

Slutrapport

Arkitektur och design av globala applikation, IV1201

Evan Saboo

saboo@kth.se

2018-03-18

- Klient kod: <https://github.com/15.sys.kth.se/GlobalaApplikationer/RecruitmentClient>
- Server kod: <https://github.com/15.sys.kth.se/GlobalaApplikationer/RecruitmentServer>
- Fristående klient: <https://github.com/15.sys.kth.se/GlobalaApplikationer/RecruitmentSAClient>
- Databas migration: <https://github.com/15.sys.kth.se/GlobalaApplikationer/DatabaseMigration>

1 Introduktion

Projektet går ut på att utveckla ett rekryteringssystem. Systemet ska kunna skilja mellan två typer av användare, en jobbsökande och en rekryterare. En ansökande ansöker om ett jobb som erbjuds av företaget som äger systemet medan en rekryterare är anställd av företaget för att hantera ansökningar.

I Systemet ska en jobbsökande kunna registrera arbetsansökningar och rekryterare ska man kunna administrera ansökningarna.

Systemet ska följa nedanstående kraven:

- **Robusthet:**
 - Flexibel och lättförståelig kod.
 - Lämplig lagrad Arkitektur.
 - Javadoc kommentarer ska finnas för alla publika definitioner.
- **Säkerhet:**
 - Auktorisering
 - Autentisering
 - Loggning
- **Transaktioner:**
 - Det ska vara tydligt hur vart transaktioner börjar och slutar i systemet.
- **O/R Mappning:**
 - Databasen ska ha ett lämpligt sätt att generera primära nycklar.
 - Relationer i databasen ska vara tydliga och översättas på ett lämpligt sätt till relationer i systemet.
- **Felhantering:**
 - Hantering av olika "runtime" undantag ska utföras på lämpligaste sätt och loggas vid behov.
- **Internationalisering och lokalisering:**
 - Webbssidor i systemet ska både internationaliseras och lokaliseras.
 - Databasen ska också internationaliseras och lokaliseras.
- **Bearbetning:**
 - All funktionalitet i systemet ska va relaterat till kundens krav. Det ska inte finnas någon kod i programmet som gör något orelaterat till systemet huvudsakliga funktionaliteter.
 - Frekventa integreringar.
 - All kod och dokumentation ska lagras i ett enda förråd av ett versionskontrollsystem.
 - Byggverktyget Maven ska användas i projektet.
 - Slutprodukten ska levereras till kunden på ett lämpligt sätt.

- En till utvecklingsverktyg ska användas i hela projektet.
- **Testning:**
 - Acceptans tester ska utföras med testverktyget Selenium.
- **Annan funktionalitet:**
 - Data som skriv in i HTML ska valideras.
 - HTML dokument ska delas upp i header, footer, navigationsmeny, och huvudinnehåll så att de kan återanvändas i andra HTML sidor.
 - En fristående klient som ska kommunicera med servern med hjälp av webbtjänsten JAX-RS.
 - En PDF dokument ska kunna generas för jobbapplikationer.
- **Andra icke-funktionella krav:**
 - Tillgänglighet – Servern ska vara robust nog för att hantera flera klientanrop och vara tillgänglig dygnet runt.
 - Non-repudiation – En användare måste validera sina uppgifter innan hen kan göra en viktig operation.
- **Extra funktionalitet:**
 - Implementera Hypertext Transfer Protocol Secure (HTTPS) för att kryptera transport av data mellan servern och klienten.

Jag utförde projektet med:

Emil Nordin – emnordi@kth.se

Oscar Härmäläinen - oham@kth.se

2 Litteraturstudie

Projektet utfördes med hjälp av dessa källor:

- Vi använde oss av Oracles hemsida för att förstå hur man kan skapa en enkel JAX - RS applikation med java EE.¹
- Vi använde StackOverflow hemsidan för att få hjälp med allmänna problem och för att få korta förklaringar på programmeringssyntaxer.²
- Java EE dokumentationshemsidan var också hjälpsam för att hitta Java EE syntaxförklaringar.³
- I ID1212 Nätverksprogrammeringskursen finns det föreläsningar om Applikation Server som var bra hjälp för att förstå Java EE som helhet och dess funktionaliteter.
- Då Selenium och TestNG användes som testverktyg behövde vi först gå igenom deras officiella dokumentationer.^{4 5}
- För att implementera HTTPS behöves ett certifikat. Vi använde oss av Payaras officiella guide för att skapa och lägga in certifikat i en Payara Server.⁶ Fjärde föreläsningen i kursen användes för att konfigurera HTTPS i webbklienten.
- I StackOverflow följde vi en guide som beskrev hur man kunde implementera autentisering och auktorisering.⁷
- Vi använde oss av Oracles dokumentation för att hitta syntaxer och förklaringar inom Java Server Faces (JSF).⁸
- För att implementera loggning och loggfiler behövde vi gå igenom Oracles dokumentation om Java loggning.⁹
- Med hjälp av en guide i hemsidan vogella.com kunde vi implementera PDF generering.¹⁰

¹ <https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cp0n/index.html>

² <https://stackoverflow.com/>

³ <https://javaee.github.io/javaee-spec/javadocs/>

⁴ <http://testng.org>

⁵ <https://www.seleniumhq.org/docs/>

⁶ <http://blog.payara.fish/securing-payara-server-with-custom-ssl-certificate>

⁷ <https://goo.gl/K4N5Zh>

⁸ <https://docs.oracle.com/javaee/6/javaserverfaces/2.1/docs/vldocs/facelets/>

⁹ <https://docs.oracle.com/javase/7/docs/api/java/util/logging/package-summary.html>

¹⁰ <http://www.vogella.com/tutorials/JavaPDF/article.html>

3 Metod

I början av projektet diskuterade vi först vilka krav systemet skulle uppfylla och vilka funktionaliteter den skulle ha för att uppnå kundens kravspecifikation. Det var viktigt för oss att planera vilka krav som skulle utföras först för att vidareutveckla dem under projektet gång. Vårt mål var att bli klara med de obligatoriska kraven innan vi började med de extra kraven.

Det var också viktigt att välja rätt kommunikationsparadigm i början av projektet för inte behöva ändra om senare. Vi tänkte från början gå med Websockets eftersom vi hade tidigare erfarenhet med den men vi valde till slut JAX-RS. Anledning till beslutet var för att vi ville lära oss om hur RESTful webbtjänster fungerade. Med JAX-RS kunde vi också implementera en fristående klient utan att behöva lägga till kod i REST servern.

Efter att vi hade en fungerade server-klient REST kommunikation delades hela arbetet i tre delar. En fick implementera inloggning och registrering, en annan tog hand om funktionaliteter för ansökande och den sista personen fick göra funktioner för rekryterare. Medan varje person tog hand om sin del kunde man även implementera några krav som hängde ihop med arbetet, såsom auktorisering och autentisering, internationalisering och lokalisering, validering av HTML data och uppdelning av HTML dokument.

Netbeans var vår första val som utvecklingsmiljö eftersom alla i gruppen hade använt verktyget förut. Vi använde också utvecklingsverktyget Apache Maven eftersom den redan var integrerad i Netbeans och vi hade tidigare erfarenheter med den. Ett annat utvecklingsverktyg som vi tog nytta av var FindBugs Code Analyzer. Verktöget användes för att hitta potentiella problem och upptäcka inkonsekvenser i projektets källkod.

Det var ganska lätt att installera och använda verktyget eftersom man kunde installera den som plugin i Netbeans. Med FindBugs kunde vi också hitta alla metoder och klasser som var okommenterade, vilket ledde till att vi fick kommentera allting enligt kravspecifikationen.

Versionshanteringssystemet Git användes för att kunna sammanfoga kod från varje gruppmedlem. För att varje medlem ska kunna komma åt den sammansatta koden användes webbhotellet Github. Från början var det lite svårt att förstå hur man slog ihop kod i Github och det blev ganska stökigt med att använda Githubs egna applikation för att hämta och skicka kod från/till Github. Vi bestämde oss till slut för att använda Netbeans egna Git integration eftersom den visste vilka filer som skulle skickas och tas emot från Github. En person fick hantera sammanfogandet av kod och fick testa om allting fungera som det ska.

För att kunna köra applikationen i webben behövdes en applikationsserver. Vi använde Payara Server, en förbättrad version av Oracles GlassFish Server.

Leverans av slutprodukten:

I Slutet av projektet var det viktigt att bestämma hur slutprodukten skulle leverans till kunden. Vi bestämde oss direkt för att själva hantera och hosta systemet i vårt egna webbhotell eftersom vi kan lättare hantera serverfel och vidareutveckla systemet om kunden vill göra det.

4 Resultat

När du gå in i hemsidan presenteras du med startsidan [Figur 1]. I sidan finns lite information om hemsidans syfte och man kan trycka på "Login/Register" knappen för att komma till inloggning/registreringssidan. I inloggningssidan kan man antingen logga in som ansökande eller rekryterare, man kan också registrera sig som ansökande [Figur 2].

När en ansökande loggar in blir hen omdirigerad till jobbansökningssidan [Figur 3]. Där kan man skicka en ansökan genom att välja kompetenser och/eller välja tillgängligheter. Innan man skickar in ansökan måste man först skriva in lösenordet för att inte "råka" skicka in ansökan.

Om man loggar in som rekryterare blir man omdirigerad till sökningssidan för alla ansökningar [Figur 4]. Man kan söka efter ansökningar med olika sökparametrar såsom registreringsdatum, tillgänglighetsperiod, kompetens och ansökandens förnamn. Om man inte vill ha en sökparameter i sökningen behöver man bara låta den vara tom. Efter att sökningen är klar visas en tabell med alla sökresultat. Där kan man se översikt över ansökningarna och välja att se mer information om en specifik ansökning. Efter att man har valt en ansökan blir man omdirigerad till ansökningsöversiktssidan [Figur 5]. Där kan man se all information om ansökan och ändra dess status, men man måste skriva in lösenordet för att bekräfta ändringen.

Om något fel sker i klienten eller om en användare försöker komma åt en obehörig sida kommer hen bli omdirigerad till feluppsvisningssidan [Figur 6].

Nedan kommer jag gå igenom alla krav som vi har uppfyllt:

Robusthet:

Flexibel och lättförståelig kod:

För att göra koden flexibel har vi skapat klasser och metoder som bara gör vad det ska. Vi har också försökt dela upp kod i flera metoder för att det ska vara återanvändbart. Om man till exempel kollar på Authentication klassen i klienten ser man att det bara finns metoder som är relaterad till autentisering. I ApplyManager klassen finns det bara kod för att skapa en ansökan och i ApplicationListing klassen finns det bara metoder för att söka efter ansökningar. I Serven har koden också delats upp i klasser som bara har relation till deras funktionaliteter. Om man kollar på alla REST slutpunktsklasser ser man att de bara tar emot förfrågor som är relaterade till klassen.

För att också göra koden mer lättförståelig har vi kommenterat alla publika definitioner. Detta gör att vi kan skapa javadoc vilket kan användas för att hitta dokumentation till klasser eller metoder i koden.

Nedan förklaras lite om varje ramverk som användes i projektet:

Bootstrap är ett ramverk för HTML, CSS, och JavaScript som man kan använda som grund för att skapa snygga och responsiva hemsidor. Med ramverket kunde vi enkelt designa vår hemsida utan att behöva mecka mycket med CSS och JavaScript.

JSF tillåter webbutvecklare att bygga användargränssnitt för JavaServer-applikationer och den stöds av webbservrar som kör Java EE. Med ramverket förenklas skapandet av webbapplikationer genom att tillhandahålla en standarduppsättning av verktyg för att bygga användargränssnitt. Till exempel kan en webbutvecklare kalla på en enkel JSF-funktion som generar ett webbformulär istället för att koda **formuläret** i HTML. En annan JSF-funktion kan användas för att bearbeta de data som användaren har angett. Dessa funktioner behandlas på servern och den resulterande data matas ut till klientens webbläsare.

EJB ramverket används för att utveckla skalbara, robusta och säkrare företagsapplikationer i Java. EJB erbjuder middleware-tjänster såsom säkerhet, transaktionshantering och databasoperationer (med hjälp av JPA).

Java Persistence API är en samling av klasser och metoder för att kontinuerligt lagra stora mängder data i en databas. Med JPA kan man lagra, uppdatera och ta bort javaobjekt i databasen utan att behöva tänka SQL frågor och databasstruktur.

Lagrad arkitektur:

Både Servern och klienten har delats upp i flera lager.

Servern följer MVC arkitekturen vilket börjar från Net, sedan går till Controller, sedan till Model och till sist Integration. Eftersom servern är statslös har vi en Net lager istället för View där alla REST slutpunkter finns. Controller lagret hanterar anrop mellan Net, Model och Integration. I Model lagret hanteras data och logik vilket bl.a. är entiteter, autentisering, auktorisering och Dataöverföringsobjekt (DTO klasser). Integrationslagret hanterar all kommunikation med databasen, dvs. alla anrop som görs till databasen får bara göras från integrationslagret.

Klienten har fyra lager, View, Model, RestCommuncation (Net) och Logger. View hanterar data som ska visas för klienten, Model har alla DTOs och en klass för att hantera hemsidans språk, och RestCommuncation har metoder för att kommunicera med servern via REST. Logger lagret har en klass som heter LoggHandler för att kunna logga undantag. Vi tyckte inte att Controller lagret behövdes eftersom det inte skulle tjäna något syfte i klienten.

Transaktioner:

I Servern startar transaktioner från Controller klassen för att utföra transaktioner i integrationslagret. Vi behöver transaktioner för att lagra och hämta data från databasen. Om något går fel i en transaktion görs ett "transaction rollback" vilket gör att transaktionen avbryts och databasen går tillbaka till ett tidigare tillstånd.

Säkerhet:

Autentisering:

För autentisering följde vi en guide i Stackoverflow som förklarade hur man implementerade token-baserad autentisering. Det som är bra med token är att den är statslös, dvs. användardata hämtas genom att kolla på användarens token. Man kan säga att den betar sig som en användarsession mellan servern och klienten. Tokens kan också användas för att lägga till flera serverar utan att behöva hålla koll på vilken server som hanterar specifika användarsessioner då tokens är statslösa och kan kommas åt av flera serverar från databasen.¹¹

¹¹ <https://stormpath.com/blog/token-authentication-scalable-user-mgmt>

Nedan förklaras steg för steg hur token autentisering går till:

En användare skriver in användaruppgifter i klientgränssnittet för att kunna bli inloggad i sitt konto. Uppgifterna skickas till REST servern för att kontrollera om de är giltiga. Om uppgifterna är giltiga generas en giltig token i TokenGenerator klassen och skickas tillbaka till användaren, samtidigt sparas en token i databasen med giltighetstid för framtida autentiseringar.

Varje gång klienten utför en förfrågan skickas den sparade token med i förfrågan för att kunna valideras i servern. Valideringen sker i AuthentitactionFilter klassen vilket sedan ger två möjliga svar till klienten:

- Om valideringen lyckas skickar servern ett accepterat svar till klienten. Servern uppdaterar också tokens giltighetstid för att den ska vara giltig under längre tidsperiod.
- Om den inte lyckas skickar servern istället tillbaka en "nekad förfråga" meddelande.

Auktorisering:

I autentiseringsguiden fanns också instruktioner för hur man implementerar roll-baserad auktorisering.

Vid varje användarspecifik klientförfråga kollas också om användaren har rätt roll för att förfrågan ska bli accepterad. Valideringen utförs i AuthorizationFilter klassen genom att kolla i databasen om den specifika token har samma roll som klassen/metoden som klienten vill komma åt. I REST servern har vi olika rest slutpunkter som klienten kan komma åt. Varje slutpunkt är kopplat till en klass som kan ha en @Secured tagg följt med en angiven Enum roll. Taggen är definierad i AuthorizationFilter klassen och ska kunna användas för att beskriva vad en slutpunkt ska ha för roll för att auktorisera den. I servern har vi definierat två roller, jobbsökande och rekryterare. Genom att tagga REST slutpunkter med @Secured kan då en rekryterare bara komma applikationssöknings- och ansökningsöversiktsmetoder medan en jobbsökande bara kan skapa och ändra en jobbsökning.

Felhantering och Loggning:

I Servern hanterades alla undantag i klassen ApplicationExceptionMapper. Vi tar emot allt från användaruppgifter till alvarliga databasundantag, vilket gör att vi kan hantera alla undantag på samma ställe och logga dem vid behov. Vid varje serverundantag skickas ett felmeddelande till klienten för att informera användaren om felen. Vid loggning

används Javas egna logger bibliotek för att logga nödvändiga undantag sedan spara i en av tre definierade loggning filer beroende på vilken nivå undantaget har. Vi hanterar tre nivåer av undantag, där nivån INFO används för att logga felaktiga användarinmatningar från klienten, WARNING är till alla förfrågningar som misslyckas i servern och SEVERE till misslyckad databasuppkoppling och annat okänt undantag.

Klienten har samma loggningssystem som servern. Det som loggas är alla undantag som kan ske i klienten och alla svar från servern som behöver loggas, tex. http felkod 500 eller 403.

O/R Mappning:

Från början tog vi alla tabeller och relationer från kundens kravspecifikation men vi lade också till en tabell som sparade alla jobbansökan för att kunna användas för sökning av alla ansökningar. Med Netbeans kunde vi autogenera entitetsklasser för att JPA ska synka med databasens tabeller. Relationerna var också med i entiteterna vilket gjorde att vi kunde hämta relationsdata direkt från databasen utan att behöva göra någon relationsjämförelse i JPQL (Java Persistence Query language) satserna.

Primära nycklar i varje tabell autogeneras med hjälp av JPA GenerationType, vilket gör att vi inte behöver skapa primära nycklar manuellt. Detta gör att vi inte behöver oroa oss över att behöva kontrollera om en id redan finns i tabellen innan vi kan lagra nya data. Varje entitet har en ID kolumn och identifieras som primär nyckel. Tabellen som lagrar användaruppgifter har också andra unika kolumner som måste kontrolleras innan nya data kan sättas in. Ett exempel är vid registrering där personnummer och användarnamn måste kontrolleras först med databasen innan användaren kan registrera sig.

Internationalisering och lokalisering:

I varje webbsida förutom feluppvisningssidan finns det två knappar som gör att man kan ändra språk på innehållet i hemsidan. Om man byter på språket och går till en annan sida kommer det angivna språket fortfarande kvar. Även felmeddelanden och data från databasen (kompetenser) har översatts för att internationalisera hela hemsidan. Hemsidans standardspråk är engelska för att användare från andra länder ska kunna förstå sidans innehåll och för att engelska är ett allmänt språk.

I Klienten finns klassen LanguageChange som har metoden changeLanguage för att ändra hemsidans språk till användarens val. När språket ändras hämtas alla

översättningarna från en "properties" fil och visas upp för användaren. I klienten finns det två "properties" filer, en för svenska och en för engelska.

Även datumen i alla inmatningsområden är internationaliserade eftersom olika länder har olika datumformat.

I databasen finns en tabell som har information på vilka språk som stöds i hemsidan. I kompetens- och statustabellen finns en relation till språktabellen för att man ska kunna hämta rätt språk från tabellerna. När till exempel alla kompetenser ska visas för användaren kollas först vilken språk hemsidan är satt till och sedan hämtar alla kompetenser som har den angivna språket, språkändringen görs direkt i klienten.

Testning:

Det är viktigt att alla funktioner i systemet ska kunna användas utan något undantag, därför skapade vi 12 tester i Selenium för att testa dem. Testerna exekveras enligt följande ordning:

1. Gå till registreringssidan.
2. Testa att registrera en ny användare med ogiltiga användaruppgifter och sedan giltiga. (Användaruppgifterna autogeneras för att kunna registrera flera användare vid längre testperioder)
3. Testa logga ut användaren efter lyckad registrering.
4. Testa logga in som med de generade användaruppgifterna.
5. Gå till varje sida i hemsidan och kolla om det finns en sida som användaren inte har tillgång till.
6. Gå till ansökningssidan och försök skapa en ansökan med felaktigt data och sedan med korrekt data.
7. Logga ut användaren från kontot.
8. Logga in som en rekryterare.
9. Gå till varje sida i hemsidan och kolla om det finns en sida som rekryteraren inte har rättighet att komma in i.
10. Gå till applikationssökningssidan och testa att söka efter ansökan som nyligen hade registrerats.
11. Få upp information om ansökan och kontrollera om alla uppgifter stämmer överens med ansökningen som registrerades i punkt 6. Testa sedan att acceptera och neka ansökan flera gånger.
12. Logga ut som rekryterare.

Vid registrering av en ansökan måste användaren skriva in hens lösenord för att lämna in den. Samma funktionalitet implementerades när en rekryterare ska byta ansökans

status. Vid testningen kontrollerades även om man behövde skriva in lösenord vid fallen ovan.

Målet med acceptanstesterna var att vi skulle kontrollera om alla krav från kunden fungerade även vid längre testperioder, samtidigt ville vi kolla om servern kunde hantera alla förfrågningar. Därför körde vi en 24 timmars extensiva Selenium tester för att vara säkra på att kraven uppfylldes utan problem.

Med testningsverktyget TestNG kunde vi köra alla Selenium tester på två webbläsare parallellt för att illustrera två användare som konstant ska använda alla funktioner i 24 timmar. Efter 24 timmars av testande visade resultatet att vi totalt fick bara 7 misslyckade tester [Figur 7]. Det som hände var att en test misslyckades vilket gjorde att de resterande testerna också misslyckades eftersom varje test är beroende på deras förgående test. Man kan inte garantera att alla tester lyckas varje gång pga. Selenium bugg, dålig felhantering av testerna eller renderingsfel vid laddning av en sida.

Med resultatet kom vi komma fram till att systemet är stabilt nog för att kunna användas dagligen utan några interna serverfel. Vi kan också garantera att implementerade funktioner fungerar som det ska.

Hela testresultatet finns i klientens Github repository.¹²

Annan funktionalitet:

Validering av HTML data:

I inloggning- och registreringsformuläret valideras data med hjälp av JSF. Om man ex. låter fälten vara tomma kommer JSF att visa ett felmeddelande [Figur 8, Figur 9]. Om alla JSF valideringar lyckas kommer uppgifterna skickas till servern och kontrolleras om tex. användarnamnet och lösenordet är korrekt vid ett inloggningsförsök. Om valideringen i servern misslyckas skickas ett felmeddelande tillbaka till klienten vilket sedan visas för användaren. För registrering valideras användarnamnet och personnumret i servern för att inte ha duplikat data i databasen.

Uppdelning av HTML dokument:

Med JSF kunde vi dela upp alla html komponenter för att kunna återanvända dem i andra sidor. Vi har en xhtml sida (masterTemplate.xhtml) som fungerar som en mall för

¹² <https://goo.gl/qjV5VD>

alla andra sidor förutom feluppsvisningssidan. I alla sidor har titeln, head, header, navigationsmenyn och footer definierats i masterTemplate sidan och det är bara body sektionen som är annorlunda mellan sidorna.

Fristående klient:

Vi utvecklade också en JavaFX klient som kommunicerar med servern via JAX-RS. Vi hann bara implementera inloggning och registrering [Figur 10] eftersom vi inte hade mycket tid på oss att implementera flera funktioner, därför är fristående klienten bara en prototyp. Det finns tre lager i prototypen, View, Model och Rest (Net). View lagret hanterar alla interaktioner med användaren såsom texter, knappar och Inmatningsområden. Model innehåller två klasser för inloggning och registrering som används i Rest lagret för att skicka objekt av klasserna till servern.

Inloggning och registrering har också valideringsmekanismer. När en användare loggar in kontrolleras om användaruppgifterna är korrekta genom att låta servern hantera valideringen. Vid registrering hanterar servern också valideringen av registreringsuppgifterna och skickar tillbaka ett svar till klienten som Response objekt.

Generering av PDF dokument:

I application_overview.xhtml sidan finns en knapp för att generera en PDF av ansökan som visas i sidan. Klienten skickar en förfrågan till servern om vilken ansökning som ska genereras som PDF med hjälp av ansökningens ID och vilket språk innehållet ska vara på, vilket tas från hemsidans valda språk. Servern kallar på PDFGenerator klassen som i sin tur hämtar all data om ansökan från databasen. Integrationslagret skickar tillbaka ett objekt av ApplicationDetailsDTO till PDFGenerator klassen och då konverteras all data i objektet till en PDF, vilket sedan skickas tillbaka till klienten och visas för rekryteraren.

I Figur 11 visas hur en genererad PDF på en ansökning ser ut.

Andra icke-funktionella krav:

Tillgänglighet:

I **testningsdelen** förklaras hur vi bevisade att servern är tillgänglig under längre testperiod.

Non-repudiation:

När en användare ska skicka sin ansökan måste hen verifiera lösenordet. En rekryterare måste också skriva in lösenordet för att kunna ändra på status i jobbansökan. Detta har implementerats för att man ska ha ett extra lager av säkerhet och för att inte "råka" utföra en kritisk operation.

Extra funktionalitet (HTTPS):

Eftersom vi använde tokens för att hantera användarsessioner behövde vi kryptera den varje gång den skickades till servern. Detta löstes med att implementera HTTPS i hemsidan. Vi följde en guide i Payara hemsidan för att skapa ett certifikat och lägga in den i Payara Servern. Vi behövde också ändra i klientens web.xml för att kryptera GET och POST metoderna, vilket vi utförde med hjälp av föreläsningarna i kursen. Sedan var det bara att starta Payara Servern och gå till hemsidan via port 8181 (default HTTPS port i Payara). Vi fick alltid meddelande av webbläsaren att certifikatet inte är trovärdig eftersom den är självsignerad, annars skulle webbläsaren acceptera ett betalt certifikat.

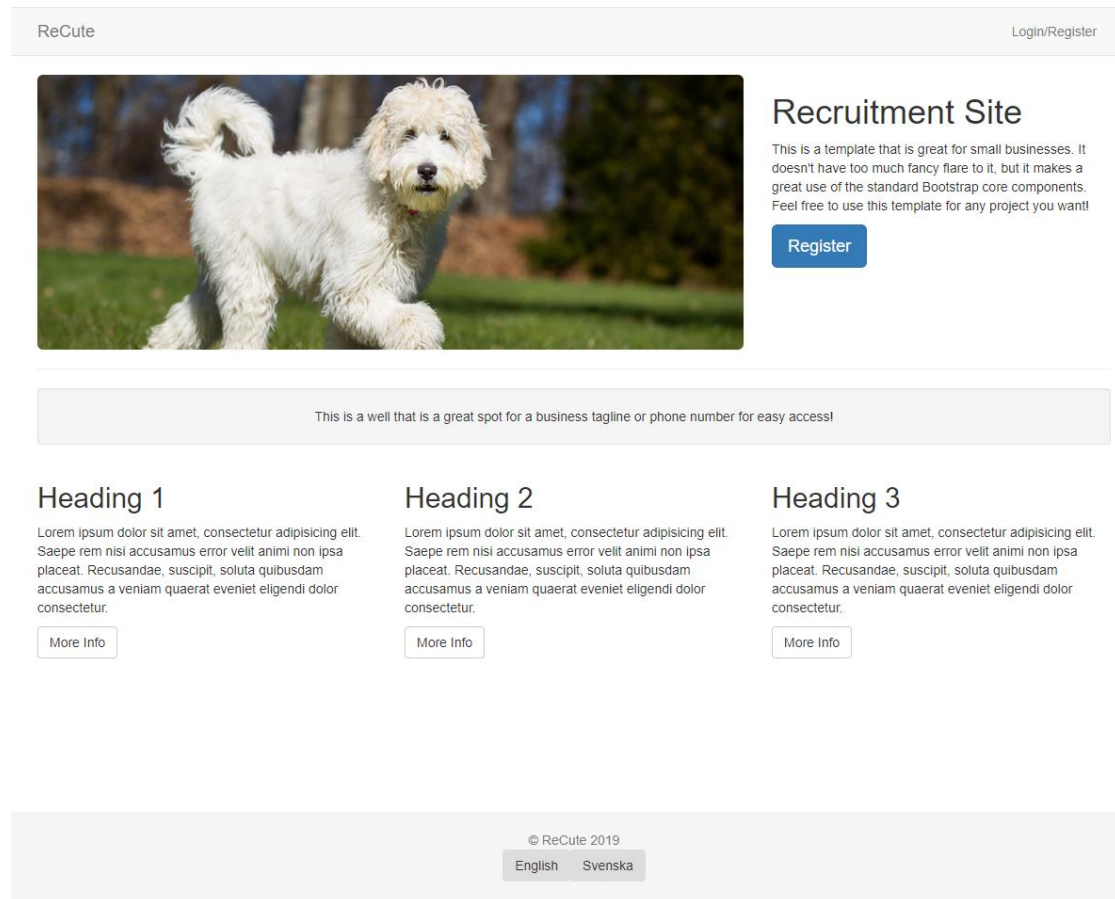
Databasmigrering:

I kundens krav behövde man migrera den gamla databasen från deras gamla system till den nya som finns i nuvarande projektet. Därför skapade vi ett script som migrerar data från en gammal testdatabas till en ny databas som har samma struktur som vår egna databas i projektet.

Migrationen börjar med att sätta in data som den nya databasen behöver såsom vilka språk hemsidan stödjer, vilka status en ansökan kan ha och vilka kompetenser som kan väljas. Sedan hämtas objekt från den gamla databasen till den nya i ordning för att kunna hantera undantag manuellt om något går fel med migrationen. Vid undantag stoppas migrationen och då får man se vad som gick fel för att kunna fixa det. Om allt går som det ska skapas tabellerna i den nya databasen och all data sätts dit.

Getters och setters kommenterades inte eftersom migrationsprogrammet behöver bara köras en gång (om allt lyckas) och metodnamnen är självförklarande.

Figurer



Figur 1: Startsidan

The mockup shows a web interface for 'ReCute'. At the top, a header bar contains the 'ReCute' logo on the left and a 'Login/Register' link on the right. Below the header, there are two main forms. The first form, located at the top center, is for login and registration. It features input fields for 'Username' and 'Password', a 'Login' button (circled in red), and a 'Register' button. The second form, located below the first, is titled 'Register to use ReCute'. It contains several input fields: 'First name', 'Surname', 'Email', 'Username', 'Password' (twice), and 'Social security number'. It also has a 'Register' button and a 'Cancel' button. At the bottom of the page, there is a footer bar containing the copyright notice '© ReCute 2019' and language selection buttons for 'English' and 'Svenska'.

ReCute

Login/Register

Username

Password

Login

Register

Register to use ReCute

First name

Surname

Email

Username

Password

Password

Social security number

Register

Cancel

© ReCute 2019

English Svenska

Figur 2: Inloggning-/Registreringssidan

ReCute

evan ▾

Create New Job Application

1. Add Competence

Competences

Choose competence... ▾

Years of Experience

Years of experience ▾

Add Competence

Added Competences

Competence	Competence ID	Years of Experience	Remove
milk	3	0.75	Delete
popcorn	2	1	Delete

2. Add Availability

Available From

MM/dd/yy (Ex: 03/15/18)

Available To

MM/dd/yy (Ex: 03/15/18)

Add Availability

Added Availabilities

Available From	Available To	Remove
3/15/18	3/15/18	Delete

Password

Submit Application

© ReCute 2019

English Svenska

Figur 3: Sida för att skicka en ansökan.

ReCute
recruiter

Search

Application Submission Date

MM/dd/yy (Ex: 03/15/18)

Available From

MM/dd/yy (Ex: 03/15/18)

Available To

MM/dd/yy (Ex: 03/15/18)

Competence

Any

Applicant Name

Sök

Applications

Show 10 entries

Search:

First Name	Surname	Email	Full Application Detail
AahnVpTQsPYSJ	wcAxVyFnAdyHG	pZAJh@pZAJh.com	View Application
AandQHTpXrJMH	IFirVTFNoUuXq	eqxLk@eqxLk.com	View Application
AAtsfBPpLGqJe	RLDwUgfmtdgB	HJuyM@HJuyM.com	View Application
AbdujHXdzApRP	pcLbzBplARJjs	fohIB@fohIB.com	View Application
abkcKfHaklvvd	MHvBhtvTrfkL	mMEEL@mMEEL.com	View Application
ABkKOBOWiQNe	lyHfHAEDnOldY	aTBnE@aTBnE.com	View Application
AbNFmrBOFRewZ	AqJPTDfmeqcSZ	wlVIX@wlVIX.com	View Application
ABUsPIJEzAAoe	CGdRbSbnbnwY	ZgjpN@ZgjpN.com	View Application
ACIONIGXxzAmD	ThwVogMDpCfC	WkjOF@WkjOF.com	View Application
aCjrmIXkVcHvC	PQFSdzttIOXAqw	NvxOz@NvxOz.com	View Application

Showing 1 to 10 of 2,599 entries

Previous 1 2 3 4 5 ... 260 Next

© ReCute 2019
English Svenska

Figur 4: Sida för att söka efter jobbansökningar.

ReCute

recruiter ▾

Job Application Overview

Generate PDF

Change Status

Password

Accept

Reject

Application Information

Registration Date	3/10/18
Current Status	Rejected

User Information

First Name	AahnVpTQsPYSJ
Surname	wcAxVyFnAdyHG
Email	pZAJh@pZAJh.com
Social Security Number	309182059737479

Competences

Name	Years Of Experience
rollercoaster	0.75
music	0.75

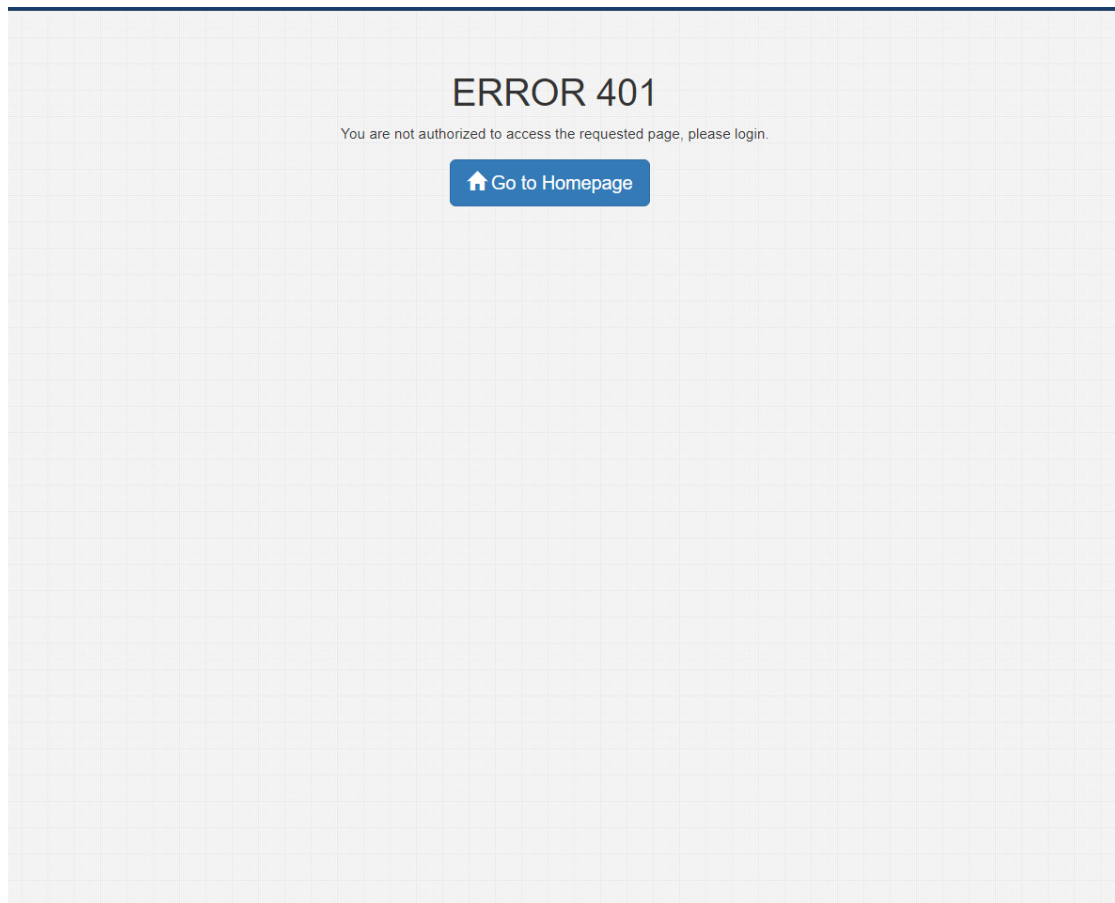
Availability

From	To
3/15/18	3/15/19
3/15/19	3/15/20
3/15/18	3/15/18

© ReCute 2019

English Svenska

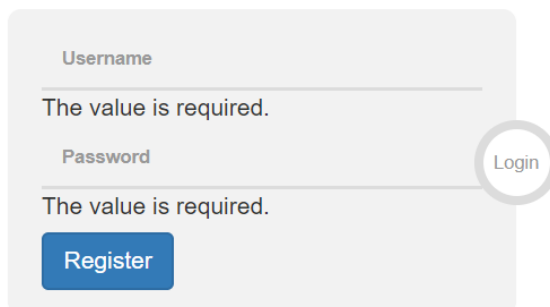
Figur 5: Översikt över en jobbansökan.



Figur 6: Feluppvisningssidan.

```
=====
Chrome Tests:
Total tests run: 15600
Total tests Passed: 15593
Total tests Failed: 7
Total tests Skipped: 0
=====
Firefox Tests:
Total tests run: 15600
Total tests Passed: 15600
Total tests Failed: 0
Total tests Skipped: 0
=====
All Tests:
Total tests run: 31200
Total tests Passed: 31200
Total tests Failed: 7
Total tests Skipped: 0
=====
The tests took 24 hours, 0 minutes and 19 seconds.
=====
-----
BUILD SUCCESS
-----
Total time: 24:00:20.813s
Finished at: Sun Mar 11 01:36:48 CET 2018
Final Memory: 7M/155M
-----
```

Figur 7: Resultat på 24 timmar av Selenium testning.



The image shows a login form with two input fields: 'Username' and 'Password'. Both fields have a red error message 'The value is required.' below them. To the right of the form is a circular 'Login' button. Below the 'Password' field is a blue 'Register' button.

Figur 8: Uppvisning av felmeddelanden vid tom inmatning.

Register to use ReCute

First name

First name

The value is required.

Surname

Surname

The value is required.

Email

test

Please enter a correct email-address

Username

Username

The value is required.

Password

Password

The value is required.

Password

Password

The value is required.

Social security number

abc

'abc' is not a SSN. Example:
9501011234

Register

Cancel

Figur 9: Felmeddelanden i registreringsformuläret.

The image shows a JavaFX application window titled "Welcome to ReCute". The main window has a light gray background. At the top center, it says "Welcome to ReCute" in a large, bold, black font. Below this, there is a "Login" button. Further down, there are two input fields labeled "Username" and "Password". At the bottom center, there is a "Register?" button. To the right of the main window, there is a smaller, light blue window titled "Enter Information:". This window contains a registration form with the following fields: "Username", "Password", "Password Again", "First name", "Surname", "Social security number", and "Email". Each field has a corresponding input box. At the bottom of this form is a "Submit" button.

Figur 10: Den fristående klienten.

From	To
03-15-2018	03-15-2019
03-15-2019	03-15-2020
03-15-2018	03-15-2018

24 (27)

5 Diskussion

Följande krav har uppfyllt:

- **Robusthet:**
 - Koden är flexibel och lättförstådd.
 - Vårt system följer MVC arkitekturen.
 - Alla public definitioner har kommaterats för Javadoc.
- **Säkerhet:**
 - Auktorisering
 - Autentisering
 - Loggning
- **Transaktioner:**
 - Transaktioner sker i servern mellan controller och integration lager.
- **O/R Mappning:**
 - Primära nycklar autogeneras av JPA.
 - Relationer i databasen är tydliga översätts korrekt till relationer i systemet.
- **Felhantering:**
 - "runtime" undantag hanteras och loggas vid behov.
- **Internationalisering och lokalisering:**
 - Webbssidor i systemet är både internationaliserade och lokaliserade.
 - Databasen är också internationaliserad och lokaliserad.
- **Bearbetning:**
 - All funktionalitet i systemet är relaterat till kundens krav. Det finnas inte någon kod i programmet som gör något orelaterat till systemet huvudsakliga funktionaliteter.
 - Kod från gruppmedlemmar slås ihop och testas för att försäkra om att inga konflikter ska ske.
 - All kod och dokumentation lagras i Github där alla gruppmedlemmar kan komma åt.
 - Byggverktyget Maven användas i projektet.
 - Slutprodukten hostas och hanteras av oss.
 - Code Analyzer används för att kontrollera all kod i hela systemet.
- **Testning:**
 - Acceptans tester har utförts med testverktyget Selenium.
- **Annan funktionalitet:**
 - Data som skriv in i HTML valideras av systemet.
 - HTML dokument har delas upp i header, footer, navigationsmeny, och huvudinnehåll.
 - En fristående klient har utvecklats som kommunicerar med servern med hjälp av webbtjänsten JAX-RS.

- Generering av PDF dokument för jobbansökningar har implementerats.
- **Andra icke-funktionella krav:**
 - Tillgänglighet – Serven är robust nog för att hantera flera klientanrop och kan vara tillgänglig dygnet runt.
 - Non-repudiation – En användare måste validera sina uppgifter innan hen kan göra en viktig operation.
- **Extra funktionalitet:**
 - Hypertext Transfer Protocol Secure (HTTPS) har implementerats för att kryptera transport av data mellan serven och klienten.

Under projektets gång skedde flera små och stora problem. Det första problemet vi stötte på var Rest kommunikationen mellan klienten och serven. Vi visste inte riktigt vad för länk man skulle använda för att komma åt en av serven rest slutpunkt, vi visste inte heller om servens ändpunkt fungerade som det ska. Vi var tvungna att testa flera länkar till vi kom fram till den rätta länken.

Ett annat stort problem var när en av oss skulle sätta ihop git grenar från varje gruppmedlem men eftersom det var så mycket ändringar i varje gren inte Github inte sammanfoga. Vi fick gå igenom alla konflikter manuellt vilket tog 1–2 timmar. Vi lärde oss av misstaget och började istället sammanfoga kod direkt efter att man var med en funktionalitet eller krav.

Själv har jag stött mest problem på Selenium testerna. Det var ganska jobbigt att testa hela tiden om ett html element kunde hittas av Selenium eftersom några element inte kunde hittas pga. okända orsaker. Jag var ibland tvungen att använda XML Path¹³ för att lösa problemet. Ett annat problem med Selenium var att testerna inte fungerade lika bra med Microsoft Edge jämfört med Chrome och Firefox. Jag försökte fixa testerna men det var för många för att hålla koll på och ibland gav Edge okända fel vilket gjorde att vi slutade använda webbläsaren för testerna.

Vad har jag lärt mig?

Jag har lärt mig minst något i varje krav som har uppfyllts men JAX-RS och grupparbetet är två saker som jag har lärt mig mest av. Med JAX-RS fick jag lära

¹³ <https://www.w3.org/TR/1999/REC-xpath-19991116/>

hur REST webbtjänster fungerar och hur man kan skapa fristående klienter för att kommunicera med en REST server.

Under projekts gång har jag lärt mig av problemen alla meddelar kan samtidigt stötta på och hur man kan lösa dem. Jag har också lärt mig om hur man ska planera ett sådant stort och tidskrävande projekt.

Det är viktig att planera och ta de rätta arkitekturella besluten för att man inte ska stöta på problem i projektet gång. Man ska alltid göra ordentlig förstudie innan man påbörjar med utvecklingen för att tex. inte fastna på saker som gör att man behöver göra om dem.

6 Kommentarer om kursen

Jag har spenderat ungefär 200 timmar åt kursen om man inkluderar föreläsningarna.