**TitelSida**

* **Titel/projektnamn(var tydlig och informativ -skriv gärna i två led)**
* **Rapportbeteckning: Projektrapport/Examensarbete**
* **Författare/projektdeltagare**
* **Utbildningsprogram, utbildningsanordnare, ort och datum**

Sammandrag

**Ett sammandrag av hela rapporten på svenska.**

Training-avdelningen på Alten har beställt en applikation för att hantera kurser som de tillhandahåller. Idag används excel-dokument som skickas fram och tillbaka mellan alla involverade. Det är detta arbetssätt de vill komma ifrån. De vill ha all information samlad på en och samma plattform.

Denna rapport kommer gå igenom de delar vi har bidragit med under vår andra lia-period för att lösa detta.

Den första uppgiften vi fick var att implementerat roller för de olika användarna av systemet. Totalt fem olika roller som var och en ska ha olika behörigheter.

Vi har också implementerat en sökningsfunktion där en kan söka igenom alla kurstillfällen.

En kan söka på kursnamn, kursnummer, status, tidsintervall och plats. Detta krävdes för att underlätta arbetsflödet och lätt kunna hitta de önskade kurstillfällena.

Vi har skapat en budget-vy där en kan skapa individuella budgetar för alla säljare för en månad, en totalbudget för alla sälj en månad samt en för alla öppna kurser. En kan även se alla sålda kurstillfällen av en säljare för en månad.

Utöver detta har vi även skapat en pingfunktion som skickar mail till relevant person t.ex. när ett kurstillfälle börjar bli färdigt för fakturering så meddelas invoice-avdelningen att de ska gå in i systemet och skriva in workorder nummer för det kurstillfället. Vi har dessutom lagt till att det dyker upp en varningsruta ifall en med rollen ”invoicing” inte har klickat i knappen ”ifylld av invoice” när de är klara med att fylla i alla workordernummer.

För att helt slippa skicka dokument emellan sig har vi även gjort så att det går att bifoga dokument under alla kurstillfällen.

Vi har lagt upp systemet på en testmiljö där alla som ska använda applikationen kan gå in och testa så att det fungera som de vill.

Då vi har haft mycket möten med de tänkta användarna av systemet har vi även skrivit lite kring hur detta har gått till.

Utöver detta har det varit en hel del smågrejer som vi

Förord

* **Kort om de praktiska förutsättningarna för projektets genomförande och rapportskrivning.(Utbildningsprogram, övergripande syfte med projektuppdraget, tid m.m)**
* **Eventuella begränsingar i genomförandet anges här, t.ex. Saker man inte hunnit med eller av andra anledningar inte kunna genomföra.**
* **Eventuella tack till personer/organisationer eller företag som varit till värdefull hjälp för projektets genomförande. En hjälpsam handledare namges och tackas här.**

Detta examensarbete är den sista delen av den tvååriga YH utbildningen “Java Enterprise Utvecklare” på YRGO på Lindholmen i Göteborg. Vi har skrivit arbetet på Alten i Göteborg under våran andra LIA-period.

LIA-perioden var 17 veckor lång men examensarbetet avser de första 10 veckorna då vi efter dessa fick ett nytt uppdrag. På grund av den flytten hann vi inte alla punkter i målsättningen.

Vi vill tacka teamet på Altens training avdelning då de har givit oss väldigt bra feedback och haft förtroende för det arbete vi utfört. Ett extra tack till Giorgio Di Feola för en mycket bra handledning genom projektets gång.

**Innehållsförteckning**

* **Alternativt figur- eller tabellförteckning. Läggs direkt efter innehålsföteckningen.**
* **Bilder, figurer och tabeller ska numreras och beskrivas i teckt både inne i arbetet(***texten ska då vara kursiverad och mindre (t. ex. 10 punkter) än övrig löpande text i rapporten***) och i förteckningen.**

Innehållsförteckning

[1 Inledning 1](#_Toc7176434)

[1.1 Bakgrund 1](#_Toc7176435)

[1.1 Syfte och mål 1](#_Toc7176436)

[1.2 Avgränsingar 2](#_Toc7176437)

[1.3 Metod 2](#_Toc7176438)

[1.3.1 Verktyg 3](#_Toc7176439)

[2. Utveckling 4](#_Toc7176440)

[2.1 Säkerhet och behörighet 4](#_Toc7176441)

[2.1.1 Back-end(B) 4](#_Toc7176442)

[2.1.2 Front-end(F) 5](#_Toc7176443)

[2.2 Sökning 7](#_Toc7176444)

[2.2.1 Back-end 7](#_Toc7176445)

[2.2.2 Front-end 8](#_Toc7176446)

[2.3 Budget 9](#_Toc7176447)

[2.3.1 Back-end 9](#_Toc7176448)

[2.3.2 Front-end 11](#_Toc7176449)

[2.4 Sålda kurser 12](#_Toc7176450)

[2.4.1 Back-end 12](#_Toc7176451)

[2.4.2 Front-end 12](#_Toc7176452)

[2.5 Fakturerad kursinstans 12](#_Toc7176453)

[2.5.1 Back-end 12](#_Toc7176454)

[2.5.2 Front-end 12](#_Toc7176455)

[2.6 Pingfunktion 12](#_Toc7176456)

[2.6.1 Back-end 12](#_Toc7176457)

[2.6.2 Front-end 13](#_Toc7176458)

[2.7 Bifoga dokument 14](#_Toc7176459)

[2.7.1 Back-end 14](#_Toc7176460)

[2.7.2 Front-end 14](#_Toc7176461)

[2.8 Varning ifall workordernummer inte är ifyllt 14](#_Toc7176462)

[2.8.1 Front-end 14](#_Toc7176463)

[2.9 Testmiljö 15](#_Toc7176464)

[2.10 Kundkontakt 16](#_Toc7176465)

[Bilagor 16](#_Toc7176466)

# Inledning

## Bakgrund

**Sammanhang, problemmotivering, behov av en undersökning eller lösning**

Alten har en avdelning, Training, som arbetar med att sälja kurser främst till andra företag men även internt samt öppna kurser. I dagsläget sker planeringen av dessa kurser i Excel-dokument som skickas fram och tillbaka mellan alla personer som är involverade. Detta arbetssätt har varit rörigt och lätt att det blivit fel.

Training har då önskat ett system som de ska kunna använda för att planera kurser med alla kostnader och intäkter, kommunicera med Invoice samt göra budgetar.

Projektet påbörjades av lia-studenter 2017. Henning och Max blev tilldelad projektet under båda våra lia-perioder.

Efter 10 veckor av vår andra lia-period blev vi dock tilldelade ett annat projekt så vi har totalt arbetat med detta i 20 veckor. Examensarbetet omfattar dock endast det vi gjort under vår andra lia-period.

Arbetet skedde agilt med kundmöten nästan veckovis. Under dessa möten visade vi upp vad som gjorts och diskuterade vad vi skulle fokusera på här näst för att så snabbt som möjligt nå en första version som skulle kunna användas.

Varför vi har implementerat de olika funktionerna

## Syfte och mål

**Övergripande syfte och specifikt slutmål**

* **Formulera vilket/vilka problem du avser t arbeta med/belysa/lösa och avgränsa ditt problemområde så exakt som möjligt.**
* **Klargör också ditt slutmål, vad du vill åstakomma med projektarbetet, samt vad du hoppas man ska kunna använda resultatet till.**

Syftet har varit att få fram en användbar applikationen som personalen på Alten Training kan förlita sig på helt. Applikationen skall användas i olika syften beroende vilken roll man har på Alten.

Målet vi hade var alltså att få ut en första version av systemet som skulle kunna användas och processen att gå över till applikationen helt påbörjas.

För detta krävdes

* Säkerhet och behörighet- De olika användarna av systemet ska kunna göra olika saker. För att implementera roller ska även säkerhets biten ses över och delvis bytas ut.
* Sökning på kurser det ska gå att söka på kursnummer, plats, kursnamn samt datumintervall.
* Hämta ut kunder och leverantörer från internt system.
* Budget-vy - I denna vyn ska budgetar kunna sättas för säljare, öppna kurser och en total budget. Visa vad som sålts, vilken vinstmarginal som finns samt vem som sålt vad.
* Testmiljö - Sätta upp en testmiljö där alla har tillgång att testa systemet och se att det fungerar som tänkt.
* Namngivning - Reda ut vad allt ska kallas. T.ex ska “utgifter” kallas “kostnader”
* Pingfunktion - Det ska gå att pinga Invoice när ett kurstillfälle är redo för fakturering samt ifall ett fakturerbart kurstillfälle behöver förändringar.

Kunden hade även önskningar för att arbetet i systemet skulle gå smidigare som vi skulle göra i mån av tid.

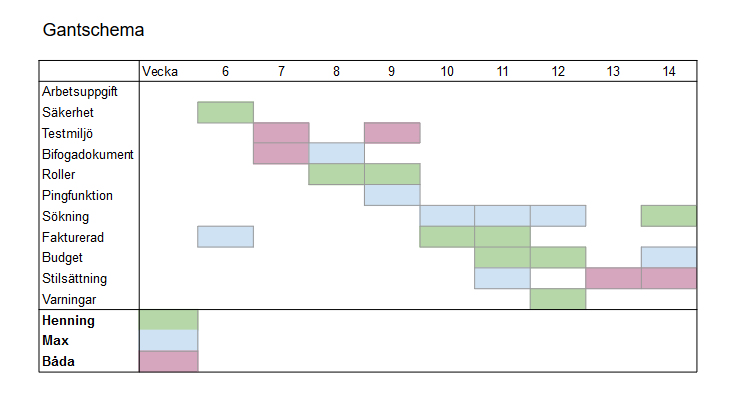
* Bifoga dokument. Detta ska gå att bifoga och ladda ner dokument för varje kurstillfälle.
* Stilsättning - Utefter en UX-designers rekommendationer.
* Varning i fall en med rollen invoice försöker lämna sidan där den fyller i workordernummer och inte har meddelat training.
* Buggar - t.ex datum sparas fel och fel kurser visas i kalendern. Se till att uträkningarna räknar på rätt saker.

## Avgränsingar

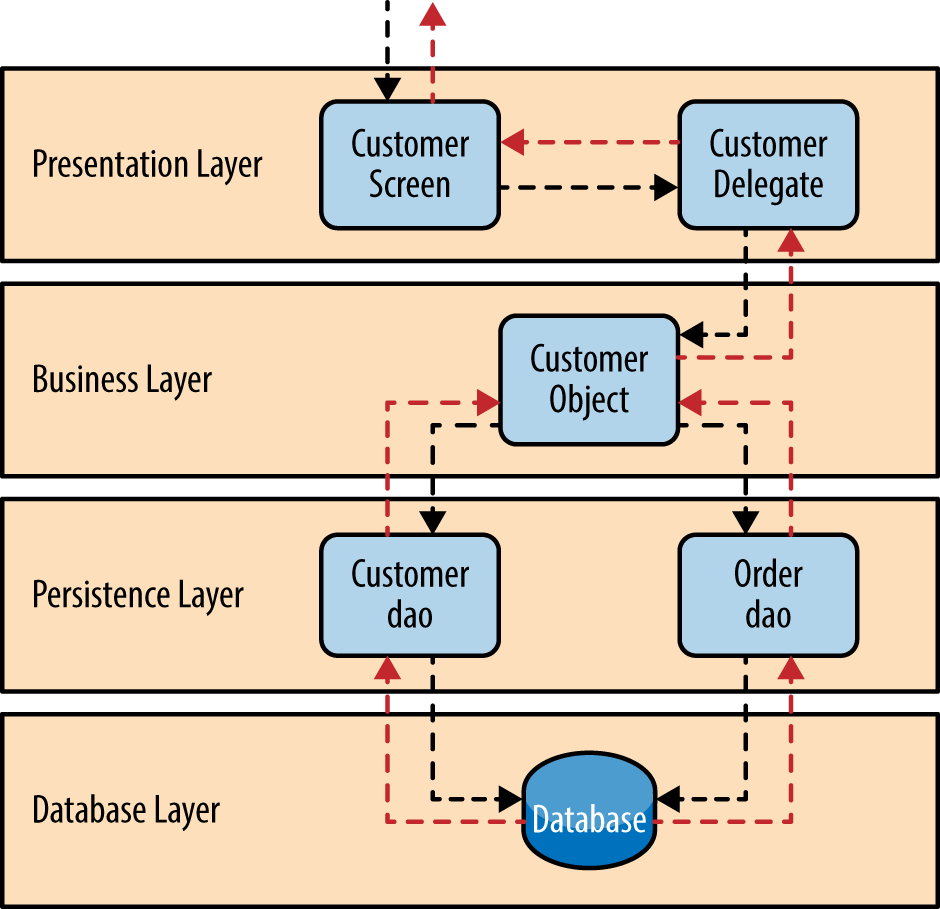
## Metod

**-Beskriv hur du tänker lägga upp arbetet och vilket program/verktyg du avser använda. Ta upp vilka delmoment som ingår i arbetet, i vilken ordningn dessa ska utföras, samt hur mycket tid(absolut eller i procent) som du har avsatt för varje moment. Lägg upp ett tidsschema.**

**-Vid grupp/lagarbete anges även anvsarsområden, dvs vem som ansvarar för vad.**

****

Back-end i den här applikationen är skriven med Layered Archiceture vilket betyder att den har envägs beroende från Rest-anrop till databas lagret. Detta illusteras i fig1.



### Verktyg

#### Utvecklingsverktyg

* Git - Vi använde oss utav Git för versionshantering. Vi använde oss även utav git flow för hantera grenar av projektet.
* Scrum - Vi hade dagliga standups med handledare och gick igenom vad vi gjort sedan igår, vad vi ska göra idag och om vi har några problem. För ärendehantering använda vi oss utav Jira.
* Texteditor - Vi använde oss utav IntelliJ samt utav Visual Studio Code.

#### Teknologier

* Node
* Npm
* Angular 5
* Spring Boot
* Maven
* MySQL
* WinSCP
* Putty

# Utveckling

## Säkerhet och behörighet

När vi tog över detta projekt var säkerhetskoden rörig och det låg kvar kod som inte används. När vi då skulle implementera roller var det svårt att få in det i koden som fanns så vi bestämde oss för att börja om. Vi kom fram till att JWT - JSON Web Tokens skulle fungera bra. Detta sammanflätat med Spring-security. Med JWT skickas en autentiseringsnyckel (token) där vi även kunde baka in rollen som respektive användare har. I front-end kan då denna autentiseringsnyckel avkodas för att få fram hur länge koden är giltig och vilken roll en användare har. [[1]](#footnote-2)[[2]](#footnote-3)

[Avkodad autentiseringsnyckel](#_Avkodad_autentiseringsnyckel)

### Back-end(B)

Spring-security gör det lätt att konfigurera säkerheten. Vi gör detta genom att ärva klassen WebSecurityConfigurerAdapter där vi använder metoden configurer(). Där sätts vilka ändpunkter som kräver autentisering vilket i detta fall är alla förutom login.

Här kan vi även sätta vilka roller som ska ha tillgång till vilka ändpunkter. Detta kan även göras på controller-nivå. Vi skapar också ett filter som kontrollerar alla autentiseringsnycklar som skickas med i anropen till back-end.[[3]](#footnote-4)

Då detta projekt endast ska användas här på Alten prioriterades inte att låsa ändpunkterna för olika roller i back-end. Låsningen av applikationen för olika roller görs alltså endast i front-end förtillfället.

I entiteten för användare sätts ett fält med en lista av roller. Vi kan alltså ha flera roller men det är låst i front-end till att endast kunna ha en roll då en användare för tillfället endast är tänkt kunna ha en roll.

Rollerna sätter vi som Enums och dessa implementerar GrantedAuthority från Spring-security som behövs för att skapa en autentiseringsnyckel i vår klass TokenProvider. I TokenProvider kontrolleras även ifall de inkomna autentiseringsnycklar är godkända, dvs att de matchar den ursprungligt skapade autentiseringsnyckeln och att utgångsdatumet för autentiseringsnyckeln inte har passerats.[[4]](#footnote-5)[[5]](#footnote-6)

Varje gång en loggar in i applikationen skapas och skickas en ny autentiseringsnyckel som vi sedan sparar i front-end och skickar med i alla resterande anrop till back-end.

### Front-end(F)

I front-end görs ett anrop till login-ändpunkten och ifall svaret skickar status-kod 200 sparar vi token i local-storage och sedan skickar en vidare till rätt sida för rätt roll. De olika rollerna har alltså olika startsidor.

Vi har en klass TokenService som avkodar autentiseringsnyckel och kan på så vis få fram vilken roll en användare har.

Denna klass har en metod hasRole() som returnerar en boolean, true ifall den inloggade användaren har den satta rollen och kan då användas för att begränsa delar av applikationen för de olika rollerna. T.ex. ska navbaren se olika ut beroende på vilken roll en har. Där kan vi alltså låsa olika flikar med metoden hasRole(). Ifall en flik endast ska visas för en med rollen admin kan vi alltså sätta *\*ngIf*="hasRole*([*roles.*admin])*".

För att låsa hela vyer använder vi istället en RoleGuard som vi sätter i våran routing-fil där vi också specificerar vilka roller som är behöriga för en viss vy.

Ifall en då försöker skriva in en adress för en vy en inte har behörighet för så hamnar en istället på en sida där det står att behörighet saknas. Är token inte giltig längre så hamnar en på inloggningssidan.[[6]](#footnote-7)

När en loggar ut raderas autentiseringsnyckel från local-storage och inloggningssidan visas.

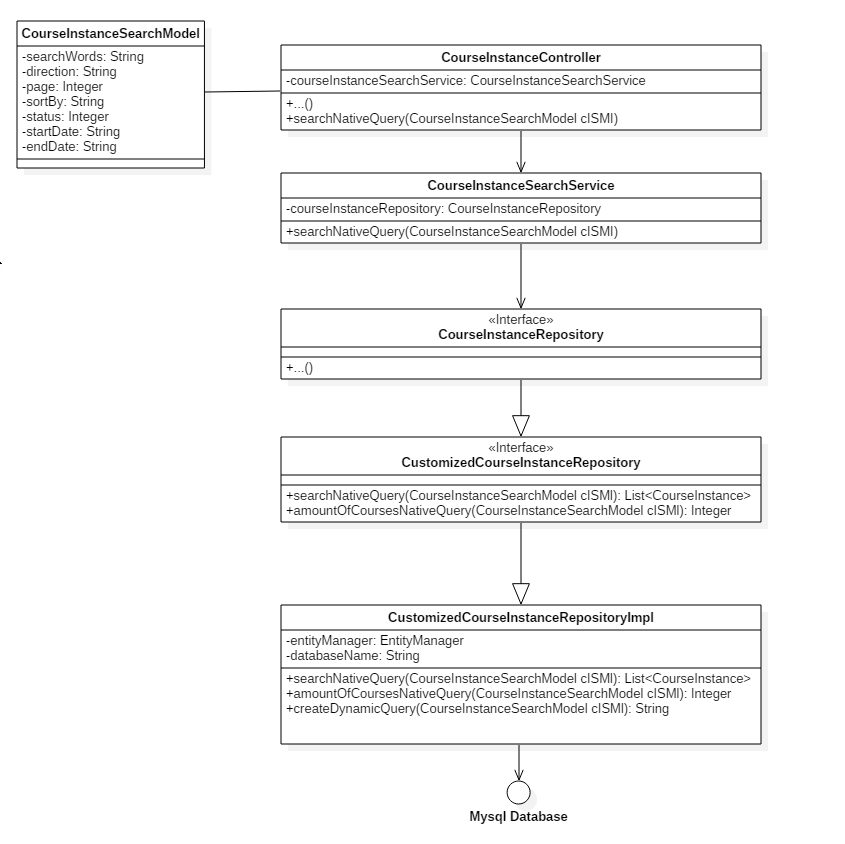
## Sökning

En viktig del som efterfrågades var att kunna söka och sortera på kurser. Sökfunktion har funnits i appen tidigare men då var den implementerad på ett sätt som inte var skalbart. Hur det fungerade då var att front-end hämtade upp alla kurser och sedan sökte man igenom en array av kurser på de sökord man ville. Det fungerade bra när applikationen hade få kurser. Men det gick ganska fort innan det inte var hållbart. Målet var att flytta sökningen till back-end och skicka upp ett visst antal kurser. Kraven för sökning:

* Kunna söka på kursnummer, plats, kurs(B)
* Kunna sortera på kursnummer, plats, kurs, datum samt kursstatus(B)
* Kunna filtrera på ett visst intervall(B)
* Kunna filtrera på en viss status(B)
* Paginering dvs skicka upp ett specifikt antal kurser(B)
* Minne, med minne menas att om man klickar in på en kurs skall man komma tillbaka till samma sökning.(F)

Samtliga krav blev uppfyllda, valet av metod stod mellan att använda Spring Data JPAs standard metoder och att skriva egen mysql native query. Vi valde det senare av anledning att ha kontroll på vad man skall kunna söka på och att sökningen skulle bli så dynamisk som möjligt. Mysql har stöd för indexsökning vid fält som har fulltext påslaget. Se exempel i bifogade filer de fält som har fulltext har nyckeln “mul”.Funktionen är att front-end skickar ett sökord samt parametrar såsom datum, status och antal. Back-end filtrerar ut och skickar tillbaka resultatet.

Sökningen implementerades i två etapper. Första etappen skickades endast sökord och parametrar som hämtas från URI:n men detta blev inte skalbart. Den slutgiltiga versionen använder istället en modell som specificerar all information i en sökning. Se fig2.

****

Figur 2. Flödet för en sökning, alla metoder är inte specificerade.

### Back-end

Som i fig1 tar back-end emot en sökningsmodell som går hela vägen ner till servicelagret CustomizedCourseInstanceRepositoryImpl (CCIR) där logiken sker. För att kunna skapa dynamiska sökningar blev vi tvungna att skapa en egen datakatalog(repository). Kursinstansens datakatalog ärver från den egenskapade för att kunna implementera egna frågor mot databasen.

Klassen CCIR har tre metoder, ge en lista på alla kurstillfällen, antalet kurser som sökningen ger och metoden med frågan mot databasen. Den sistnämnda separerar alla ord på mellanslag och lägger in dem i en lista. Sedan går den igenom listan sedan beroende på antal ord och i parametrar skapar metoden en fråga som sedan returneras. Vad som gör den dynamisk är att oavsett antal ord kommer den förlänga frågan mot databasen. Skulle man inte ge några parametrar alls hämtar den allt från tabllen.

En liten men kraftfull detalj är att vi har specificerat olika databasnamn beroende på ifall applikationen är i produktion, test eller utveckling.

Vidare fortsätter metoden att sortera ut på datum och status, StringBuffer används för att lägga till delar till frågan. Ett exempel på hur en fråga kan se ut när man skickat med två ord, datum och status som parameter:

SELECT **\*** **FROM** training.course\_instance

**WHERE** MATCH(course\_number,location,course\_name)

AGAINST('*40\**' **IN** **BOOLEAN** MODE)

**AND** MATCH(course\_number,location,course\_name)

AGAINST('*bas\**' **IN** **BOOLEAN** MODE)

**AND** status LIKE 0

**AND** start\_date **>=** '*2017-4-4*'

**AND** end\_date **<=**'*2019-4-27*'

I ovan stående exempel har metoden fått två inkompletta ord men med hjälp av ”IN BOOLEAN MODE” kollar den bara ifall fälten innehåller det som skrivs in. Metoden har även ett antal if-satser som kollar ifall det finns status, start och slut datum.

### Front-end

Det skulle vara lätt att söka på de olika kurserna i applikationen front-end är utvecklad på det viset att användaren ser en input ruta och två fält för att filtrera på datum. Front-end har en likadan sökmodell och beroende på var man är i applikationen kommer rätt parametrar fyllas i.

En viktig funktion i Front-end var att om man sökte på något måste den sparas så länge den informationen är relevant. Exempelvis gör användaren den sökningen som utförs i fig 2 , man presenteras av en lista med kurstillfällen och klickar in på en. Skulle det vara fel kurs och man vill återgå till listan vill man behålla samma sökresultat. Detta är ett problem som vi löste med hjälp av webbläsarens lokalaminne(localstorage), som har fyra metoder setItem(), getItem(), **removeItem()** och **clear()**. Vi sparade ner sökmodellens variabler i localstorage och sedan återanvände den informationen när man gick tillbaka på sidan.

## Budget

Kunden vill kunna sätta budgets för alla sina säljare, öppna kurser samt totalt. Kunden ville även kunna se hur mycket en säljare sålt, hur långt det är kvar till målet samt sätta kommentarer.

### Back-end

Vi fick börja med att länka ihop varje såld kursanmälan med en säljare. En kursanmälan heter i back-end CompanySignup. Varje CompanySignup innehåller alltså ett säljar-Id.

En budget entitet gjordes där vi sätter alla fält som behövs sparas, t.ex månad och en säljares id. Vi har en unik nyckel som säger att det inte får finnas två budgetar med samma säljare i samma månad och samma budgetTyp.

Vi byggde så att alla budgetar för en månad skapas samtidigt. Alltså för en för varje säljare, en för öppna kurser och en total.

Vi gjorde en modell, MonthlyBudgetModel, som vi populerar och skickar till front-end. Denna populeras med resultat, alltså t.ex. hur mycket som sålts totalt under en månad, hur långt det är kvar till målet, både i procent och SEK.

Tre querys till databasen gjordes för att få ut de CompanySignups som behövdes för att räkna ut resultat.

En för att hämta alla CompanySignups som en säljare sålt en viss månad,

en för allt från en öppen kurs en viss månad och en för alla sälj en viss månad.

Exempel:

@**Query**(

value **=** "*SELECT \* FROM company\_signup where course\_instance\_id in*

*(select id from course\_instance*

*where end\_date >= ?2*

*and end\_date <= ?3*

*and (status = 0 or status = 1 or status = 2 or status=4))*

*and salesperson like ?1*",

nativeQuery **=** true)

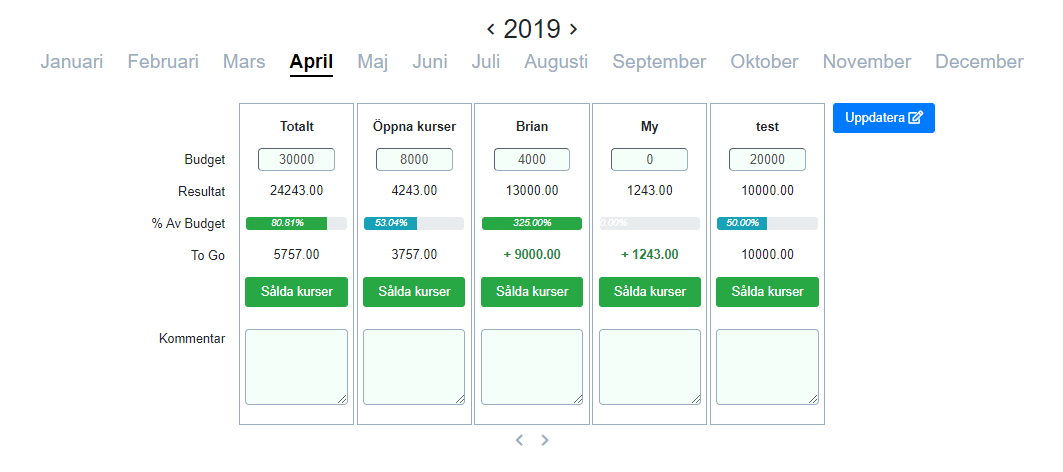
**List**<**CompanySignup**> getAllCompanySignupPerMonthBySalespersonId(**Long** id, **Date** firstDayOfMonth, **Date** LastDayOfMonth);

Status beskriver i vilket stadie en kurs är, dvs planerad, godkänd, fakturerbar, arkiverad eller avbruten.

Vi sorterar sedan så att den totala budgeten kommer först, sedan öppna kurser och därefter säljarna sorterade på namn. Detta för att underlätta hur vi visar upp vår data i front-end.

### Front-end

En vy har gjorts där en väljer vilka budgetar som ska visas med år och månad. Ifall en budget inte är skapad visas en knapp “Skapa budget” och när denna trycks visas de nyskapade budgetarna. Ifall budgetarna redan är skapta visas de direkt när en klickar in på den månaden.



Figur 3. Budget-vy

När en klickar på “Uppdatera” kan en budget sättas eller ändras samt att det går att lägga till kommentarer under varje budget.

Vyn krävde mycket CSS för att efterlikna en mall som en UX-designer gjorde åt oss. Max 6 budgetar visas i taget. Finns fler kan pagineringspilarna användas.

## Sålda kurser

Utöver Budget vyn ville kunden även kunna se sålda kurser per månad per person som syns i figur 3. Detta var ett senare önskemål som vi hann implementera. Applikationens centrum är kurstillfället, den har mest information men även nästlad information i form av lärare, företag, kostnader, internadeltagare checklistor ect. Vilket gör att i en lista som denna vill man inte hämta upp all information som finns i kurstillfället. Varför vi ville belysa det här exemplet var för att mycket av arbetet är att hämta upp exakt den information man behöver inte mer eller mindre. Tidigare versioner av applikationen har inte varit skalbar på grund av det.

### Back-end

Det utmanande med den här funktionen var att öppna kurser innehåller deltagare en säljare har sålt. Detta måste presenteras korrekt. Vyn innehåller den relevanta informationen för att hämta ut alla sålda kurser år, månad, säljare. Men kopplingen var tidigare inte implementerad. Vi skapade en listmodell med den datan som kunden efterfrågade. Sedan samman fogade informationen från kurstillfället och företagsanmälan i metoden [getSoldCourseInstances](#_getSoldCourseInstances).

### Front-end

På samma sätt skapde vi en modell i Front-end som tog emot datan och listade den i en tabell som i figur 3.

## Fakturerad kursinstans

### Back-end

### Front-end

## Pingfunktion

Vad som menas med pingfunktion är att personerna som arbetar med applikationen skall kunna notifiera varandra när en uppgift är utförd. Det är två olika avdelningar som arbetar under planeringsstadiet av en kurs. Avdelningen Training planerar kursern, och avdelningen Invoice fakturerar för kursen. Pingfunktionen skickar mail till respektive avdelning när ett stadie är utfört.

### Back-end

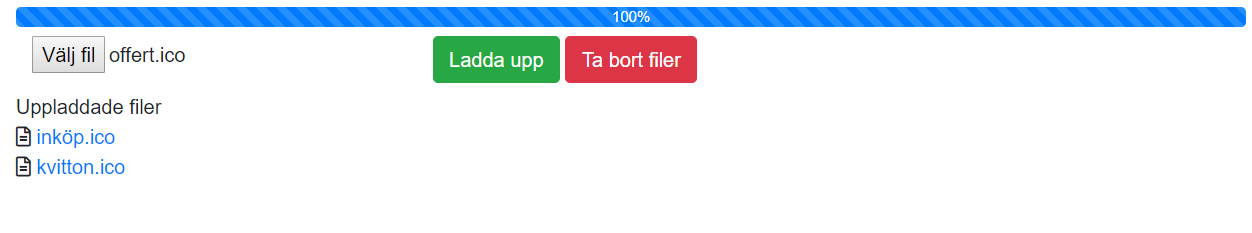
Springramverket har stöd email och är väldigt lätt att konfigurera. Man använder sig av JavaMailSender som är ett interface för alla olika typer av sätt att skicka mail. Sedan kan man skapa en egen modell men vi använde oss av springs SimpleMailMessage som är en standardklass i **springmail** där vi använde följande:

* Från
* Till
* Ämne
* Meddelande

Det är en förhållandevis enkel setup och den använde vi för att man ska kunna skicka obegränsat med email. I filen [application properties](#_Application_properties) innehåller konfigurations filen för mail springmail. Den kan konfigureras olika beroende på vilken profil man använder. Själva logiken för mail ut skicket görs med klassen [mailservice](#_MailService). Det finns betalnings api:er som möjliggör en mycket enklare setup men då kostar det pengar efter ett visst antal mail.

### Front-end

## Bifoga dokument



Figur exempel på uppladdade filer.

Vid planering av en kurs var det viktigt för kunden att kunna spara dokument som var tillgängligt för alla som behandlade kursen. Dokumenten skulle följa med i alla stadier och sparas per kurs. För detta krävdes ett sätt att kunna spara filerna på ett skalbart sätt oberoende av vilken storlek filen hade.

### Back-end

Vi valde att spara ner filerna i namngivna kataloger lokalt på datorn/servern istället för att spara ner dem i databasen. Fördelarna vi såg med att spara ner dem lokalt på datorn dels att det är mycket enklare att utöka minnet på en server. Kraven specifiserade inte storleken på filerna eller typen därför underlättade det att spara dem på lokalt. Men det är också enklare att exportera filerna direkt från servern. Produktions servern är nämligen en windows remote server som gör att en person som inte är insatt enkelt kan hämta filerna från servern.

I nu läget körs applikationen på tre olika sätt utveckling, testmiljö (AWS server) och produktionsmiljlö (Windows server). Filerna måste kunna sparas på olika platser, vi använde miljövariabler för att specificera var den skulle placeras.

Att [spara ner](#_StorageService_spara) filen använde vi en springramverkets MultipartFile som bryter ner filen i mindre delar, javascript har stöd för formatet vilket gör det väldigt lätt att skicka. Klienten skickar filen samt kursnumret som används till katalognamnet då numret är unikt.

Att [lista filerna](#_StorageService_lista_alla) på kurssidan gjordes genom att skicka en lista med filernas namn till klienten som sedan presenterade det i en tabell. Klienten skickar kursnumret till backend

För att [ladda ner/hämta](#_StorageService_ladda_upp) en fil var det enklare att gå ifrån MultipartFile och istället använda byte-array som filtyp. Metoden tar emot filnamnet och kursnumret konverterar filen till byte-array och skickar tillbaka det till Front-end. Varför vi valde att skicka med den typen var för att klientens nedladdningsimplementation var enklare att tolka med byte-array.

### Front-end

Resultatet blev som i figur 6. Gränssnittet för att välja fil är webbläsar- och operativsystembaserat vilket är en standard som finns i html. Listan med filer skapas när servern skickar alla uri:er som klienten sedan formaterar om till filnamnet. Sedan skapade vi en funktion som kollar vad det är för filtyp och väljer rätt ikon för rätt fil.

## Varning ifall workorder nummer inte är ifyllt

Ifall en användare har rollen “invoice” ska denna få en varning ifall den försöker lämna vyn för en “Godkänd kurs” utan att ha fyllt i alla workorder nummer och sedan tryckt på knappen “Ifylld av invoice”.

Detta behövdes då en med rollen invoice inte har många uppgifter i appen och lätt att glömma trycka på knappen. Är knappen ej tryckt kan de med andra roller inte jobba vidare med kurstillfället.

### Front-end

Först började vi reda ut hur en användare kan lämna sidan. Det kan göras genom att gå vidare i applikationen, genom att stänga fönstret, gå tillbaka till föregående sida eller skriva in en annan sökväg.

För att kolla ifall en användare försöker gå vidare i applikationen implementerade vi “CanDeactivate<CourseInstanceOverviewComponent>” i våran RoleGuard. Utifrån denna kollar vi om kursen är i stadiet “Godkänd” och ifall användaren är en “Invoice” samt ifall de tryckt i knappen “ifylld av invoice”.

Är knappen inte klickad kommer en varningsruta upp och påminner om att trycka i knappen ifall allt är klart.

Ifall en försöker lämna sidan utanför applikationens vägar behövs en annan lösning. Vi kan lyssna på webbläsaren som meddelar ifall håller på att rensas. Detta kan göras med hjälp av ”@HostListener” som kan lyssna på DOM-events. I detta fallet ”beforeunload”. När detta event triggas och alla kriterier är mötta visar vi återigen varningsrutan.

## Testmiljö

Vi satte upp en server på AWS som vi använde som testmiljö. Denna kunde vi nå genom att ssh:a in och ladda upp våran applikation. Uppsättningen av servern gjordes via ett alten-konto som vi inte har tillgång till så vi skriver endast om hur vi kan skeppa applikation till den.

Först bygger vi front-end via Angular CLI och skriver kommandot:

ng build --base-href=/training/ --prod --configuration test --aot --output-path=../src/main/resources/static

När detta är byggt bygger vi en .jar med både front-end och back-end inkluderat. Detta görs via Maven-kommandot:

mvn install

Denna .jar kopierade vi sedan in på servern. För detta använde vi oss utan WinSCP. Med WinSCP kan vi logga in på servern med våran SSH nyckel och komma åt filsystemet och kopiera in de filer som behövs. För att sedan dra igång applikationen gjorde vi ett skript, deploy.sh.

sudo service training stop && sudo cp ~/training.jar . && sudo service training start

För att köra skriptet använde vi oss utav programmet Putty. Med Putty kommer vi åt servern och kan då köra skriptet som stoppar den tidigare versionen. Den nya versionen kopieras in i rätt mapp och startas sedan.

## Kundkontakt

Vi har haft möten med kund nästintill varje vecka. Med på mötena var en eller två representanter från training-avdelningen här på Alten. De som kommer använda applikationen mest. Men för att applikationen ska vara värdefull krävs att andra delar av Alten använder applikationen, t.ex. Invoice-avdelningen. Vi hade även några möten med dessa.

På mötena har vi visat upp vad vi gjort sedan sist. Vi diskuterar om det blev som de tänkt, om allas behov är mötta samt vad som ska prioriteras härnäst.

Efter mötena har vi suttit med handledare och gjort nya ärenden i Jira utifrån information på mötena. Mot slutet av projektet började vi även estimera hur komplicerad ett ärende kommer vara. För att sedan kunna utvärdera hur många Story Points vi hinner med under en sprint och kan då lättare planera hur mycket vi ska ha med i nästkommande sprint.

# Avslutning

## Resultat

## Slutsats

**Avslutningsdel** OBS! Detta är inte någon överförbar rubrik!

Avslutningen bör hänga väl samman med inledningen. Syfte och mål som formulerats i inledningen återkommer och besvaras i avslutningsdelen. Lämpliga rubriker är:

**Resultat:**

Sammanfatta här arbetet/projektet och ange slutresultat och huruvida målet/målen uppnåts.

*Denna del redovisar fakta och innehåller inga subjektiva kommentarer.*

**Slutsats**

Drag slutsatser baserade på resultatet. *Denna del kan vara mer värderande och subjektiv.*

**Eventuell ordlista**

Förklaringar av facktermer, initialord, utländska uttryck och speciella förkortningar.

**Källförteckning**

# Bilagor

## Kod

### Säkerhet och behörighet

#### Avkodad autentiseringsnyckel

{

"sub": "admin@admin.se",

"auth": [

{

"authority": "ROLE\_NATIONALMANAGER"

}

],

"iat": 1554712393,

"exp": 1555576393

}

### Sålda kurser

#### getSoldCourseInstances



### Bifoga dokument

#### StorageService spara



#### StorageService ladda ner



#### StorageService lista alla filer



### Pingfunktion

#### Application properties





#### MailService

#### 

Detaljerade beskrivningar som bara intressar en midre grupp av de tänkta läsarna. Numreras och ges egna rubriker enligt “Bilaga 1: Ritning över X”

* Tabeller, grafer,mätresultat eller annat viktigt underlag.

Lektion med anders:

Rapport skall vara inskickad en veckan innan redovisning.

27-28 för redovisning som senast. Kanske att det sker veckan innan.

Kod skall bifogas och hänvisas till.

1. <https://docs.spring.io/spring-security/site/docs/5.2.0.BUILD-SNAPSHOT/api/> [↑](#footnote-ref-2)
2. <https://jwt.io/> [↑](#footnote-ref-3)
3. <https://docs.spring.io/spring-security/site/docs/5.2.0.BUILD-SNAPSHOT/api/org/springframework/security/config/annotation/web/configuration/WebSecurityConfigurerAdapter.html> [↑](#footnote-ref-4)
4. https://docs.spring.io/spring-security/site/docs/5.2.0.BUILD-SNAPSHOT/api/org/springframework/security/core/GrantedAuthority.html [↑](#footnote-ref-5)
5. <https://www.devglan.com/spring-security/jwt-role-based-authorization> [↑](#footnote-ref-6)
6. <https://medium.com/@ryanchenkie_40935/angular-authentication-using-route-guards-bf7a4ca13ae3> [↑](#footnote-ref-7)