer sent. Man får mejl med info men på hemsidan står det inte samma sak.
    - Bra med mejl, bra länkat på kurshemsidan.
    - Ibland dålig framförhållning på när material ska läsas.

Är det något som ni vill tillägga?

- Hemsidorna borde vara sammanlänkade så att man lätt kan hitta till dem.
- Det är enkelt och gör det som ska göras redan idag.
- Kurshemsidan borde länkas från Studentportalen.
- Om mejlnotifikationer: ska gå att stänga av.
- Tydligare anmälan för adepter om att få mentor (är gömt nu på hemsidan).
- Visa kötiden tills man får en mentor.
- Elever kommer till drop-in då det inte är något. Kanske ska stå på hemsidan att drop-in pågår mellan vissa veckor. Uppdatera hemsidan oftare.

B | Önskemål från Intize

Följande punkter är de önskemål som tillhandahölls av Intize i början av utvecklingsprocessen. Det är dessa önskemål som kravspecifikationen utgick ifrån.

Intranät som skall:

- Vara kopplat till Intizes Google Apps
- Fungera som en informationskanal för föreningens samtliga medlemmar
- Erbjuda mentorer följande:
  - Rapportera timmar
  - Antal inrapporterade timmar
  - Anmälan till drop-in
  - Skapa felärenden
- De mentorer som läser kursen ska kunna:
  - Få nyheter om kursen
  - Se vilka delar av kursen de är godkända i
- Erbjuda verksamhetsgruppmedlemmar följande:
  - Alla:
    - Ärendehantering
  - Drop-in-ansvarig:
    - Se bemanning på drop-in
    - Statistik över bemanning
  - Mentorskapsansvarig:
    - Databas över adepter och mentorer
    - Skapa en kö med adepter
  - Verksamhetsansvarig och viceverksamhetsansvarig:
    - Följa projekt
    - Kunna kommentera

    - Kursansvarig:
        - Rapportera in resultat
        - Tillgängliggöra kursmaterial
        - Se vilka som är godkända/underkända
        - Inlämningar
        - Kurshemsida
    - IT-ansvarig:
        - Kunna administrera, uppdatera och förändra systemet
- Övrigt:
    - Behörighetssystem som är integrerat med det i Google Apps
    - Vara lättöverskådligt
    - Vara väldokumenterat
    - Logga in genom Intize hemsida (intize.org)
    - Anmälningssystem för adepter
    - Hemsidan för Intize

C | Kravspecifikation

Kraven i detta dokument är uppdelade efter vad olika aktörer kan göra i systemet. Rollerna är hierarkiska och är i stigande ordning: alumn, drop-in-task-force, mentor och verksamhetsgruppmedlem, även kallad VG mentor. Till exempel gäller alla krav för en alumn även en mentor. Utöver det finns även en sorts verksamhetsgruppmedlem som ärver från drop-in-task-force istället för mentor, denna kallas VG drop-in-task-force.

Funktionella krav

1 Alumn

1.1 Logga in i systemet

En användare ska kunna logga in i systemet.

1.2 Se nyheter

En alumn ska kunna se nyheter.

1.3 Se personlig information

En alumn ska kunna se all sin personliga information.

2 Drop-in-task-force

2.1 Generellt

2.1.1 Lista kommande drop-in-tillfällen

En medlem i drop-in-task-force ska kunna se en lista över kommande drop-in-tillfällen.

2.1.2 Lista tidigare drop-in-tillfällen

En medlem i drop-in-task-force ska kunna se en lista över alla drop-in-tillfällen som passerat.

2.1.3 Se drop-in-tillfälle

En medlem i drop-in-task-force ska kunna se information om ett enskilt drop-in-tillfälle.

2.1.4 Anmäla sig till drop-in-tillfällen

En medlem i drop-in-task-force ska kunna anmäla sig till kommande drop-in-tillfällen.

2.1.5 Se tidigare drop-in-tillfällen

En medlem i drop-in-task-force ska kunna se tidigare drop-in-tillfällen som han eller hon har närvarat vid.

2.1.6 Se kommande drop-in-tillfällen

En medlem i drop-in-task-force ska kunna se kommande drop-in-tillfällen som han eller hon är anmäld till.

2.1.7 Avanmäla sig från drop-in-tillfälle

En medlem i drop-in-task-force ska kunna avanmäla sig från kommande drop-in-tillfällen om det sker en viss tid i förväg.

2.1.8 Lista kommande fredagslunch

En medlem i drop-in-task-force ska kunna se nästa fredagslunch.

2.1.9 Lista tidigare fredagsluncher

En medlem i drop-in-task-force ska kunna se en lista över alla fredagsluncher som passerat.

2.1.10 Se fredagslunch

En medlem i drop-in-task-force ska kunna se information om en fredagslunch.

2.1.11 Anmäla sig till en fredagslunch

En medlem i drop-in-task-force ska kunna anmäla sig till en fredagslunch.

2.1.12 Avanmäla sig från en fredagslunch

En medlem i drop-in-task-force ska kunna avanmäla sig från en fredagslunch som han eller hon anmält sig till.

2.1.13 Välja mat vid registrering till en fredagslunch

Om det finns mat tillgänglig på en fredagslunch ska medlemmen i drop-in-task-force kunna välja vilken typ av mat hon eller han vill ha.

2.1.14 Lista kommande evenemang

En medlem i drop-in-task-force ska kunna se en lista över alla kommande evenemang.

2.1.15 Lista tidigare evenemang

En medlem i drop-in-task-force ska kunna se en lista över alla evenemang som passerat.

2.1.16 Se evenemang

En medlem i drop-in-task-force ska kunna se information om ett evenemang.

2.1.17 Anmäla sig till ett evenemang

En medlem i drop-in-task-force ska kunna anmäla sig till ett evenemang.

2.1.18 Avanmäla sig från ett evenemang

En medlem i drop-in-task-force ska kunna avanmäla sig från ett evenemang som han eller hon anmält sig till.

2.1.19 Välja mat vid registrering till ett evenemang

Om det finns mat tillgänglig på ett evenemang ska medlemmen i drop-in-task-force kunna välja vilken typ av mat hon eller han vill ha.

2.1.20 Skapa felärenden

En medlem i drop-in-task-force ska kunna skapa felärenden som skickas till verksamhetsgruppen.

2.1.21 Lista kurser

En medlem i drop-in-task-force ska kunna lista alla kurser.

2.1.22 Se kurs

En medlem i drop-in-task-force ska kunna se information om en kurs.

2.1.23 Anmäla sig till kurs

En medlem i drop-in-task-force ska kunna anmäla sig till en kurs.

2.2 Kursregistrerade

2.2.1 Se nyheter om kurs

En kursregistrerad ska kunna se nyheter om kursen i systemet.

2.2.2 Se godkända delar i kurs

En kursregistrerad ska kunna se vilka delar av en registrerad kurs som han/hon är godkänd i.

2.2.3 Anmäla sig på kurstillfällen

En kursregistrad ska kunna anmäla sig på kurstillfällen och göra ett val av kost.

2.2.4 Ladda upp filer

En kursregistrerad ska kunna ladda upp filer för inlämning.

3 Mentorer

3.1 Generellt

3.1.1 Rapportera mentorstimmar

En mentor ska kunna rapportera in sina timmar i systemet.

3.1.2 Se mentorstimmar

En mentor ska kunna se sina inrapporterade timmar.

3.1.3 Se adepter

En mentor ska kunna se sina adepter och deras kontaktuppgifter.

3.1.4 Se mentorskapskoordinator

En mentor ska kunna se sin mentorskapskoordinators kontaktinformation.

3.2 Mentorskapskoordinator

En koordinator är en mentor som har en mindre grupp mentorer och agerar kontaktperson för dessa.

3.2.1 Se mentorer i sin grupp

En mentorskapskoordinator ska kunna se vilka mentorer som tillhör hans eller hennes grupp

4 Verksamhetsgruppmedlem

4.1 Lägg till en adept

En verksamhetsgruppmedlem ska kunna lägga till en adept

4.2 Lista adept

En verksamhetsgruppmedlem ska kunna se en lista över alla adepter.

4.3 Se adept

En verksamhetsgruppmedlem ska kunna se alla adepter i systemet med all information tillgänglig.

4.4 Redigera adept

En verksamhetsgruppmedlem ska kunna ändra en adepts kontaktinformation.

4.5 Inaktivera adept

En verksamhetsgruppmedlem ska kunna inaktivera en adept i systemet. Adepten ska då inte kunna tilldelas en mentor.

4.6 Ta bort adept

En verksamhetsgruppmedlem ska kunna ta bort en adept från systemet.

4.7 Lägg till aspirerande mentor

En verksamhetsgruppmedlem ska kunna lägga till en aspirerande mentor.

4.8 Lista aspirerande mentorer

En verksamhetsgruppmedlem ska kunna se en lista över alla aspirerande mentorer.

4.9 Se aspirerande mentor

En verksamhetsgruppmedlem ska kunna se en aspirerande mentors kontaktinformation.

4.10 Redigera aspirerande mentor
En verksamhetsgruppmedlem ska kunna redigera en aspirerande mentors kontaktinformation.

4.11 Uppgradera aspirerande mentor till mentor
En verksamhetsgruppmedlem ska kunna uppgradera en aspirerande mentor till en mentor.

4.12 Ta bort aspirerande mentor
En verksamhetsgruppmedlem ska kunna ta bort en aspirerande mentor permanent ur systemet.

4.13 Lägg till mentor
En verksamhetsgruppmedlem ska kunna lägga till en mentor i systemet.

4.14 Se alla mentorer
En verksamhetsgruppmedlem ska kunna se alla mentorer i systemet med all information tillgänglig.

4.15 Redigera mentor
En verksamhetsgruppmedlem ska kunna redigera en mentors kontaktinformation och roll.

4.16 Inaktivera mentor
En verksamhetsgruppmedlem ska kunna inaktivera en mentor i systemet, vilket gör att mentorn inte längre kan logga in i systemet, anmälas till aktiviteter eller tilldelas adepter.

4.17 Ta bort mentor
En verksamhetsgruppmedlem ska kunna permanent ta bort mentorer från systemet.

4.18 Lägga till adept till mentor
En verksamhetsgruppmedlem ska kunna tilldela en adept till en mentor.

4.19 Ta bort adept från mentor
En verksamhetsgruppmedlem ska kunna ta bort en adept från en mentor.

4.20 Se antalet anmälda till drop-in-tillfälle
En verksamhetsgruppmedlem ska kunna se antalet anmälda till ett drop-in-tillfälle.

4.21 Se antalet anmälda till evenemang
En verksamhetsgruppmedlem ska kunna se antalet anmälda till ett evenemang, om mat finns ska det även stå hur många som har valt vilket mat.

4.22 Se antalet anmälda till fredagslunch
En verksamhetsgruppmedlem ska kunna se antalet anmälda till en fredagslunch, om mat finns ska det även stå hur många som har valt vilken mat.

4.23 Lista skolor
En verksamhetsgruppmedlem ska kunna se en lista över alla skolor.

4.24 Lägg till skola
En verksamhetsgruppmedlem ska kunna lägga till en skola.

4.25 Ändra skola
En verksamhetsgruppmedlem ska kunna ändra en skola.

4.26 Ta bort skola
En verksamhetsgruppmedlem ska kunna ta bort en skola.

4.27 Lista mattyper
En verksamhetsgruppmedlem ska kunna se en lista över alla mattyper.

4.28 Ändra mattyper
En verksamhetsgruppmedlem ska kunna ändra en mattyp.

4.29 Lägg till mattyp
En verksamhetsgruppmedlem ska kunna lägga till en mattyp.

4.30 Ta bort mattyp
En verksamhetsgruppmedlem ska kunna ta bort en mattyp.

4.31 Lista matematikkurser
En verksamhetsgruppmedlem ska kunna se en lista över alla matematikkurser.

4.32 Ändra matematikkurs
En verksamhetsgruppmedlem ska kunna ändra en matematikkurs.

4.33 Lägg till matematikkurs
En verksamhetsgruppmedlem ska kunna lägga till en matematikkurs.

4.34 Ta bort matematikkurs
En verksamhetsgruppmedlem ska kunna ta bort en matematikkurs.

4.35 Ändra globala inställningar
En verksamhetsgruppmedlem ska kunna ändra värden på de globala inställningarna.

4.36 Se statistik En verksamhetsgruppmedlem ska kunna se statistik över bemanningen på tidigare drop-in-tillfällen.

4.37 Skapa projekt
En verksamhetsgruppmedlem ska kunna skapa ett projekt.

4.38 Lägga till medlemmar i projekt
En verksamhetsgruppmedlem ska kunna lägga till medlemmar till ett projekt.

4.39 Se alla projekt
En verksamhetsgruppmedlem ska kunna se alla projekt som sker på Intize, vilka medlemmar som deltar samt deras information.

4.40 Kommentera projekt
En verksamhetsgruppmedlem ska kunna kommentera på projekt.

4.41 Hantera timrapporter
En verksamhetsgruppmedlem ska kunna hantera timrapporter för anställda.

4.42 Exportera sökresultat
En verksamhetsgruppmedlem ska kunna exportera sina sökresultat i csv-format.

5 Kursansvarig

Kursansvarig är en i verksamhetsgruppen som har hand om en kurs.

5.1 Rapportera resultat i en kurs
Kursansvarig ska kunna rapportera in en mentors resultat i en kurs.

5.1 Tillgängliggöra information om kursen
Kursansvarig ska kunna tillgängligöra information om kursen till mentorer.

5.3 Se godkända och underkända i en kurs
Kursansvarig ska kunna se vilka deltagare i en kurs som är godkända och vilka som är underkända.

5.4 Hantera inlämningar
Kursansvarig ska kunna hantera inlämningar från kursen.

5.5 Hantera kurshemsida
Kursansvarig ska kunna hantera kursen hemsida.

6 Projektmedlemmar

Projektmedlemmar är användare i systemet som har tillsammans påbörjat ett projekt.

6.1 Lägga till projektnyheter
En projektmedlem på Intize ska kunna lägga till nyheter på projekt de tillhör.

Icke-funktionella

1. Autentisering
Autentisering ska ske med hjälp av Google-inloggningar.

2. Sista anmälningsdatum till drop-in-tillfällen
Sista anmälningsdatumet ska vara globalt, efter att detta passerat kan endast verksamhetsgruppmedlemmar anmäla personer till drop-in-tillfället.

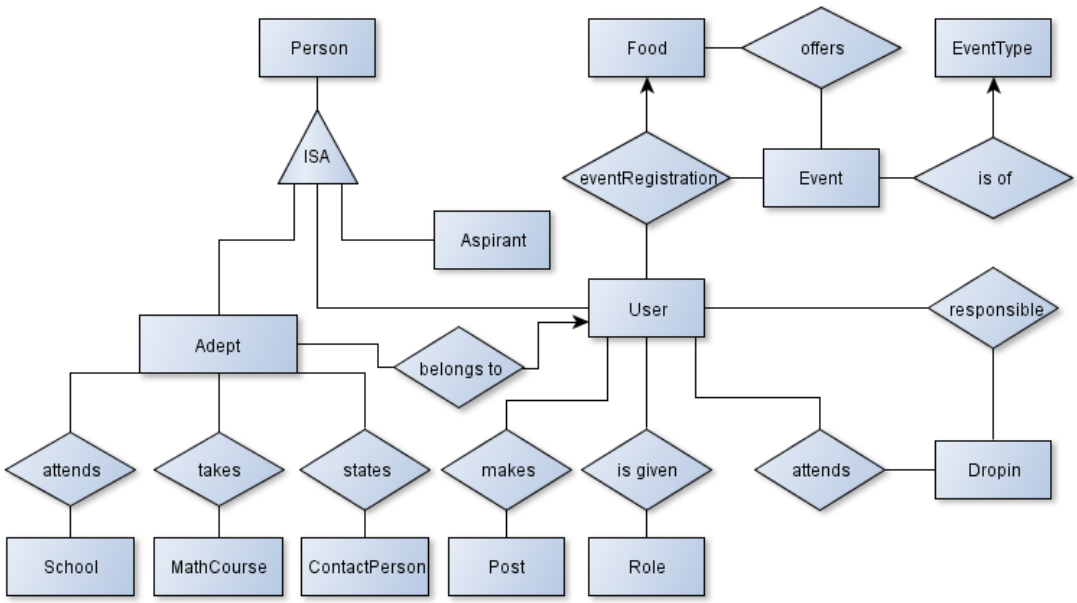
4. Sista avanmälningsdatum till drop-in-tillfällen
Sista avanmälningsdatumet ska vara globalt, efter att detta passerat kan endast verksamhetsgruppmedlemmar avanmäla personer till drop-in-tillfället.

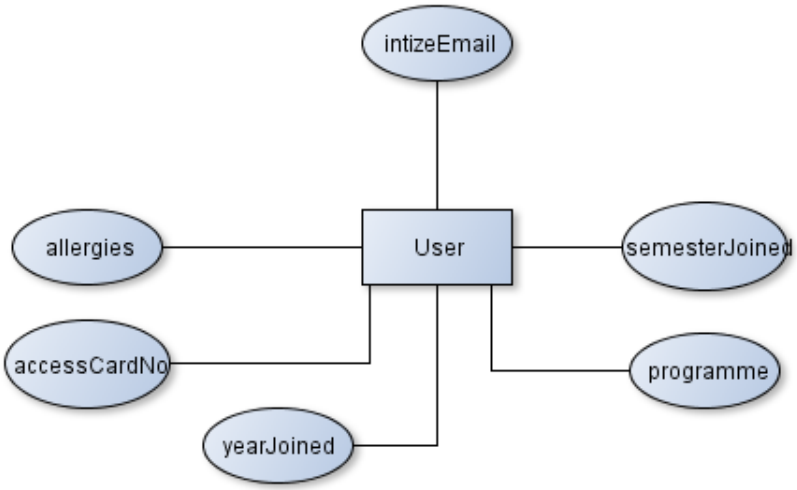
5. Sista anmälningsdatum/avanmälningsdatum till fredagsluncher
Sista anmälningsdatumet/avanmälningsdatumet ska vara globalt, efter att detta passerat kan endast verksamhetsgruppmedlemmar anmäla/avanmäla personer till fredagslunchen.

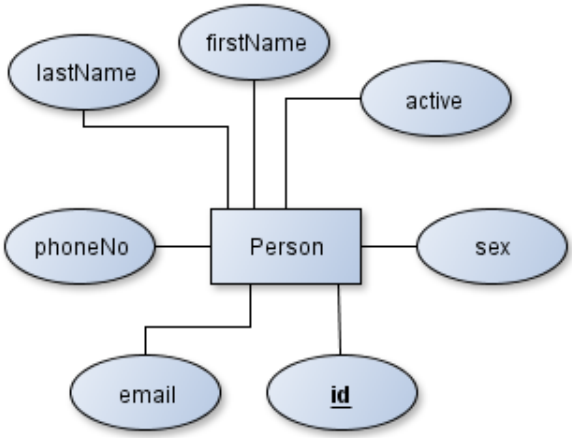
6. Sista anmälningsdatum/avanmälningsdatum till ett evenemang
Sista anmälningsdatum/avanmälningsdatum ska finnas för varje evenemang, efter att detta passerat kan endast verksamhetsgruppmedlemmar anmäla personer till evenemanget.

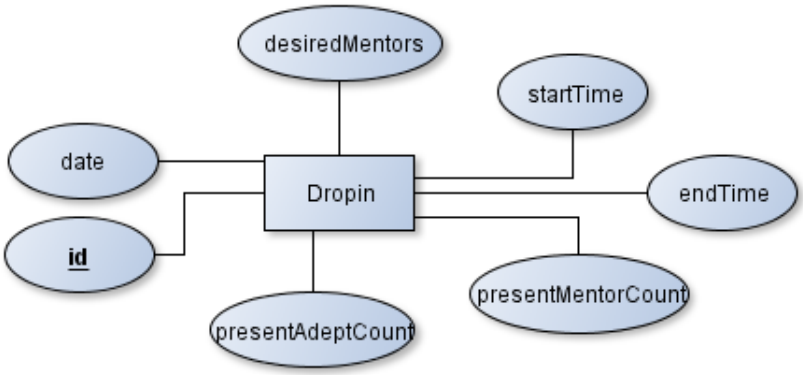
D | Databasmodellen

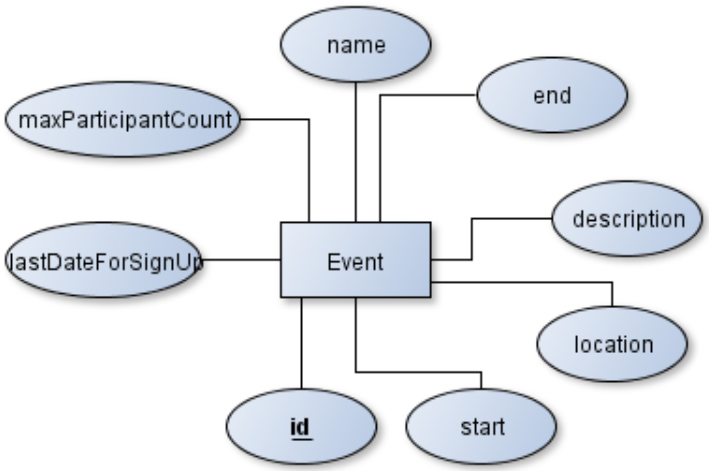
I denna bilaga visas hela designen av databasen. Först presenteras en översiktsbild med relationer mellan alla entiteter. Därefter visas varje entitet med sina attribut.

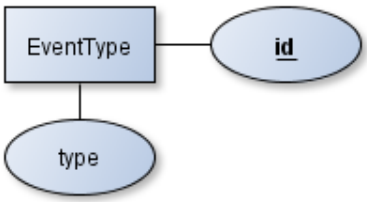


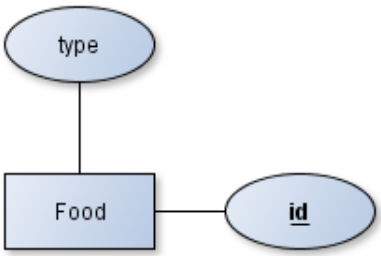

intizeEmail
allergies
User
semesterJoined
accessCardNo
programme
yearJoined

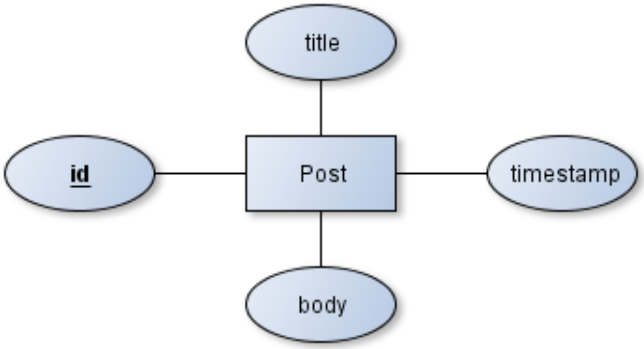

firstName
lastName
active
phoneNo
Person
sex
email
id

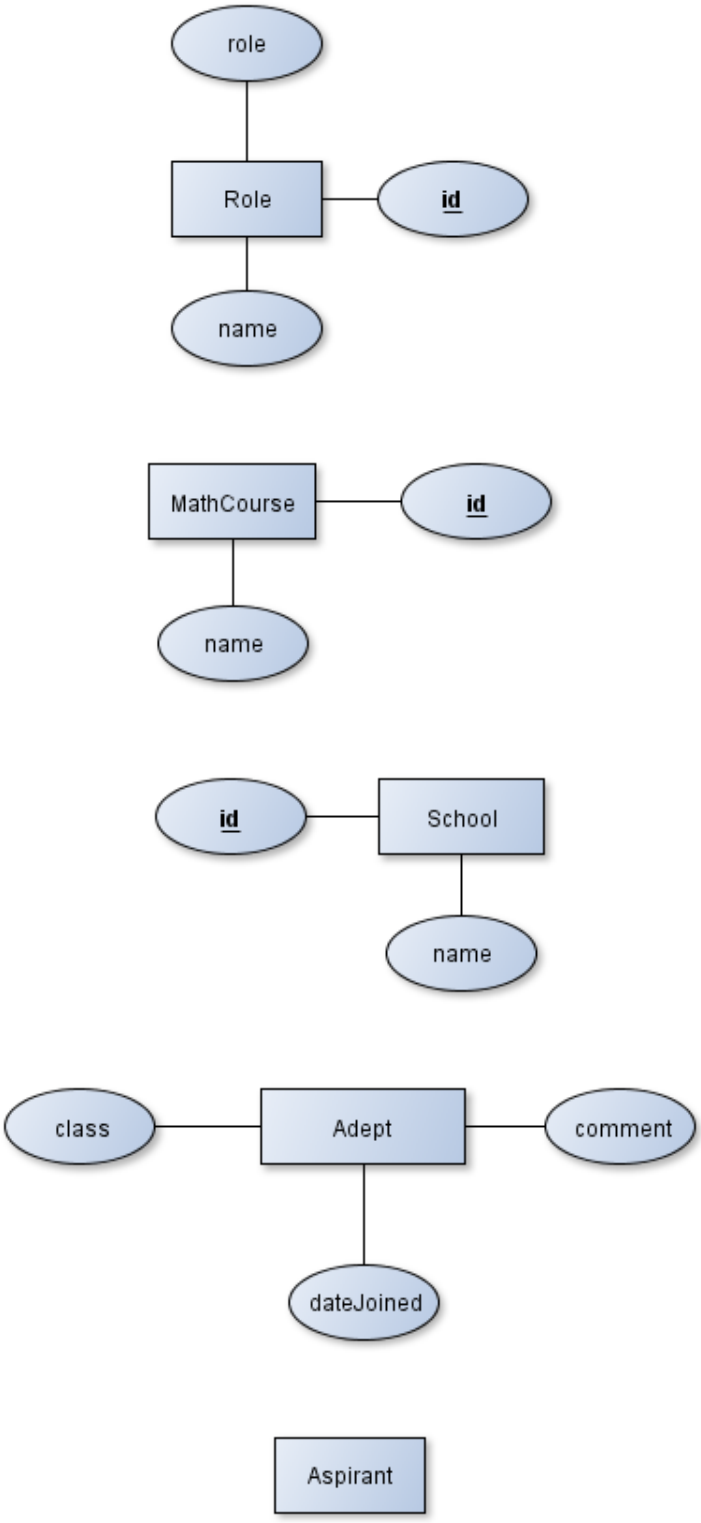

desiredMentors
startTime
date
Dropin
endTime
id
presentMentorCount
presentAdeptCount

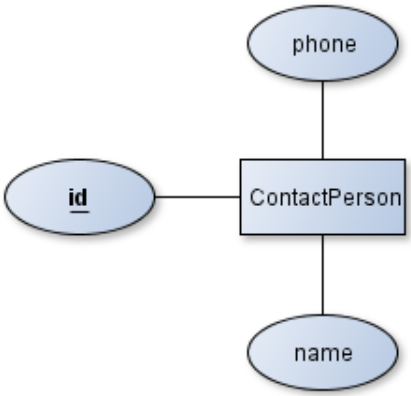

name
end
maxParticipantCount
description
lastDateForSignUp
Event
location
id
start


EventType
id
type


type
Food
id


title
id
Post
timestamp
body


role
Role
id
name
MathCourse
id
name
id
School
name
class
Adept
comment
dateJoined
Aspirant


phone
id
ContactPerson
name